Predictive analytics and data mining

Charles Elkan
elkan@cs.ucsd.edu

May 28, 2013

Contents

Chapter 1

Introduction

There are many definitions of data mining. We shall take it to mean the application of learning algorithms and statistical methods to real-world datasets. There are numerous data mining domains in science, engineering, business, and elsewhere where data mining is useful. We shall focus on applications that are related to business, but the methods that are most useful are mostly the same for applications in science or engineering.

The focus will be on methods for making predictions. For example, the available data may be a customer database, along with labels indicating which customers failed to pay their bills. The goal will then be to predict which other customers might fail to pay in the future. In general, analytics is a newer name for data mining. Predictive analytics indicates a focus on making predictions.

The main alternative to predictive analytics can be called descriptive analytics. In a nutshell, the goal of descriptive analytics is to discover patterns in data. Descriptive and predictive analytics together are often called "knowledge discovery in data" or KDD, but literally that name is a better fit for descriptive analytics. Finding patterns is often fascinating and sometimes highly useful, but in general it is harder to obtain direct benefit from descriptive analytics than from predictive analytics. For example, suppose that customers of Whole Foods tend to be liberal and wealthy. This pattern may be noteworthy and interesting, but what should Whole Foods *do* with the finding? Often, the same finding suggests two courses of action that are both plausible, but contradictory. In such a case, the finding is really not useful in the absence of additional knowledge. For example, perhaps Whole Foods should direct its marketing towards additional wealthy and liberal people. Or perhaps that demographic is saturated, and it should aim its marketing at a currently less tapped, different, group of people?

In contrast, predictions can be typically be used directly to make decisions that

maximize benefit to the decision-maker. For example, customers who are more likely not to pay in the future can have their credit limit reduced now. It is important to understand the difference between a prediction and a decision. Data mining lets us make predictions, but predictions are useful to an agent *only if* they allow the agent to make decisions that have better outcomes.

Some people may feel that the focus in this course on maximizing profit is distasteful or disquieting. After all, maximizing profit for a business may be at the expense of consumers, and may not benefit society at large. There are several responses to this feeling. First, maximizing profit in general is maximizing efficiency. Society can use the tax system to spread the benefit of increased profit. Second, increased efficiency often comes from improved accuracy in targeting, which benefits the people being targeted. Businesses have no motive to send advertising to people who will merely be annoyed and not respond.

On the other hand, there are some applications of data mining where the profit is positive, but small in comparison with the cost of data mining. In these cases, there might be more social benefit in directing the same effort towards a different objective. For example, in a case study covered below, a charity can spend $70,000 sending solicitations to donors who contribute $84,000 in total. The process has a net benefit for the charity, but the total benefit to society may be negative when the cost to recipients is considered.

1.1 Limitations of predictive analytics

It is important to understand the limitations of predictive analytics. First, in general, one cannot make progress without a dataset for training of adequate size and quality. Second, it is crucial to have a clear definition of the concept that is to be predicted, and to have historical examples of the concept. Consider for example this extract from an article in the London *Financial Times* dated May 13, 2009:

> Fico, the company behind the credit score, recently launched a service that pre-qualifies borrowers for modification programmes using their in-house scoring data. Lenders pay a small fee for Fico to refer potential candidates for modifications that have already been vetted for inclusion in the programme. Fico can also help lenders find borrowers that will best respond to modifications and learn how to get in touch with them.

It is hard to see how this could be a successful application of data mining, because it is hard to see how a useful labeled training set could exist. The target concept is "borrowers that will best respond to modifications." From a lender's perspective (and Fico works for lenders not borrowers) such a borrower is one who would not pay

under his current contract, but who would pay if given a modified contract. Especially in 2009, lenders had no long historical experience with offering modifications to borrowers, so FICO did not have relevant data. Moreover, the definition of the target is based on a counterfactual, that is on reading the minds of borrowers. Data mining cannot read minds.

For a successful data mining application, the actions to be taken based on predictions need to be defined clearly and to have reliable profit consequences. The difference between a regular payment and a modified payment is often small, for example $200 in the case described in the newspaper article. It is not clear that giving people modifications will really change their behavior dramatically.

For a successful data mining application also, actions must not have major unintended consequences. Here, modifications may change behavior in undesired ways. A person requesting a modification is already thinking of not paying. Those who get modifications may be motivated to return and request further concessions.

Additionally, for predictive analytics to be successful, the training data must be representative of the test data. Typically, the training data come from the past, while the test data arise in the future. If the phenomenon to be predicted is not stable over time, then predictions are likely not to be useful. Here, changes in the general economy, in the price level of houses, and in social attitudes towards foreclosures, are all likely to change the behavior of borrowers in the future, in systematic but unknown ways.

Last but not least, for a successful application it helps if the consequences of actions are essentially independent for different examples. This may be not the case here. Rational borrowers who hear about others getting modifications will try to make themselves appear to be candidates for a modification. So each modification generates a cost that is not restricted to the loss incurred with respect to the individual getting the modification.

An even more clear example of an application of predictive analytics that is unlikely to succeed is learning a model to predict which persons will commit a major terrorist act. There are so few positive training examples that statistically reliable patterns cannot be learned. Moreover, intelligent terrorists will take steps not to fit in with patterns exhibited by previous terrorists [Jonas and Harper, 2006].

A different reason why data mining can be dangerous is that it can lead to missing qualitative issues. An article in the *Wall Street Journal* on March 22, 2012 says

> ...Lulu has gone back to basics. It doesn't use focus groups, website visits or the industry staple–customer-relationship management software, which tracks purchases. Instead, Ms. Day [the CEO] spends hours each week in Lulu stores observing how customers shop, listening to their complaints, and then using the feedback to tweak product and

Figure 1.1: Top stories as selected by data mining for the Yahoo front page.

> stores. "Big data gives you a false sense of security," says Ms. Day, who spent 20 years at Starbucks Corp., overseeing retail operations in North America and around the world."

A related issue is that data mining can lead to an ever-increased focus on optimizing existing processes, at the expense of understanding the broader situation. For example, for many years Yahoo has used data mining to maximize clicks on the news stories on its front page. As illustrated by Figure 1.1, this has led to the stories becoming more trashy, optimized for a demographic that seeks celebrity news rather than more demanding content.

1.2 Opportunities for predictive analytics

This section discusses criteria for judging the potential success of a proposed application of data mining. The questions here are listed in a reasonable order in which they should be asked and answered. In tandem with an explanation of each question, there is a discussion of the answers to the questions for a sample application. The application is to predict the success of calls made using a mobile telephone. For each

call, the alternative label values are normal termination, call dropped by the network, call dropped by the calling phone, call dropped by the receiving phone, and possibly more.

Does the domain involve numerous individual cases? Data mining and predictive analytics are not directly relevant to making one-off decisions such as selecting the overall strategy of a company or organization. In the example domain, a case is one telephone call, and obviously there are many of these.

Is there a clear objective to be optimized? If it is unclear what the goal is for each case, then there is no definite problem to be solved. Note that the objective is usually from the point of view of a specific agent. For example, if each case is a commercial transaction, then the buyer and seller may have partially conflicting objectives. Data mining is applied to achieve the goals of the agent doing the data mining. In the example domain, the agent doing the data mining is the telephone company, and the goal is for the termination of each call to be successful. While customers have the same general goal, objectives are not perfectly aligned. In particular, each customer is most interested in the success of his or her own calls, but the company may be motivated to prioritize the calls made by its most profitable customers.

Are there actions that influence the objective that can be taken with respect to each case? This is a crucial question. If the agent cannot change the outcomes of cases, then there is no problem to be solved. In the example domain, some available actions are to change the transmission power level of the telephone and/or the base station. In general, a higher power level increases the chance that the call is successful.

Is there an unknown target value that is relevant to the objective that can be predicted for each case? Predictive analytics involves analyzing individual cases in order to predict some relevant characteristic of them that cannot be observed directly at a time when it would be useful to know the characteristic. In the example domain, the target value is whether or not the call will fail.

Is the target value known for numerous historical cases? Yes, after each call its target value is known and can be stored.

Can features of each case be observed that are correlated with the target value? Yes. These features include the phone model, the weather, the location of the phone and of the base station, the relevant power levels, and derived features such as the distance between the phone and the base station.

Are the cases reasonably independent? That is, does the label of one case not influence strongly the labels of other cases? Yes. The failure of one call does not cause the success or failure of another call.

1.3 Overview

In this course we shall only look at methods that have state-of-the-art accuracy, that are sufficiently simple and fast to be easy to use, and that have well-documented successful applications. We will tread a middle ground between focusing on theory at the expense of applications, and understanding methods only at a cookbook level. Often, we shall not look at multiple methods for the same task, when there is one method that is at least as good as all the others from most points of view. In particular, for classifier learning, we will look at support vector machines (SVMs) in detail. We will not examine alternative classifier learning methods such as decision trees, neural networks, boosting, and bagging. All these methods are excellent, but it is hard to identify clearly important scenarios in which they are definitely superior to SVMs. We may also look at random forests, a nonlinear method that is often superior to linear SVMs, and which is widely used in commercial applications nowadays.

Sample CSE 255 Quiz

Instructions. Do this quiz in partnership with exactly one other student. Write both your names at the top of this page. Discuss the answer to the question with each other, and then write your joint answer below the question. Use the back of the page if necessary. It is fine if you overhear what other students say, because you still need to decide if they are right or wrong. You have seven minutes.

Question. Suppose that you work for Apple. The company has developed an unprecedented new product called the iKid, which is aimed at children. Your colleague says "Let's use data mining on our database of existing customers to learn a model that predicts who is likely to buy an iKid."

Is your colleague's idea good or bad? Explain at least one significant specific reason that supports your answer.

Chapter 2

Predictive analytics in general

This chapter explains supervised learning, linear regression, and data cleaning and recoding.

2.1 Supervised learning

The goal of a supervised learning algorithm is to obtain a classifier by learning from training examples. A classifier is something that can be used to make predictions on test examples. This type of learning is called "supervised" because of the metaphor that a teacher (i.e. a supervisor) has provided the true label of each training example.

Each training and test example is represented in the same way, as a row vector of fixed length p. Each element in the vector representing an example is called a feature value. It may be real number or a value of any other type. A training set is a set of vectors with known label values. It is essentially the same thing as a table in a relational database, and an example is one row in such a table. Row, tuple, and vector are essentially synonyms. A column in such a table is often called a feature, or an attribute, in data mining. Sometimes it is important to distinguish between a feature, which is an entire column, and a feature value.

The label y for a test example is unknown. The output of the classifier is a conjecture about y, i.e. a predicted y value. Often each label value y is a real number. In this case, supervised learning is called "regression" and the classifier is called a "regression model." The word "classifier" is usually reserved for the case where label values are discrete. In the simplest but most common case, there are just two label values. These may be called -1 and +1, or 0 and 1, or no and yes, or negative and positive.

With n training examples, and with each example consisting of values for p different features, the training data are a matrix with n rows and p columns, along with

a column vector of y values. The cardinality of the training set is n, while its dimensionality is p. We use the notation x_{ij} for the value of feature number j of example number i. The label of example i is y_i. True labels are known for training examples, but not for test examples.

2.2 Data validation and cleaning

At the start of a data mining project, typically the analyst does not understand the data fully. Important documentation may be missing, and the data may come from multiple sources that each produced data in a slightly different way. Therefore, the first stage in a data mining project is typically to attempt to detect errors, and to reduce their prevalence. This is often one of the most time-consuming stages of a project, and one of the hardest stages to make routine, since it requires experience, intuition, judgment, and interaction with numerous other people.

Validating data means confirming that it is reliable, while cleaning data means fixing errors in it. Often, it is impossible to be sure whether or not a given data value is correct or not, and if it is incorrect, it is impossible to find the correct value. Moreover, there may be so many errors that the cost of fixing all of them would be prohibitive. However, it is often possible to gain reasonable confidence that certain data is likely to be correct or incorrect, because it does or does not pass a series of checks. When data is likely to be incorrect, the simplest approach to cleaning it is simply to discard it. While this may introduce biases, more sophisticated cleaning methods are often not beneficial. Generally, more effort should be devoted to data validation than to data repair.

Something that is not exactly data repair is making a feature more meaningful. For example, suppose that the longevity of a customer is reset to zero when the person changes address. In this case, it may be useful to compute a new feature that measures true longevity regardless of address changes.

Another important type of data cleaning is merging records that refer to the same entity, and should not be separate. The separate records often exist because of variations in representation, such as different ways of writing the same address. This cleaning process has many names, including record linkage, reduplication, and merge/purge. There are sophisticated methods for it that are outside the scope of this chapter.

A first step in validating data is to examine reports that show basic statistics (minimum, maximum, most frequent values, etc.) for each feature. The reason why each frequent value is frequent should be understood. For example, the social security number 999-99-9999 may be frequent because it means "newly immigrated foreigner." Features that are supposed to have unique values, i.e. database keys such

as id numbers, should be verified to be unique. When the actual value of a feature is supposed to have no intrinsic meaning, this should also be verified. For example, a plot of id numbers may show that certain numerical intervals contain many more numbers than other intervals. If so, the reason for this pattern should be investigated.

A so-called orphan record is one that should have a counterpart elsewhere in the data, but does not. For example, if a transaction record has a certain credit card number, there should be a record elsewhere containing information about the holder of the card, etc. If no such record exists, then the transaction record is an orphan. Obviously, one should search for orphans and deal with them if they exist.

It is often useful to visualize a table of data as a bitmap. In each column, place a 1 for a value of special interest and a 0 for all other values. Then, the analyst can inspect a large table visually. For example, a 1 can be displayed when the actual value is zero, or when the actual value is missing. It is common that a bitmap display shows that there are obvious patterns in data that should not be present.

It can happen that a record is anomalous even though none of its individual feature values is unusual. One way to find anomalous records is to apply a clustering algorithm. The anomalous records may show up as clusters that are small, or even singletons.

Given data containing m features, there are $m(m-1)/2$ possible pairs of features. Therefore, it can be infeasible to look for anomalies in the correlation between every pair of features. However, it can be worthwhile to look for correlations with specific features where correlations should not exist, or should be small. Id number is an example of a feature that should be uncorrelated with others. Time is a feature where it is important to look for nonlinear patterns. For example, suppose that orders for a product peak at the beginning of each month. If so, this may be a genuine pattern, or it may be an artifact due to orders simply being processed in batches.

Another important data validation step is to check that distributions that should be similar really are. For example, a medical dataset may consist of a control group and a treatment group, where each patient was supposedly assigned randomly to one of the two groups. In this case, it is important to check that the distributions of age, previous diagnoses, etc., are all similar for both groups. Often, groups that should be statistically identical turn out to have systematic differences.

2.3 Data recoding

In real-world data, there is a lot of variability and complexity in features. Some features are real-valued. Other features are numerical but not real-valued, for example integers or monetary amounts. Many features are categorical, e.g. for a student the feature "year" may have values freshman, sophomore, junior, and senior. Usually the

names used for the different values of a categorical feature make no difference to data mining algorithms, but are critical for human understanding. Sometimes categorical features have names that look numerical, e.g. zip codes, and/or have an unwieldy number of different values. Dealing with these is difficult.

It is also difficult to deal with features that are really only meaningful in conjunction with other features, such as "day" in the combination "day month year." Moreover, important features (meaning features with predictive power) may be implicit in other features. For example, the day of the week may be predictive, but only the day/month/year date is given in the original data. An important job for a human is to think which features may be predictive, based on understanding the application domain, and then to write software that makes these features explicit. No learning algorithm can be expected to discover day-of-week automatically as a function of day/month/year.

Even with all the complexity of features, many aspects are typically ignored in data mining. Usually, units such as dollars and dimensions such as kilograms are omitted. Difficulty in determining feature values is also ignored: for example, how does one define the year of a student who has transferred from a different college, or who is part-time?

Some training algorithms can only handle categorical features. For these, features that are numerical can be discretized. The range of the numerical values is partitioned into a fixed number of intervals that are called bins. The word "partitioned" means that the bins are exhaustive and mutually exclusive, i.e. non-overlapping. One can set boundaries for the bins so that each bin has equal width, i.e. the boundaries are regularly spaced, or one can set boundaries so that each bin contains approximately the same number of training examples, i.e. the bins are "equal count." Each bin is given an arbitrary name. Each numerical value is then replaced by the name of the bin in which the value lies. It often works well to use ten bins.

Other training algorithms can only handle real-valued features. For these, categorical features must be made numerical. The values of a binary feature can be recoded as 0.0 or 1.0. It is conventional to code "false" or "no" as 0.0, and "true" or "yes" as 1.0. Usually, the best way to recode a feature that has k different categorical values is to use k real-valued features. For the jth categorical value, set the jth of these features equal to 1.0 and set all $k-1$ others equal to 0.0.[1]

Categorical features with many values (say, over 20) are often difficult to deal with. Typically, human intervention is needed to recode them intelligently. For example, zipcodes may be recoded as just their first one or two letters, since these

[1] The ordering of the values, i.e. which value is associated with $j = 1$, etc., is arbitrary. Mathematically, it is sometimes preferable to use only $k-1$ real-valued features. For the last categorical value, set all $k-1$ features equal to 0.0. For the jth categorical value where $j < k$, set the jth feature value to 1.0 and set all $k-1$ others equal to 0.0.

indicate meaningful regions of the United States. If one has a large dataset with many examples from each of the 50 states, then it may be reasonable to leave this as a categorical feature with 50 values. Otherwise, it may be useful to group small states together in sensible ways, for example to create a New England group for MA, CT, VT, ME, NH.

An intelligent way to recode discrete predictors is to replace each discrete value by the mean of the target conditioned on that discrete value. For example, if the average y value is 20 for men versus 16 for women, these values could replace the male and female values of a variable for gender. This idea is especially useful as a way to convert a discrete feature with many values, for example the 50 U.S. states, into a useful single numerical feature.

However, as just explained, the standard way to recode a discrete feature with m values is to introduce $m - 1$ binary features. With this standard approach, the training algorithm can learn a coefficient for each new feature that corresponds to an optimal numerical value for the corresponding discrete value. Conditional means are likely to be meaningful and useful, but they will not yield better predictions than the coefficients learned in the standard approach. A major advantage of the conditional-means approach is that it avoids an explosion in the dimensionality of training and test examples.

Topics to be explained also: Mixed types, sparse data.

Normalization. After conditional-mean new values have been created, they can be scaled to have zero mean and unit variance in the same way as other features.

When preprocessing and recoding data, it is vital not to peek at test data. If preprocessing is based on test data in any way, then the test data is available indirectly for training, which can lead to overfitting. (See Section 2.5 for an explanation of overfitting.) If a feature is normalized by z-scoring, its mean and standard deviation must be computed using only the training data. Then, later, the same mean and standard deviation values must be used for normalizing this feature in test data.

Even more important, target values from test data must not be used in any way before or during training. When a discrete value is replaced by the mean of the target conditioned on it, the mean must be only of *training* target values. Later, in test data, the same discrete value must be replaced by the same mean of training target values.

2.4 Missing data

A very common difficulty is that the value of a given feature is missing for some training and/or test examples. Often, missing values are indicated by question marks. However, often missing values are indicated by strings that look like valid known

values, such as 0 (zero). It is important not to treat a value that means missing inadvertently as a valid regular value.

Some training algorithms can handle missingness internally. If not, the simplest approach is just to discard all examples (rows) with missing values. Keeping only examples without any missing values is called "complete case analysis." An equally simple, but different, approach is to discard all features (columns) with missing values. However, in many applications, both these approaches eliminate too much useful training data.

Also, the fact that a particular feature is missing may itself be a useful predictor. Therefore, it is often beneficial to create an additional binary feature that is 0 for missing and 1 for present. If a feature with missing values is retained, then it is reasonable to replace each missing value by the mean or mode of the non-missing values. This process is called imputation. More sophisticated imputation procedures exist, but they are not always better.

There can be multiple types of missingness, and multiple codes indicating a missing value. The code NA often means "not applicable" whereas NK means "not known." Whether or not the value of one feature is missing may depend on the value of another feature. For example, if wind speed is zero, then wind direction is undefined.

2.5 The issue of overfitting

In a real-world application of supervised learning, we have a training set of examples with labels, and a test set of examples with unknown labels. The whole point is to make predictions for the test examples.

However, in research or experimentation we want to measure the performance achieved by a learning algorithm. To do this we use a test set consisting of examples with known labels. We train the classifier on the training set, apply it to the test set, and then measure performance by comparing the predicted labels with the true labels (which were not available to the training algorithm).

It is absolutely vital to measure the performance of a classifier on an independent test set. Every training algorithm looks for patterns in the training data, i.e. correlations between the features and the class. Some of the patterns discovered may be spurious, i.e. they are valid in the training data due to randomness in how the training data was selected from the population, but they are not valid, or not as strong, in the whole population. A classifier that relies on these spurious patterns will have higher accuracy on the training examples than it will on the whole population. Only accuracy measured on an independent test set is a fair estimate of accuracy on the

whole population. The phenomenon of relying on patterns that are strong only in the training data is called overfitting. In practice it is an omnipresent danger.

Most training algorithms have some settings that the user can choose between. For ridge regression the main algorithmic parameter is the degree of regularization λ. Other algorithmic choices are which sets of features to use. It is natural to run a supervised learning algorithm many times, and to measure the accuracy of the function (classifier or regression function) learned with different settings. A set of labeled examples used to measure accuracy with different algorithmic settings, in order to pick the best settings, is called a validation set. If you use a validation set, it is important to have a final test set that is independent of both the training set and the validation set. For fairness, the final test set must be used only once. The only purpose of the final test set is to estimate the true accuracy achievable with the settings previously chosen using the validation set.

Dividing the available data into training, validation, and test sets should be done randomly, in order to guarantee that each set is a random sample from the same distribution. However, a very important real-world issue is that future real test examples, for which the true label is genuinely unknown, may be *not* a sample from the same distribution.

CSE 255 assignment due on April 9, 2013

Please do this assignment in a team of exactly two people. Choose a partner who has a background different from yours.

Install R and Rattle on a computer of your choice, and obtain access to the book *Data Mining with Rattle and R*. (Online access is free from on campus or using the campus VPN.) Use the example weather dataset included with Rattle.

Examine the *Data* tab of Rattle. Are the choices made by Rattle about *Ignore*, *Ident* etc. sensible? If not, make alternative choices. Examine the results of the *Explore* tab. Are the values of every feature reasonable? If not, what could you do to make the data cleaner?

Using the *Test* tab, look at the correlations between the features *Pressure9am*, *Cloud9am*, and *Humidity9am*. Are the three results reasonable? Suggest a real-world explanation for them; look up some basic meteorology if necessary.

Using the *Model* tab, use three different methods to train a classifier that predicts *RainTomorrow*: a decision tree, a linear SVM, and logistic regression. Use the default options of Rattle for each method. Using the *Evaluate* tab, look at six confusion matrices: for each method, one for the training set and one for the test set. Which methods, if any, are underfitting or overfitting the training set?

Next, go back to the *Model* tab and look at the actual model learned by each method (either a tree or a linear function). For each model, do you see any characteristics of it that suggest that the model may be underfitting or overfitting?

Last but not least, go back to the *Transform* tab. Try various transformation of the features. Can you find any transformations that improve accuracy of the best model on test data? Is it believable that any improvement would carry over to future test data?

The deliverable for this assignment is a brief report, like a memo. Describe your findings for each step above. Try hard not to fool yourselves, and explain any limitations on the reliability of your conclusions. Do not speculate about future work, and do not explain at length anything that was not successful. Write in the present tense. Organize your report logically, not chronologically. Make the report typeset and format it similarly to this assignment description.

Chapter 3

Linear regression

This chapter explains linear regression, both because it is intrinsically important, and because the issues that arise with linear regression also arise with many other data mining methods.

Let x be an instance and let y be its real-valued label. For linear regression, x must be a vector of real numbers of fixed length. Remember that this length p is often called the dimension, or dimensionality, of x. Write $x = \langle x_1, x_2, \ldots, x_p \rangle$. The linear regression model is

$$y = b_0 + b_1 x_1 + b_2 x_2 + \ldots + b_p x_p.$$

The righthand side above is called a linear function of x. The linear function is defined by its coefficients b_0 to b_p. These coefficients are the *output* of the data mining algorithm.

The coefficient b_0 is called the intercept. It is the value of y predicted by the model if $x_i = 0$ for all i. Of course, it may be completely unrealistic that all features x_i have value zero. The coefficient b_i is the amount by which the predicted y value increases if x_i increases by 1, if the value of all other features is unchanged. For example, suppose x_i is a binary feature where $x_i = 0$ means female and $x_i = 1$ means male, and suppose $b_i = -2.5$. Then the predicted y value for males is lower by 2.5, everything else being held constant.

Suppose that the training set has cardinality n, i.e. it consists of n examples of the form $\langle x_i, y_i \rangle$, where $x_i = \langle x_{i1}, \ldots, x_{ip} \rangle$. Let b be any set of coefficients. The predicted value for x_i is

$$\hat{y}_i = f(x_i; b) = b_0 + \sum_{j=1}^{p} b_j x_{ij}.$$

The semicolon in the expression $f(x_i; b)$ emphasizes that the vector x_i is a variable input, while b is a fixed set of parameter values. If we define $x_{i0} = 1$ for every i, then we can write

$$\hat{y}_i = \sum_{j=0}^{p} b_j x_{ij}.$$

The constant $x_{i0} = 1$ can be called a pseudo-feature.

Finding the optimal values of the coefficients b_0 to b_p is the job of the training algorithm. To make this task well-defined, we need a definition of what "optimal" means. The standard approach is to say that optimal means minimizing the sum of squared errors on the training set, where the squared error on training example i is $(y_i - \hat{y}_i)^2$. The training algorithm then finds

$$\hat{b} = \text{argmin}_b \sum_{i=1}^{n} (f(x_i; b) - y_i)^2.$$

The objective function $\sum_i (y_i - \sum_j b_j x_{ij})^2$ is called the sum of squared errors, or SSE for short. Note that during training the n different x_i and y_i values are fixed, while the parameters b are variable.

The optimal coefficient values $\hat{b}$ are not defined uniquely if the number n of training examples is less than the number p of features. Even if $n > p$ is true, the optimal coefficients have multiple equivalent values if some features are themselves related linearly. Here, "equivalent" means that the different sets of coefficients achieve the same minimum SSE. For an intuitive example, suppose features 1 and 2 are temperature measured in degrees Celsius and degrees Fahrenheit respectively.[1] Then $x_2 = 32 + 9(x_1/5) = 32 + 1.8x_1$, and the same model can be written in many different ways:

- $y = b_0 + b_1 x_1 + b_2 x_2$
- $y = b_0 + b_1 x_1 + b_2(32 + 1.8x_1) = [b_0 + 32b_2] + [b_1(1 + 1.8b_2)]x_1 + 0x_2$

and an infinite number of other ways. In the extreme, suppose $x_1 = x_2$. Then all models $y = b_0 + b_1 x_1 + b_2 x_2$ are equivalent for which $b_1 + b_2$ equals a constant.

When two or more features are approximately related linearly, then the true values of the coefficients of those features are not well determined. The coefficients obtained by training will be strongly influenced by randomness in the training data. Regularization is a way to reduce the influence of this type of randomness. Consider

[1] Having two features which are in fact duplicates, but measured in different ways, is a common data quality problem. The problem can be hard to detect, because there maybe differing noise in the measurements, and/or some of the duplicate measurements may be missing.

all models $y = b_0 + b_1 x_1 + b_2 x_2$ for which $b_1 + b_2 = c$. Among these models, there is a unique one that minimizes the function $b_1^2 + b_2^2$. This model has $b_1 = b_2 = c/2$. We can obtain it by setting the objective function for training to be the sum of squared errors (SSE) plus a function that penalizes large values of the coefficients. A simple penalty function of this type is $\sum_{j=1}^{p} b_j^2$. A parameter λ can control the relative importance of the two objectives, namely SSE and penalty:

$$\hat{b} = \text{argmin}_b \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \; + \; \lambda \frac{1}{p} \sum_{j=1}^{p} b_j^2.$$

If $\lambda = 0$ then one gets the standard least-squares linear regression solution. As λ increases, the penalty on large coefficients gets stronger, and the typical values of coefficients get smaller. The parameter λ is often called the strength of regularization. The fractions $1/n$ and $1/p$ do not make an essential difference. They can be used to make the numerical value of λ easier to interpret.

The penalty function $\sum_{j=1}^{p} b_j^2$ is the square of the L_2 norm of the vector b. Using it for linear regression is called ridge regression. Any penalty function that treats all coefficients b_j equally, like the L_2 norm, is sensible only if the typical magnitudes of the values of each feature are similar; this is an important motivation for data normalization. Note that in the formula $\sum_{j=1}^{p} b_j^2$ the sum excludes the intercept coefficient b_0. One reason for doing this is that the target y values are typically not normalized.

3.1 Interpreting the coefficients of a linear model

It is common to desire a data mining model that is interpretable, that is one that can be used not only to make predictions, but also to understand patterns, and even possible mechanisms, in the phenomenon that is being studied. Linear models, whether for regression or for classification as described in later chapters, do appear interpretable at first sight. However, much caution is needed when attempting to derive conclusions from numerical coefficients.

Consider the linear regression model in Table 3.1 for predicting high-density cholesterol (HDL) levels.[2] The example illustrates at least two crucial issues. First, if predictors are collinear, then one may appear significant and the other not, when in fact both are significant or both are not. Above, diastolic blood pressure is statistically significant, but systolic is not. This may possibly be true for some physiological reason. But it may also be an artifact of collinearity.

[2] HDL cholesterol is considered beneficial and is sometimes called "good" cholesterol. Source: `http://www.jerrydallal.com/LHSP/importnt.htm`. Predictors have been reordered from most to least statistically significant, as measured by p-value.

predictor	coefficient	std.error	Tstat	p-value
intercept	1.16448	0.28804	4.04	<.0001
BMI	−0.01205	0.00295	−4.08	<.0001
LCHOL	0.31109	0.10936	2.84	0.0051
GLUM	−0.00046	0.00018	−2.50	0.0135
DIAST	0.00255	0.00103	2.47	0.0147
BLC	0.05055	0.02215	2.28	0.0239
PRSSY	−0.00041	0.00044	−0.95	0.3436
SKINF	0.00147	0.00183	0.81	0.4221
AGE	−0.00092	0.00125	−0.74	0.4602

Table 3.1: Linear regression model to predict HDL cholesterol levels. From most to least statistically significant, the predictors are body mass index, the log of total cholesterol, glucose metabolism, diastolic blood pressures, and vitamin C level in blood. Three predictors are not statistically significant: systolic blood pressure, skinfold thickness, and age in years.

Second, a predictor may be practically important, and statistically significant, but still useless for interventions. This happens if the predictor and the outcome have a common cause, or if the outcome causes the predictor. Above, vitamin C is statistically significant. But it may be that vitamin C is simply an indicator of a generally healthy diet high in fruits and vegetables. If this is true, then merely taking a vitamin C supplement will not cause an increase in HDL level.

A third crucial issue is that a correlation may disagree with other knowledge and assumptions. For example, vitamin C is generally considered beneficial or neutral. If lower vitamin C was associated with higher HDL, one would be cautious about believing this relationship, even if the association was statistically significant.

Exercises

1.

Suppose you are building a model to predict how many dollars someone will spend at Sears. You know the gender of each customer, male or female. Since you are using linear regression, you must recode this discrete feature as continuous. You decide to use two real-valued features, x_{11} and x_{12}. The coding is a standard "one of n" scheme, as follows:

gender	x_{11}	x_{12}
male	1	0
female	0	1

Learning from a large training set yields the model

$$y = \ldots + 15x_{11} + 75x_{12} + \ldots$$

Dr. Roebuck says "Aha! The average woman spends $75, but the average man spends only $15."

(a) Explain why Dr. Roebuck's conclusion is not valid. *The model only predicts spending of $75 for a woman if all other features have value zero. This may not be true for the average woman. Indeed it will not be true for any woman, if features such as "age" are not normalized.*

(b) Explain what conclusion can actually be drawn from the numbers 15 and 75. *The conclusion is that if everything else is held constant, then on average a woman will spend $60 more than a man. Note that if some feature values are systematically different for men and women, then even this conclusion is not useful, because then it is not reasonable to hold all other feature values constant.*

(c) Explain a desirable way to simplify the model. *The two features x_{11} and x_{12} are linearly related. Hence, they make the optimal model be undefined, in the absence of regularization. It would be good to eliminate one of these two features. The expressiveness of the linear model would be unchanged.*

2.

Suppose that you are training a model to predict how many transactions a credit card customer will make. You know the education level of each customer. Since you are using linear regression, you recode this discrete feature as continuous. You decide to use two real-valued features, x_{37} and x_{38}. The coding is a "one of two" scheme, as follows:

	x_{37}	x_{38}
college grad	1	0
not college grad	0	1

Learning from a large training set yields the model

$$y = \ldots + 5.5x_{37} + 3.2x_{38} + \ldots$$

(a) Dr. Goldman concludes that the average college graduate makes 5.5 transactions. Explain why Dr. Goldman's conclusion is likely to be false. *The model only predicts 5.5 transactions if all other features, including the intercept, have value zero. This may not be true for the average college grad. It will certainly be false if features such as "age" are not normalized.*

(b) Dr. Sachs concludes that being a college graduate causes a person to make 2.3 more transactions, on average. Explain why Dr. Sachs' conclusion is likely to be false also. *First, if any other feature have different values on average for men and women, for example income, then* $5.5 - 3.2 = 2.3$ *is not the average difference in predicted* y *value between groups. Said another way, it is unlikely to be reasonable to hold all other feature values constant when comparing groups.*

Second, even if 2.3 is the average difference, one cannot say that this difference is caused by being a college graduate. There may be some other unknown common cause, for example.

3.

Suppose that you are training a model (a classifier) to predict whether or not a graduate student drops out. You have enough positive and negative training examples, from a selective graduate program. You are training the model correctly, without overfitting or any other mistake. Assume further that verbal ability is a strong causal influence on the target, and that the verbal GRE score is a good measure of verbal ability.

Explain two separate reasonable scenarios where the verbal GRE score, despite all the assumptions above being true, does *not* have strong predictive power in a trained model.

CSE 255 assignment due on April 16, 2013

As before, do this assignment in a team of exactly two people, and choose a partner with a background different from yours. You may keep the same partner, or change.

Download the file `cup98lrn.zip` from from `http://archive.ics.uci.edu/ml/databases/kddcup98/kddcup98.html`. Read the associated documentation. Load the data into R, or other software for data mining. Select the 4843 records that have feature `TARGET_B=1`. Now, build a linear regression model to predict the field `TARGET_D` as accurately as possible. Use root mean squared error (RMSE) as the definition of error. Do a combination of the following:

- Discard useless features.
- Process missing feature values in a sensible way.
- Recode non-numerical features as numerical.
- Add transformations of existing features.
- Compare different strengths of ridge (L_2) regularization.

Do the steps above repeatedly in order to explore alternative ways of using the data. The outcome should be the best model that you can find that uses 10 or fewer of the original features. After recoding features, you will likely have more than 10 derived features, but these should all be based on at most 10 of the original features. Use cross-validation to evaluate each model that you consider. The RMSE of your final model (on validation folds, not on training folds) should be less than $9.00.

You should discard useless features quickly; for this assignment, you do not need to use a systematic process to identify these. If you normalize all input features, and you use strong L_2 regularization (ridge parameter 10^7 perhaps), then the regression coefficients will be an (imperfect) indicator of the relative importance of features. L_1 regularization is an alternative way of identifying important features quickly. Construct the best model that you can, including choosing the best strength of L_2 regularization for the features that you retain.

After you have found a final model that you expect to generalize well, use ten-fold cross-validation to estimate its accuracy. Write down what RMSE you expect to achieve on future data. Then, run this model on the independent test set `cup98val.zip`. For the approximately 5% of records for which `TARGET_B` = 1, compute the RMSE of your predictions with respect to the `TARGET_D` values in the file `valtargt.txt`. Be sure to write down your expected RMSE in advance, to help avoid fooling yourselves. Note that you must use the `CONTROLN` feature to join the `cup98val.zip` and `valtargt.txt` files correctly.

The deliverable, like before, is a brief report. Describe what you did that worked, and your results. Explain any assumptions that you made, and any limitations on the validity or reliability of your results. Explain the RMSE that you expected on the test data, and the RMSE that you observed. Include your final regression model.

Comments

To make your work more efficient, be sure to save the 4843 records in a format that your software can reload quickly. You can use three-fold cross-validation during development to save time, but do use ten-fold cross-validation to evaluate the RMSE of your final chosen model.

It is typically useful to rescale predictors to have mean zero and variance one. However, it loses interpretability to rescale the target variable. Note that if all predictors have mean zero, then the intercept of a linear regression model is the mean of the target, $15.6243 here. The intercept should usually not be regularized.

The assignment specifically asks you to report root mean squared error, RMSE. Mean squared error, MSE, is an alternative, but whichever is chosen should be used consistently. In general, do not confuse readers by switching between multiple measures without a good reason, and do follow precise instructions precisely.

As is often the case, good performance can be achieved with a very simple model. The most informative single feature is `LASTGIFT`, the dollar amount of the person's most recent gift. A model based on just this single feature achieves RMSE of $9.98. The assignment asks you to produce a final model based on at most 10 of the original features. It is not a good idea to begin by choosing a subset of the 480 original features based on human intuition, because intuition is unreliable. It is also not a good idea to eliminate automatically features with missing values.

Many tools for data mining have operators that search for highly predictive subsets of variables. These operators tend to have two major problems. First, they are too slow to be used on a large initial set of variables, so it is easy for human intuition to pick an initial set that is not good. Second, they try a very large number of alternative subsets of variables, and pick the one that performs best on some training or tuning set. Because of the high number of alternatives considered, the chosen subset is likely to overfit substantially the training or tuning set.

It is important that you apply *only one* model to the final test data. After you do this, you may be tempted to go back and train a different model, and to apply that to the `cup98val.zip` test data. However, doing so would be cheating, because it would allow the test data to influence the training process, which would lead to overfitting the test data, and hence to an overoptimistic estimate of accuracy. More generally, even for a task as simple and well-defined as the one in this assignment,

it is easy to fool yourselves, and your readers, about the correctness and success of your work. Do apply critical thinking, and aim for simplicity and clarity.

Chapter 4

Introduction to Rapidminer

By default, many Java implementations allocate so little memory that Rapidminer will quickly terminate with an "out of memory" message. This is not a problem with Windows 7, but otherwise, launch Rapidminer with a command like

```
java -Xmx1g -jar rapidminer.jar
```

where `1g` means one gigabyte.

4.1 Standardization of features

The recommended procedure is as follows, in order.

- Normalize all numerical features to have mean zero and variance 1.
- Convert each nominal feature with k alternative values into k different binary features.
- Optionally, drop all binary features with fewer than 100 examples for either binary value.
- Convert each binary feature into a numerical feature with values 0.0 and 1.0.

It is not recommended to normalize numerical features obtained from binary features. The reason is that this normalization would destroy the sparsity of the dataset, and hence make some efficient training algorithms much slower. Note that destroying sparsity is not an issue when the dataset is stored as a dense matrix, which it is by default in Rapidminer.

Normalization is also questionable for variables whose distribution is highly uneven. For example, almost all donation amounts are under $50, but a few are up

to $500. No linear normalization, whether by z-scoring or by transformation to the range 0 to 1 or otherwise, will allow a linear classifier to discriminate well between different common values of the variable. For variables with uneven distributions, it is often useful to apply a nonlinear transformation that makes their distribution have more of a uniform or Gaussian shape. A common transformation that often achieves this is to take logarithms. A useful trick when zero is a frequent value of a variable x is to use the mapping $x \mapsto \log(x + 1)$. This leaves zeros unchanged and hence preserves sparsity.

Eliminating binary features for which either value has fewer than 100 training examples is a heuristic way to prevent overfitting and at the same time make training faster. It may not be necessary, or beneficial, if regularization is used during training. If regularization is not used, then at least one binary feature must be dropped from each set created by converting a multivalued feature, in order to prevent the existence of multiple equivalent models. For efficiency and interpretability, it is best to drop the binary feature corresponding to the most common value of the original multivalued feature.

If you have separate training and test sets, it is important to do all preprocessing in a perfectly consistent way on both datasets. The simplest is to concatenate both sets before preprocessing, and then split them after. However, concatenation allows information from the test set to influence the details of preprocessing, which is a form of information leakage from the test set, that is of using the test set during training, which is not legitimate.

4.2 Example of a Rapidminer process

Figure 4.2 shows a tree structure of Rapidminer operators that together perform a standard data mining task. The tree illustrates how to perform some common subtasks.

The first operator is ExampleSource. “Read training example” is a name for this particular instance of this operator. If you have two instances of the same operator, you can identify them with different names. You select an operator by clicking on it in the left pane. Then, in the right pane, the Parameters tab shows the arguments of the operator. At the top there is an icon that is a picture of either a person with a red sweater, or a person with an academic cap. Click on the icon to toggle between these. When the person in red is visible, you are in expert mode, where you can see all the parameters of each operator.

The example source operator has a parameter named attributes. Normally this is file name with extension “aml.” The corresponding file should contain a schema definition for a dataset. The actual data are in a file with the same name with exten-

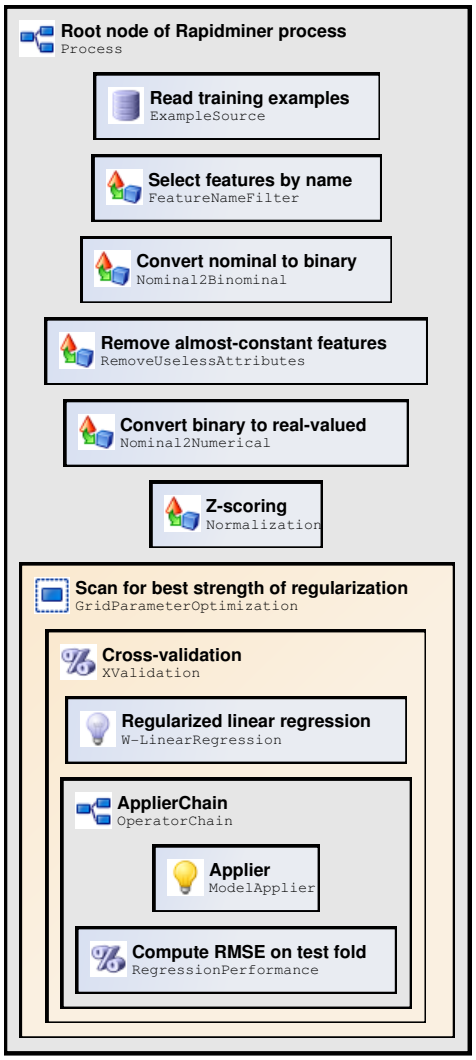

Figure 4.1: Rapidminer process for regularized linear regression.

sion "dat." The easiest way to create these files is by clicking on "Start Data Loading Wizard." The first step with this wizard is to specify the file to read data from, the character that begins comment lines, and the decimal point character. Ticking the box for "use double quotes" can avoid some error messages.

In the next panel, you specify the delimiter that divides fields within each row of data. If you choose the wrong delimiter, the data preview at the bottom will look wrong. In the next panel, tick the box to make the first row be interpreted as field names. If this is not true for your data, the easiest is to make it true outside Rapidminer, with a text editor or otherwise. When you click Next on this panel, all rows of data are loaded. Error messages may appear in the bottom pane. If there are no errors and the data file is large, then Rapidminer hangs with no visible progress. The same thing happens if you click Previous from the following panel. You can use a CPU monitor to see what Rapidminer is doing.

The next panel asks you to specify the type of each attribute. The wizard guesses this based only on the first row of data, so it often makes mistakes, which you have to fix by trial and error. The following panel asks you to say which features are special. The most common special choice is "label" which means that an attribute is a target to be predicted.

Finally, you specify a file name that is used with "aml" and "dat" extensions to save the data in Rapidminer format.

To keep just features with certain names, use the operator FeatureNameFilter. Let the argument `skip_features_with_name` be `.*` and let the argument `except_features_with_name` identify the features to keep. In our sample process, it is `(.*AMNT.*)|(.*GIFT.*)(YRS.*)|(.*MALE)|(STATE)|(PEPSTRFL)|(.*GIFT)|(MDM.*)|(RFA_2.*)`.

In order to convert a discrete feature with k different values into k real-valued 0/1 features, two operators are needed. The first is `Nominal2Binominal`, while the second is `Nominal2Numerical`. Note that the documentation of the latter operator in Rapidminer is misleading: it cannot convert a discrete feature into multiple numerical features directly. The operator `Nominal2Binominal` is quite slow. Applying it to discrete features with more than 50 alternative values is not recommended.

The simplest way to find a good value for an algorithm setting is to use the `XValidation` operator nested inside the `GridParameterOptimization` operator. The way to indicate nesting of operators is by dragging and dropping. First create the inner operator subtree. Then insert the outer operator. Then drag the root of the inner subtree, and drop it on top of the outer operator.

4.3 Other notes on Rapidminer

Reusing existing processes.

Saving datasets.

Operators from Weka versus from elsewhere.

Comments on specific operators: Nominal2Numerical, Nominal2Binominal, RemoveUselessFeatures.

Eliminating non-alphanumerical characters, using quote marks, trimming lines, trimming commas.

Chapter 5

Support vector machines

This chapter explains support vector machines (SVMs). The SVMs described here are called soft margin SVMs, for reasons that will be explained later. This variant is by far the most commonly used in practice. The chapter discusses linear and nonlinear kernels. It also discusses preventing overfitting via regularization.

We have seen how to use linear regression to predict a real-valued label. Now we will see how to use a similar model to predict a binary label. In later chapters, where we think probabilistically, it will be convenient to assume that a binary label takes on true values 0 or 1. However, in this chapter it will be convenient to assume that the label y has true value either $+1$ or -1.

5.1 Loss functions

Let x be an instance (a numerical vector of dimension d), let y be its true label, and let $f(x)$ be a prediction. Assume that the prediction is real-valued. If we need to convert it into a binary prediction, we will threshold it at zero:

$$\hat{y} = 2 \cdot I(f(x) \geq 0) - 1$$

where $I()$ is an indicator function that is 1 if its argument is true, and 0 if its argument is false.

A so-called loss function $l(\cdot, \cdot)$ measures how good the prediction is. Typically loss functions do not depend directly on x, and a loss of zero corresponds to a perfect prediction, while otherwise loss is positive. The most obvious loss function is

$$l(f(x), y) = I(\hat{y} \neq y)$$

which is called the 0-1 loss function. The usual definition of accuracy uses this loss function. However, it has undesirable properties. First, it loses information: it does

not distinguish between predictions $f(x)$ that are almost right, and those that are very wrong. Second, mathematically, its derivative with respect to the value of $f(x)$ is either undefined or zero. This makes the 0/1 loss function difficult to use in training algorithms that try to minimize loss by adjusting parameter values using derivatives.

A loss function that is preferable in several ways is squared error:

$$l(f(x), y) = (f(x) - y)^2.$$

This function is infinitely differentiable everywhere, and does not lose information when the prediction $f(x)$ is real-valued. However, it says that the prediction $f(x) = 1.5$ is as undesirable as $f(x) = 0.5$ when the true label is $y = 1$. Intuitively, if the true label is $+1$, then a prediction with the correct sign that is greater than 1 should not be considered incorrect.

The following loss function, which is called hinge loss, satisfies the intuition just suggested:

$$l(f(x), -1) = \max\{0, 1 + f(x)\}$$
$$l(f(x), +1) = \max\{0, 1 - f(x)\}$$

To understand this loss function, look at a figure showing the loss $l(f(x), -1)$ for a negative example as a function of the output $f(x)$ of the classifier. The loss is zero, which is the best case, as long as the prediction $f(x) \leq -1$. The further away $f(x)$ is from -1 in the wrong direction, meaning the more $f(x)$ is bigger than -1, the bigger the loss. There is a similar but opposite pattern when the true label is $y = +1$. It is mathematically convenient, but less intuitive, to write the hinge loss function as

$$l(f(x), y) = \max\{0, 1 - yf(x)\}.$$

Using hinge loss is the first major insight behind SVMs. An SVM classifier f is trained to minimize hinge loss. The training process aims to achieve predictions $f(x) \geq 1$ for all training instances x with true label $y = +1$, and to achieve predictions $f(x) \leq -1$ for all training instances x with $y = -1$. Note that we have not yet said anything about what the space of candidate functions f is. Overall, training seeks to classify points correctly, but it does not seek to make predictions be exactly $+1$ or -1. In this sense, the training process intuitively aims to find the best possible classifier, without trying to satisfy any unnecessary additional objectives also. The training process does attempt to distinguish clearly between the two classes, by predictions that are inside the interval $(-1, +1)$.

5.2 Regularization

Given a set of training examples $\langle x_i, y_i \rangle$ for $i = 1$ to $i = n$, where y_i is the true, actual label of the example x_i, the total training loss is the sum

$$\sum_{i=1}^{n} l(f(x_i), y_i).$$

This sum is called the empirical loss because it is computed directly on the specific available data. Suppose that the function f is selected by minimizing empirical loss:

$$f = \text{argmin}_{f \in F} \sum_{i=1}^{n} l(f(x_i), y_i)$$

where F is a space of candidate functions. If F is too flexible, and/or the training set is too small, then we run the risk of overfitting the training data. But if F is too restricted, then we run the risk of underfitting the data. In general, we do not know in advance what the best space F is for a particular training set. A possible solution to this dilemma is to choose a flexible space F, but at the same time to impose a penalty on the complexity of f. Let $c(f)$ be some real-valued measure of complexity. The learning process then becomes to solve

$$f = \text{argmin}_{f \in F} \lambda c(f) + \sum_{i=1}^{n} l(f(x_i), y_i).$$

Here, λ is a parameter that controls the relative strength of the two objectives, namely to minimize the complexity of f and to minimize training error.

Suppose that the space of candidate functions is defined by a vector $w \in \mathbb{R}^d$ of parameters, i.e. we can write $f(x) = g(x; w)$ where g is some fixed function. In this case we can define the complexity of each candidate function to be the norm of the corresponding w. Most commonly we use the square of the L_2 norm, as seen before in the context of linear regression:

$$c(f) = ||w||_2^2 = \sum_{j=1}^{d} w_j^2.$$

However we can also use other norms, including the L_0 norm

$$c(f) = ||w||_0 = \sum_{j=1}^{d} I(w_j \neq 0)$$

or the L_1 norm

$$c(f) = ||w||_1 = \sum_{j=1}^{d} |w_j|.$$

The squared L_2 norm is the most convenient mathematically, and works well in practice. The L_0 norm directly captures a preference for vectors w that are sparse, but is NP-hard to optimize. The L_1 norm can be optimized with gradient methods, and tends to yields sparse w vectors in practice, unlike the L_2 norm.

5.3 Linear soft-margin SVMs

A linear classifier is a function $f(x) = g(x; w) = x \cdot w$ where g is the dot product function. Putting the ideas above together, the objective of learning is to find

$$w = \text{argmin}_{w \in \mathbb{R}^d} \ \lambda ||w||^2 + \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i(w \cdot x_i)\}$$

where the multiplier $1/n$ is included because it makes the meaning of λ the same regardless of the cardinality n of the training set. A linear soft-margin SVM classifier is precisely the solution to this optimization problem. It can be proved that the solution to the minimization problem is always unique. Moreover, the objective function is convex, so there are no local minima. Note that d is the dimensionality of x, and w has the same dimensionality.

An equivalent way of writing the same optimization problem is

$$w = \text{argmin}_{w \in \mathbb{R}^d} \ ||w||^2 + C \sum_{i=1}^{n} \max\{0, 1 - y_i(w \cdot x_i)\}$$

with $C = 1/(n\lambda)$. Many SVM implementations let the user specify C as opposed to λ. A small value for C corresponds to strong regularization, while a large value corresponds to weak regularization. Intuitively, when dimensionality d is fixed, a smaller training set should require a smaller C value. However, useful guidelines are not known for what the best value of C might be for a given dataset. In practice, one has to try multiple values of C, and find the best value experimentally.

Mathematically, the optimization problem above is called an unconstrained primal formulation. It is the easiest description of SVMs to understand, and it is the basis of the fastest known algorithms for training linear SVMs.

5.4 Dual formulation

There is an alternative formulation that is equivalent, and is useful theoretically. This so-called dual formulation is

$$\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$
$$\text{subject to } 0 \leq \alpha_i \leq C.$$

The primal and dual formulations are different optimization problems, but they have the same unique solution. The solution to the dual problem is a coefficient α_i for each training example. Notice that the optimization is over $\mathbb{R}^n$, whereas it is over $\mathbb{R}^d$ in the primal formulation. The trained classifier is $f(x) = w \cdot x$ where the vector

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i.$$

This equation says that w is a weighted linear combination of the training instances x_i, where the weight of each instance is between 0 and C, and the sign of each instance is its label y_i.

Often, solving the dual optimization problem leads to α_i being exactly zero for many training instances. The training instances x_i such that $\alpha_i > 0$ are called support vectors. These instances are the only ones that contribute to the final classifier.

If the number of training examples n is much bigger than the number of features d then most training examples will not be support vectors. This fact is the basis for a picture that is often used to illustrate the intuitive idea that an SVM maximizes the margin between positive and negative training examples. This picture is not incorrect, but it is misleading for several reasons. First, the SVMs that are useful in practice are so-called soft-margin SVMs, where some training points can be misclassified. Second, the regularization that is part of the loss function for SVM training is most useful in cases where n is small relative to d. In those cases, often half or more of all training points become support vectors. Third, the strength of regularization that gives best performance on test data cannot be determined by looking at distances between training points.

The constrained dual formulation is the basis of the training algorithms used by the first generation of SVM implementations, but, as just mentioned, recent research has shown that the unconstrained primal formulation leads to faster training algorithms in the linear case. Moreover, the primal version is easier to understand and easier to use as a foundation for proving bounds on generalization error. However, the dual version is easier to extend to obtain nonlinear SVM classifiers. This extension is based on the idea of a kernel function.

5.5 Nonlinear kernels

Consider two instances x_i and x_j, and consider their dot product $x_i \cdot x_j$. This dot product is a measure of the similarity of the two instances: it is large if they are similar and small if they are not. Dot product similarity is closely related to Euclidean distance through the identity

$$d(x_i, x_j) = ||x_i - x_j|| = (||x_i||^2 - 2x_i \cdot x_j + ||x_j||^2)^{1/2}$$

where by definition $||x||^2 = x \cdot x$. If the instances have unit length, that is $||x_i|| = ||x_j|| = 1$, then Euclidean distance and dot product similarity are perfect opposites.[1] Therefore, the equation

$$f(x) = w \cdot x = \sum_{i=1}^{n} \alpha_i y_i (x_i \cdot x)$$

says that the prediction for a test example x is a weighted average of the training labels y_i, where the weight of y_i is the product of α_i and the degree to which x is similar to x_i.

Consider a re-representation of instances $x \mapsto \Phi(x)$ where the transformation Φ is a function $\mathbb{R}^d \to \mathbb{R}^{d'}$. In principle, we could use dot-product to define similarity in the new space $\mathbb{R}^{d'}$, and train an SVM classifier in this space. However, suppose that we have a function $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$. This function is all that is needed in order to write down the optimization problem and its solution. It turns out that if we know K then we do not need to know the function Φ in any explicit way. Specifically, let $k_{ij} = K(x_i, x_j)$. The learning task is to solve

$$\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j k_{ij} \quad \text{subject to} \quad 0 \leq \alpha_i \leq C.$$

The solution is

$$f(x) = [\sum_{i=1}^{n} \alpha_i y_i \Phi(x_i)] \cdot x = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x).$$

This classifier is a weighted combination of at most n functions, one for each support vector x_i. These functions are sometimes called basis functions.

[1] For many applications of support vector machines, it is advantageous to normalize features to have the same mean and variance. It can be advantageous also to normalize instances so that they have unit length. However, in general one cannot make both normalizations be true at the same time.

The result above says that in order to train a nonlinear SVM classifier, all that is needed is the matrix of size n by n whose entries are k_{ij}, which is called the kernel matrix. In order to apply the trained nonlinear classifier, that is to make predictions for arbitrary test examples, the kernel function K is needed. The function Φ never needs to be known explicitly. Using K exclusively in this way instead of Φ is called the "kernel trick." Practically, the function K can be much easier to deal with than Φ, because K is just a mapping to $\mathbb{R}$, rather than to a high-dimensional space $\mathbb{R}^{d'}$.

Intuitively, regardless of which kernel K is used, that is regardless of which re-representation Φ is used, the complexity of the classifier f is limited, since it is defined by at most n coefficients α_i. The function K is fixed, so it does not increase the intuitive complexity of the classifier.

5.6 Radial basis function kernels

One particular kernel is especially important. The radial basis function (RBF) kernel with parameter $\gamma > 0$ is the function

$$K(x_i, x) = \exp(-\gamma ||x_i - x||^2).$$

Note that regardless of γ the value of $K(x_i, x)$ is always between 0 and 1, with the maximum occurring only if $x = x_i$. Using an RBF kernel, each basis function $K(x_i, x)$ is "radial" since it is based on the Euclidean distance $||x_i - x||$ between a support vector x_i and a data point x. Intuitively, the RBF kernel places a bell shape around the support vector x_i. When γ, which is called the bandwidth parameter, is larger, then the bell shape is more peaked.

An important intuition about SVMs is that with an RBF kernel, the classifier $f(x) = \sum_i \alpha_i y_i K(x_i, x)$ is similar to a nearest-neighbor classifier. Given a test instance x, its predicted label $f(x)$ is a weighted average of the labels y_i of the support vectors x_i. The support vectors that contribute non-negligibly to the predicted label are those for which the Euclidean distance $||x_i - x||$ is small. When γ is larger, then fewer support vectors contribute non-negligibly.

The RBF kernel can also be written

$$K(x_i, x) = \exp(-||x_i - x||^2/\sigma^2)$$

where $\sigma^2 = 1/\gamma$. This notation emphasizes the similarity with a Gaussian distribution. A larger value for γ, i.e. a smaller value for σ^2, gives a basis functions that is more peaked, i.e. that is significantly above zero for a narrower range of x values. Using a larger value for σ^2 is similar to using a larger number k of neighbors in a nearest neighbor classifier.

There are some heuristics to narrow down reasonable values for the bandwidth parameter. Typically we want

$$K(x_i, x) = \exp(-||x_i - x||^2/\sigma^2) = O(1)$$

so setting σ^2 equal to a typical value of $||x_i - x||^2$ is sensible. One specific idea is to try the 10th, 50th, and 90th percentiles of the distribution of distances between negative and positive training examples.

5.7 Selecting the best SVM settings

Getting good results with support vector machines requires some care. The consensus opinion is that the best SVM classifier is typically at least as accurate as the best of any other type of classifier, but obtaining the best SVM classifier is not trivial. The following procedure is recommended as a starting point:

1. Code data in the numerical format needed for SVM training.
2. Scale each attribute to have range 0 to 1, or to have range -1 to $+1$, or to have mean zero and unit variance.
3. Use cross-validation to find the best value for C for a linear kernel.
4. Use cross-validation to find the best values for C and γ for an RBF kernel.
5. Train on the entire available data using the parameter values found to be best via cross-validation.

It is reasonable to start with $C = 1$ and $\gamma = 1$, and to try values that are smaller and larger by factors of 2:

$$\langle C, \gamma \rangle \in \{\ldots, 2^{-3}, 2^{-2}, 2^{-1}, 1, 2, 2^2, 2^3, \ldots\}^2.$$

This process is called a grid search; it is easy to parallelize of course. It is important to try a wide enough range of values to be sure that more extreme values do not give better performance. Once you have found the best values of C and γ to within a factor of 2, doing a more fine-grained grid search around these values is sometimes beneficial, but not always.

Exercises

1.

(a) Draw the hinge loss function for the case where the true label $y = 1$. Label the axes clearly.
(b) Explain where the derivative is (i) zero, (ii) constant but not zero, or (iii) not defined.
(c) For each of the three cases for the derivative, explain its intuitive implications for training an SVM classifier.

2.

As seen in class, the hinge loss function is

$$l(f(x), -1) = \max\{0, 1 + f(x)\}$$
$$l(f(x), +1) = \max\{0, 1 - f(x)\}$$

(a) Explain what the two arguments of the hinge loss function are.
(b) Explain the reason why $l(2, 1) = l(1, 1)$ but $l(-2, 1) \neq l(-1, 1)$.

3.

Suppose that you have trained SVM classifiers carefully for a given learning task. You have selected the settings that yield the best linear classifier and the best RBF classifier. It turns out that both classifiers perform equally well in terms of accuracy for your task.

(a) Now you are given many times more training data. After finding optimal settings again and retraining, do you expect the linear classifier or the RBF classifier to have better accuracy? Explain very briefly.

(b) Do you expect the optimal settings to involve stronger regularization, or weaker regularization? Explain very briefly.

CSE 255 assignment due on April 23, 2013

As before, do this assignment in a team of exactly two people, and choose a partner with a background different from yours. You may keep the same partner, or change.

The goal of this assignment is to train a support vector machine (SVM) classifier that has the best achievable, believable, 0/1 accuracy on a medical task. You should choose one of three datasets that you can find at `http://archive.ics.uci.edu/ml/datasets.html`: the Pima Indians Diabetes dataset, the Breast Cancer Wisconsin (Original) dataset, or the Breast Cancer Wisconsin (Diagnostic) dataset. Each of these has a binary class label, 569 to 768 examples, and 8 to 32 features. Be sure to read the data documentation carefully, in particular to understand the meaning of values that are zero.

Train SVM models to predict the target label as accurately as possible. Recode non-numerical features as numerical, and transform features to improve their usefulness. Train the best possible linear SVM, and also the best possible SVM using a radial basis function (RBF) kernel. The outcome should be the two SVM classifiers with highest 0/1 accuracy that you can find, without overfitting or underfitting.

Using the software named LIBSVM and LIBLINEAR is recommended. LibSVM is available in R in a package with the obscure name `e1071`. Because these are fast, you can explore models based on a large number of transformed features. Note that training nonlinear SVMs is much slower than training linear SVMs.

Use cross-validation and grid search carefully to find the best settings for training, and to evaluate the accuracy of the best classifiers as fairly as possible. Identify good values for the soft-margin parameter for both kernels, and for the width parameter of the RBF kernel. Follow carefully the guidelines for using SVMs correctly in `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`.

Look for a recent published paper that provides results on the dataset that you have selected. The paper you find can use any type of learning algorithm. In your report, discuss how your results compare with those in this paper. Discuss whether that paper uses good methodology to avoid overfitting, and whether the accuracy that it reports is reliable.

As before, the deliverable is a short report that is organized well, written well, and formatted well. Describe what works, and describe your results. Do not speculate about ideas that you have not tried, and do not write about ideas that do not work. Discuss any limitations on the validity or reliability of your results. Explain clearly your cross-validation procedure, and describe your final linear and RBF models. In general, a report should include enough detail for someone else to replicate your results exactly, but no more. Write in the present tense, and organize the report logically, not chronologically.

Chapter 6

Doing valid experiments

Consider a classifier that predicts membership in one of two classes, called positive and negative. Suppose that a set of examples has a certain total size n, say $n = 1000$. We can represent the performance of any classifier with a table as follows:

		predicted	
		positive	negative
truth	positive	tp	fn
	negative	fp	tn

where $tp + fn + fp + tn = n$. A table like the one above is called a 2×2 confusion matrix. Rows correspond to actual labels, while columns correspond to predicted labels.[1]

The four entries in a 2×2 confusion matrix have standard names. They are true positives tp, false positives fp, true negatives tn, and false negatives fn, It is important to remember that these numbers are counts, i.e. integers, not ratios, i.e. fractions. Depending on the application, different summaries are computed from a confusion matrix. In particular, accuracy is the ratio $(tp + tn)/n$.

6.1 Cross-validation

Usually we have a fixed database of labeled examples available, and we are faced with a dilemma: we would like to use all the examples for training, but we would

[1] It would be equally valid to swap the rows and columns. Unfortunately, there is no standard convention about whether rows are actual or predicted. Remember that there is a universal convention that in notation like x_{ij} the first subscript refers to rows while the second subscript refers to columns.

also like to use many examples as an independent test set. Cross-validation is a procedure for overcoming this dilemma. It is the following algorithm.

Input: Training set S, integer constant k
Procedure:
  partition S into k disjoint equal-sized subsets $S_1, \ldots, S_k$
  for $i = 1$ to $i = k$
    let $T = S$ with S_i removed
    run learning algorithm with T as training set
    test the resulting classifier on S_i obtaining tp_i, fp_i, tn_i, fn_i
  compute summed counts:
    $tp = \sum_{i=1}^{k} tp_i$, $fp = \sum_{i=1}^{k} fp_i$, $tn = \sum_{i=1}^{k} tn_i$, $fn = \sum_{i=1}^{k} fn_i$

The output of cross-validation is a confusion matrix based on using each labeled example as a test example exactly once. Whenever an example is used for testing a classifier, it has not been used for training that classifier. Hence, the confusion matrix obtained by cross-validation is a fair indicator of the performance of the learning algorithm on independent test examples.

Any learning algorithm, and any loss function, may be used during training on each set T. In general the loss function used during training is different from the loss function that is evaluated on the validation folds S_i. The former is typically regularized, while the latter is typically unregularized. It does not make sense to compare regularized loss function values for different regularization strengths. However, it does make sense to train on the same training data with multiple learning algorithms, and/or multiple strengths of regularization, and to compare the loss achieved by the alternative trained classifiers on a common validation set, or validation folds.

If n labeled examples are available, then the largest possible number of folds is $k = n$. This special case is called leave-one-out cross-validation (LOOCV). However, the time complexity of cross-validation is k times that of running the training algorithm once, so often LOOCV is computationally infeasible. In recent research the most common choice for k is 10.

A disadvantage of cross-validation is that it does not produce any single final classifier. Also, the confusion matrix that it provides is not the performance of any specific single classifier. Instead, this matrix is an estimate of the average performance of a classifier learned from a training set of size $(k-1)n/k$ where n is the size of S. The common procedure is to create a final classifier by training on all of S, and then to use the confusion matrix obtained from cross-validation as an informal estimate of the performance of this classifier. This estimate is likely to be conservative in the sense that the final classifier may have slightly better performance since it is based on a slightly larger training set.

Suppose that the time to train a classifier is proportional to the number of training examples, and the time to make predictions on test examples is negligible in comparison. The time required for k-fold cross-validation is then $O(k \cdot (k-1)/k \cdot n) = O((k-1)n)$. Three-fold cross-validation is therefore twice as time-consuming as two-fold. This suggests that for preliminary experiments, two-fold is a good choice.

The results of cross-validation can be misleading. For example, if each example is duplicated in the training set and we use a nearest-neighbor classifier, then LOOCV will show a zero error rate. Cross-validation with other values of k will also yield misleadingly low error estimates.

Subsampling means omitting examples from some classes. Subsampling the common class is a standard approach to learning from unbalanced data, i.e. data where some classes are very common while others are very rare. It is reasonable to do subsampling in training data, including in the training folds of cross-validation. However, reported performance numbers should always be based on a set of test examples that is directly typical of a genuine test population. A common mistake with cross-validation is to do subsampling on the test set. This leads to estimates of performance on future unbalanced data that are simply wrong.

6.2 Cross-validation for model selection

A model selection procedure is a method for choosing the best classifier, or learning algorithm, from a set of alternatives. Often, the alternatives are all the same algorithm but with different parameter settings, for example SVMs with different C and γ values. Another common case is where the alternatives are different subsets of features. Typically, the number of alternatives is finite and the only way to evaluate an alternative is to run it explicitly on a training set. So, how should we do model selection?

A simple way to apply cross-validation for model selection is the following:

Input: dataset S, integer k, set V of alternative algorithm settings
Procedure:
    partition S randomly into k disjoint subsets $S_1, \ldots, S_k$ of equal size
    for each setting v in V
        for $i = 1$ to $i = k$
            let $T = S \setminus S_i$
            run the learning algorithm with setting v and T as training set
            apply the trained model to S_i obtaining performance e_i
        end for
        let $M(v)$ be a summary of e_i for all i
    end for

$$\text{select } \hat{v} = \text{argmax}_v \, M(v)$$

The output of this model selection procedure is $\hat{v}$. The input set V of alternative settings can be a grid of parameter values. The most obvious summary of the validation performance values e_i is their average, but other summaries could be used also.

Viewed as a loss function, 0/1 accuracy, and other measures based on the confusion matrix such as precision and recall, are a discontinuous function of the classifier f. This means that different classifiers may have identical 0/1 accuracy. Using a continuous loss function for each e_i, such as square loss, makes finer distinctions between classifiers and can lead to less variability in the setting v that is best for each test fold.

Notice that the division of S into subsets happens just once, and the same division is used for all settings v (also called hyperparameters) . This choice reduces the random variability in the evaluation of different v. A new partition of S could be created for each setting v, but this would not overcome the issue that $\hat{v}$ is chosen to optimize performance on S.

Trying every combination of hyperparameters in some set V explicitly can be prohibitively expensive computationally. It may be preferable to search in V in a more clever way, using a genetic algorithm or some other heuristic method. The search for a good member of the set V, that is a procedure for selecting hyperparameter values v, is itself part of the learning algorithm, so this search can be intelligent and/or heuristic.

The setting $\hat{v}$ is chosen to maximize $M(v)$, so $M(\hat{v})$ is not a fair estimate of the performance to be expected from $\hat{v}$ on future data. Stated another way, $\hat{v}$ is chosen to optimize performance on all of S, so $\hat{v}$ is likely to overfit S. Researchers have proposed procedures called nested cross-validation to address this level of overfitting. However, no single nested cross-validation procedure is widely accepted and computationally efficient. In practice, people usually tolerate the fact that the chosen hyperparameters $\hat{v}$ are likely to overfit S. The smaller the number of different hyperparameter values v that have been compared, the less serious this overfitting is likely to be.

Exercises

1.

The following cross-validation procedure is intended to find the best regularization parameter R for linear regression, and to report a fair estimate of the RMSE to be expected on future test data.

Input: dataset S, integer k, set V of alternative R values
Procedure:
  partition S randomly into k disjoint subsets $S_1, \ldots, S_k$ of equal size
  for each R value in V
    for $i = 1$ to $i = k$
      let $T = S \setminus S_i$
      train linear regression with parameter R and T as training set
      apply the trained regression equation to S_i obtaining SSE e_i
    end for
    compute $M = \sum_{i=1}^{k} e_i$
    report R and $RMSE = \sqrt{M/|S|}$
  end for

(a) Suppose that you choose the R value for which the reported RMSE is lowest. Explain why this method is likely to be overoptimistic, as an estimate of the RMSE to be expected on future data. *Each R value is being evaluated on the entire set S. The value that seems to be best is likely overfitting this set.*

(b) Briefly, suggest an improved variation of the method above. *The procedure to select a value for R is part of the learning algorithm. This whole procedure should be evaluated using a separate test set, or via cross-validation,*

Additional notes. Incorrect answers include the following:

- "The partition of S should be stratified." No; first of all, we are doing regression, so stratification is not well-defined, and second, failing to stratify increases variability but not does not cause overfitting.
- "The partition of S should be done separately for each R value, not just once." No; a different partition for each R value might increase the variability in the evaluation of each R, but it would not change the fact that the best R is being selected according to its performance on all of S.

Two basic points to remember are that it is never fair to evaluate a learning method on its training set, and that any search for settings for a learning algorithm (e.g. search for a subset of features, or for algorithmic parameters) is part of the learning method.

Chapter 7

Classification with a rare class

In many data mining applications, the goal is to find needles in a haystack. That is, most examples are negative but a few examples are positive. The goal is to identify the rare positive examples, as accurately as possible. For example, most credit card transactions are legitimate, but a few are fraudulent. We have a standard binary classifier learning problem, but both the training and test sets are unbalanced. In a balanced set, the fraction of examples of each class is about the same. In an unbalanced set, some classes are rare while others are common.

A major difficulty with unbalanced data is that accuracy is not a meaningful measure of performance. Suppose that 99% of credit card transactions are legitimate. Then we can get 99% accuracy by predicting trivially that every transaction is legitimate. On the other hand, suppose we somehow identify 5% of transactions for further investigation, and half of all fraudulent transactions occur among these 5%. Clearly the identification process is doing something worthwhile and not trivial. But its accuracy is only 95%.

For concreteness in further discussion, we will consider only the two class case, and we will call the rare class positive. Rather than talk about fractions or percentages of a set, we will talk about actual numbers (also called counts) of examples. It turns out that thinking about actual numbers leads to less confusion and more insight than thinking about fractions. Suppose the test set has a certain total size n, say $n = 1000$. We can represent the performance of the trivial classifier as the following confusion matrix:

		predicted	
		positive	negative
truth	positive	0	10
	negative	0	990

The performance of the non-trivial classifier is

		predicted	
		positive	negative
truth	positive	5	5
	negative	45	945

For supervised learning with discrete predictions, only a confusion matrix gives complete information about the performance of a classifier. No single number, for example accuracy, can provide a full picture of a confusion matrix, or of the usefulness of a classifier. Assuming that the total number of examples n is fixed, three of the counts in a confusion matrix can vary independently. When writing a report, it is best to give the full confusion matrix explicitly, so that readers can calculate whatever performance measurements they are most interested in.

When accuracy is not meaningful, two summary measures that are commonly used are called precision and recall. Remember from Chapter 6 that in general a confusion matrix has the form

		predicted	
		positive	negative
truth	positive	tp	fn
	negative	fp	tn

where $tp + fn + fp + tn = n$. Precision and recall are defined as follows:

- precision $p = tp/(tp + fp)$, and
- recall $r = tp/(tp + fn)$.

The names "precision" and "recall" come from the field of information retrieval. In other research areas, recall is often called sensitivity, while precision is sometimes called positive predictive value.

Precision is undefined for a classifier that predicts that every test example is negative, that is when $tp + fp = 0$. Worse, precision can be misleadingly high for a classifier that predicts that only a few test examples are positive. Consider the following confusion matrix:

		predicted	
		positive	negative
truth	positive	1	9
	negative	0	990

Precision is 100% but 90% of actual positives are missed. F-measure is a widely used metric that overcomes this limitation of precision. It is the harmonic average of precision and recall:

$$F = \frac{2}{1/p + 1/r} = \frac{2pr}{r + p}.$$

For the confusion matrix above $F = 2 \cdot 0.1/(1 + 0.1) = 0.18$.

Besides accuracy, precision, recall, and F-measure (which is also called F1), many other summaries are also commonly computed from confusion matrices. Some of these are called specificity, false positive rate, false negative rate, positive and negative likelihood ratio, kappa coefficient, and more. Rather than rely on agreement upon and understanding of the definitions of these, it is preferable simply to report a full confusion matrix explicitly.

7.1 Thresholds and lift

A confusion matrix is always based on discrete predictions. Often, however, these predictions are obtained by thresholding a real-valued predicted score. For example, an SVM classifier yields a real-valued score $f(x)$ which is then compared to the threshold zero to obtain a discrete yes/no prediction. No single confusion matrix can represent information about the overall usefulness of underlying real-valued predictions. We shall return to this issue below, but first we shall consider the issue of selecting a threshold.

Setting the threshold determines the number $tp + fp$ of examples that are predicted to be positive. In some scenarios, there is a natural threshold such as zero for an SVM. However, even when a natural threshold exists, it is possible to change the threshold to achieve a target number of positive predictions. This target number is often based on a so-called budget constraint. Suppose that all examples predicted to be positive are subjected to further investigation. An external limit on the resources available will determine how many examples can be investigated. This number is a natural target for the value $fp+tp$. Of course, we want to investigate those examples that are most likely to be actual positives, so we want to investigate the examples x with the highest prediction scores $f(x)$. Therefore, the correct strategy is to choose a threshold t such that we investigate all examples x with $f(x) \geq t$ and the number of such examples is determined by the budget constraint.

Given that a fixed number of examples are predicted to be positive, a natural question is how good a classifier is at capturing the actual positives within this number. This question is answered by a measure called lift. Intuitively, a lift of 2 means that actual positives are twice as dense among the predicted positives as they are among all examples.

The precise definition of lift is a bit complex. First, let the fraction of examples predicted to be positive be the prediction rate $x = (tp+fp)/n$. Next, let the base rate of actual positive examples be $b = (tp + fn)/n$. Let the density of actual positives within the predicted positives be $d = tp/(tp + fp)$. Now, the lift at prediction rate x is defined to be the ratio d/b. Note that the lift will be different at different prediction rates. Typically, the smaller the prediction rate the higher the lift.

Lift can be expressed as

$$\begin{aligned} \frac{d}{b} &= \frac{tp/(tp+fp)}{(tp+fn)/n} = \frac{tp \cdot n}{(tp+fp)(tp+fn)} \\ &= \frac{tp}{tp+fn}\frac{n}{tp+fp} = \frac{recall}{prediction\ rate}. \end{aligned}$$

Lift is a useful measure of success if the number of examples that should be predicted to be positive is determined by external considerations. However, budget constraints should normally be questioned. In the credit card scenario, perhaps too many transactions are being investigated and the marginal benefit of investigating some transactions is negative. Or, perhaps too few transactions are being investigated; there would be a net benefit if additional transactions were investigated. Making optimal decisions about how many examples should be predicted to be positive is discussed in Chapter 9.

While a budget constraint is not normally a rational way of choosing a threshold for predictions, it is still more rational than choosing an arbitrary threshold. In particular, the threshold zero for an SVM classifier has some mathematical meaning but is not a rational guide for making decisions.

7.2 Ranking examples

Applying a threshold to a classifier with real-valued outputs loses information, because the distinction between examples on the same side of the threshold is lost. Moreover, at the time a classifier is trained and evaluated, it is often the case that the threshold to be used for decision-making is not known. Therefore, it is useful to compare different classifiers across the range of all possible thresholds.

Typically, it is not meaningful to use the same numerical threshold directly for different classifiers. However, it is meaningful to compare the recall achieved by different classifiers when the threshold for each one is set to make their precisions equal. This is what an ROC curve does.[1] Concretely, an ROC curve is a plot of

[1] ROC stands for "receiver operating characteristic." This terminology originates in the theory of detection based on electromagnetic (radio) waves. The receiver is an antenna and its operating characteristic is the sensitivity that it is tuned for.

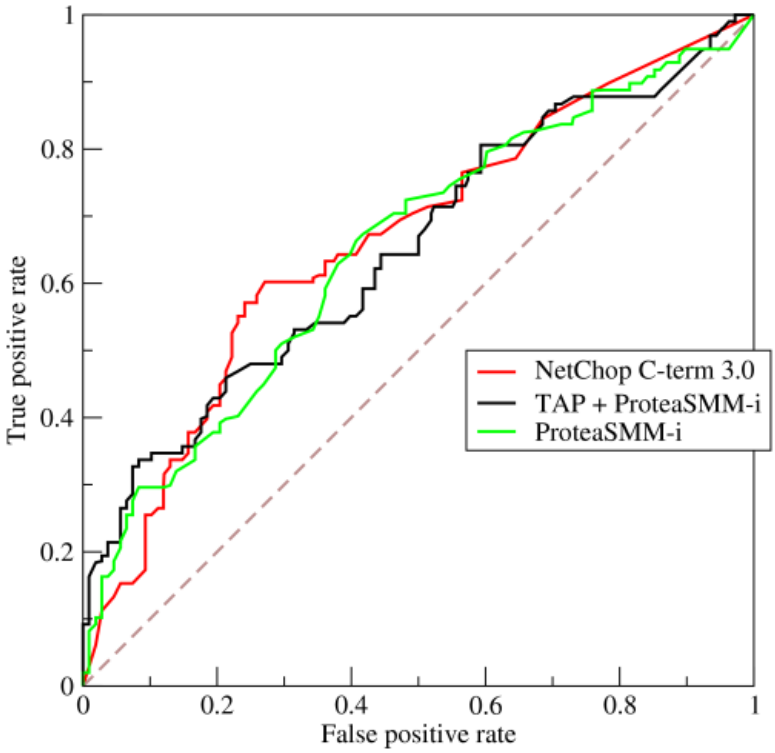

Figure 7.1: ROC curves for three alternative binary classifiers. Source: Wikipedia.

the performance of a classifier, where the horizontal axis measures false positive rate (fpr) and the vertical axis measures true positive rate (tpr). These are defined as

$$fpr = \frac{fp}{tn + fp}$$
$$tpr = \frac{tp}{tp + fn}.$$

Note that tpr is the same as recall and is sometimes also called "hit rate."

In an ROC plot, the ideal point is at the top left. One classifier uniformly dominates another if its curve is always above the other's curve. It happens often that the ROC curves of two classifiers cross, which implies that neither one dominates the other uniformly.

ROC plots are informative, but do not provide a quantitative measure of the performance of classifiers. A natural quantity to consider is the area under the ROC curve, often abbreviated AUC. The AUC is 0.5 for a classifier whose predictions are

no better than random, while it is 1 for a perfect classifier. The AUC has an intuitive meaning: it can be proved that it equals the probability that the classifier will rank correctly a randomly chosen pair of examples, where one member of the pair is actually positive and the other is negative.

One reason why AUC is widely used is that, as shown by the probabilistic meaning just mentioned, it has built into it the implicit assumption that the rare class is more important than the common class. Essentially, AUC treats each class as equally important, regardless of how unbalanced the classes are. Unfortunately, there is no natural generalization of AUC to scenarios with three or more classes.

7.3 Training to overcome imbalance

The previous sections were about measuring performance in applications where one class is rare but important. This section describes ideas for actually performing better on such applications. Of course, one idea is to change the threshold of a classifier that outputs a real-valued score.

The next most obvious idea is to undersample the common class. That is, include in the training set only a randomly chosen subset of the available negative (common) examples. A major disadvantage of this approach is that it loses information. It does however make training faster, which can be an important benefit.

Consider the regularized loss function that is minimized by many training algorithms:

$$c(f) + C \sum_{i=1}^{n} l(f(x_i), y_i).$$

The parameter C controls the trade-off between regularization and minimizing empirical loss. The empirical loss function is a sum that treats all n training examples equally. If most training examples belong to one class, then most pressure will be to achieve high accuracy on that class. One response to class imbalance is to give different importances to each class in the optimization task. Specifically, we can minimize

$$c(f) + C_{-1} \sum_{i:y_i=-1} l(f(x_i), -1) + C_1 \sum_{i:y_i=1} l(f(x_i), 1).$$

Good values of the two C parameters can be chosen by cross-validation, or their ratio can be fixed so that each class has equal importance. Many SVM software packages allow the user to specify these two different C parameters. In our notation, intuitively C_1 should be higher, since the positive class is the rare class.

A subtle idea is that we can actually optimize AUC of a linear classifier directly on the training set. The empirical AUC of a linear classifier with coefficients w is

$$A = \frac{1}{n_{-1}n_1} \sum_{i:y_i=1} \sum_{j:y_j=-1} I(w \cdot x_i > w \cdot x_j)$$

where $\cdot$ is dot product as before. This is the same as 0/1 accuracy on a new set Q:

$$A = \frac{1}{|Q|} \sum_{\langle i,j \rangle \in Q} I(I(w \cdot x_i > w \cdot x_j) = I(y_i > y_j)).$$

where $Q = \{\langle i, j \rangle : y_i \neq y_j\}$. Now, note that $w \cdot x_i > w \cdot x_j$ is equivalent to $w \cdot (x_i - x_j) > 0$. This observation suggests training a linear classifier w that tries to make every example $x_i - x_j$ have the 0/1 label $I(y_i > y_j)$. Specifically, we minimize

$$c(f) + C \sum_{\langle i,j \rangle \in Q} l(w \cdot (x_i - x_j), y_i).$$

The training set Q for this learning task is balanced because if the pair $\langle i, j \rangle$ belongs to Q then so does $\langle j, i \rangle$. Note also that if $y_i = 1$ if and only if $y_j = -1$ so we want $x_i - x_j$ to have the label 1 if and only if $y_i = 1$.

The set Q can have size up to $n^2/4$ if the original training set has size n. However, the minimization can often be done efficiently by stochastic gradient descent, picking elements $\langle i, j \rangle$ from Q randomly. The weight vector w typically reaches convergence after seeing only $O(n)$ elements of Q.

7.4 A case study in classification with a rare class

Link prediction is the task of recognizing which nodes in a graph have an edge between them. For example, the nodes may be proteins in an organism, with an edge between two nodes if and only if the two proteins have a biochemical relationship. This specific task is called predicting protein-protein interactions. Here, each example is a pair of proteins. The positive class consists of pairs that do interact, while the negative class contains all other pairs. Most pairs of proteins do not interact, so the positive class is rare. Hence, this task is a special case of classifier learning with a rare class.

This section was originally an assignment. The goal of formulating it as an assignment was to help develop three important abilities. The first ability is critical thinking, i.e. the skill of identifying what is most important and then identifying

what may be incorrect. The second ability is understanding a new application domain quickly, in this case a task in computational biology. The third ability is presenting opinions in a persuasive way. Students were asked to explain arguments and conclusions crisply, concisely, and clearly.

> **Assignment:** The paper you should read is *Predicting protein-protein interactions from primary structure* by Joel Bock and David Gough, published in the journal *Bioinformatics* in 2001. Figure out and describe three major flaws in the paper. The flaws concern
>
> - how the dataset is constructed,
> - how each example is represented, and
> - how performance is measured and reported.
>
> Each of the three mistakes is serious. The paper has 248 citations according to Google Scholar as of May 26, 2009, but unfortunately each flaw by itself makes the results of the paper not useful as a basis for future research. Each mistake is described sufficiently clearly in the paper: it is a sin of commission, not a sin of omission.
>
> The second mistake, how each example is represented, is the most subtle, but at least one of the papers citing this paper does explain it clearly. It is connected with how SVMs are applied here. Remember the slogan: "If you cannot represent it then you cannot learn it."
>
> Separately, provide a brief critique of the four benefits claimed for SVMs in the section of the paper entitled *Support vector machine learning*. Are these benefits true? Are they unique to SVMs? Does the work described in this paper take advantage of them?

Sample answers: Here is a brief summary of what I see as the most important issues with the paper.

(1) How the dataset is constructed. The problem here is that the negative examples are not pairs of genuine proteins. Instead, they are pairs of randomly generated amino acid sequences. It is quite possible that these artificial sequences could not fold into actual proteins at all. The classifiers reported in this paper may learn mainly to distinguish between real proteins and non-proteins.

The authors acknowledge this concern, but do not overcome it. They could have used pairs of genuine proteins as negative examples. It is true that one cannot be sure that any given pair really is non-interacting. However, the great majority of pairs do not interact. Moreover, if a negative example really is an interaction, that will presumably slightly reduce the apparent accuracy of a trained classifier, but not change overall conclusions.

(2) How each example is represented. This is a subtle but clear-cut and important issue, assuming that the research uses a linear classifier.

Let x_1 and x_2 be two proteins and let $f(x_1)$ and $f(x_2)$ be their representations as vectors in $\mathbb{R}^d$. The pair of proteins is represented as the concatenated vector $\langle f(x_1)\ f(x_2)\rangle \in \mathbb{R}^{2d}$. Suppose a trained linear SVM has parameter vector w. By definition $w \in \mathbb{R}^{2d}$ also. (If there is a bias coefficient, so $w \in \mathbb{R}^{2d+1}$, the conclusion is the same.)

Now suppose the first protein x_1 is fixed and consider varying the second protein x_2. Proteins x_2 will be ranked according to the numerical value of the dot product $w \cdot \langle f(x_1)\ f(x_2)\rangle$. This is equal to $w_1 \cdot f(x_1) + w_2 \cdot f(x_2)$ where the vector w is written as $\langle w_1\ w_2\rangle$. If x_1 is fixed, then the first term is constant and the second term $w_2 \cdot f(x_2)$ determines the ranking. The ranking of x_2 proteins will be the same regardless of what the x_1 protein is. This fundamental drawback of linear classifiers for predicting interactions is pointed out in [Vert and Jacob, 2008, Section 5].

With a concatenated representation of protein pairs, a linear classifier can at best learn the propensity of individual proteins to interact. Such a classifier cannot represent patterns that depend on features that are true only for specific protein pairs. This is the relevance of the slogan "If you cannot represent it then you cannot learn it."

Note: Previously I was under the impression that the authors stated that they used a linear kernel. On rereading the paper, it fails to mention at all what kernel they use. If the research uses a linear kernel, then the argument above is applicable.

(3) How performance is measured and reported. Most pairs of proteins are non-interacting. It is reasonable to use training sets where negative examples (non-interacting pairs) are undersampled. However, it is not reasonable or correct to report performance (accuracy, precision, recall, etc.) on test sets where negative examples are under-represented, which is what is done in this paper.

As for the four claimed advantages of SVMs:

1. SVMs are nonlinear while requiring "relatively few" parameters.

   The authors do not explain what kernel they use. In any case, with a nonlinear kernel the number of parameters is the number of support vectors, which is often close to the number of training examples. It is not clear relative to what this can be considered "few."

2. SVMs have an analytic upper bound on generalization error.

   This upper bound does motivate the SVM approach to training classifiers, but it typically does not provide useful guarantees for specific training sets. In any case, a bound of this type has nothing to do with assigning confidences to individual predictions. In practice overfitting is prevented by straightforward

search for the value of the C parameter that is empirically best, not by applying a theorem.

3. SVMs have fast training, which is essential for screening large datasets.

   SVM training is slow compared to many other classifier learning methods, except for linear classifiers trained by fast algorithms that were only published after 2001, when this paper was published. As mentioned above, a linear classifier is not appropriate with the representation of protein pairs used in this paper.

   In any case, what is needed for screening many test examples is fast classifier application, not fast training. Applying a linear classifier is fast, whether it is an SVM or not. Applying a nonlinear SVM typically has the same order-of-magnitude time complexity as applying a nearest-neighbor classifier, which is the slowest type of classifier in common use.

4. SVMs can be continuously updated in response to new data.

   At least one algorithm is known for updating an SVM given a new training example, but it is not cited in this paper. I do not know any algorithm for training an optimal new SVM efficiently, that is without retraining on old data. In any case, new real-world protein data arrives slowly enough that retraining from scratch is feasible.

CSE 255 assignment due on April 30, 2013

As before, you should work with one partner, and you may change partner, or keep the same partner.

Like the second assignment, this fourth assignment uses the KDD98 data. However, you should now use the entire dataset. The goal is to train a classifier with real-valued outputs that identifies test examples with `TARGET_B` $= 1$. The measure of success to optimize is AUC, the area under the ROC curve. As explained in class, AUC is also the probability that a randomly chosen example that is genuinely positive has a higher score than a random negative example.

You should compare (i) a linear classifier and (ii) any nonlinear supervised learning method of your choice. For the latter, you may use a nonlinear SVM, a method based on decision trees such as random forests, a neural network, or any other method that seems interesting to you. Note that nonlinear SVMs are slow on a dataset as large as this one. Also note that the Liblinear implementation of linear classifiers is much faster than the LibSVM implementation. As before, recode and transform features to improve their usefulness. Do feature selection to reduce the size of the training set as much as is reasonably possible. Handle missing values in a sensible but simple way. Describe all preprocessing in your report, with enough detail for someone else to be able to reproduce your results.

Both learning methods should be applied to identical training and validation sets. The objective is to see whether the nonlinear method can perform better than the linear method, using the same data coded in the same way. For both methods, apply cross-validation to find good hyperparameter values.

In the same way as for the second assignment, after you have trained the linear and nonlinear classifiers that you think are the best possible, apply the two classifiers to the test set `cup98val.zip`. Remember to use the CONTROLN feature to join the `cup98val.zip` and `valtargt.txt` files correctly. Compute the AUC of your predictions with respect to the `TARGET_B` values in `valtargt.txt`. Write down your expected performance numbers *in advance*, to help avoid fooling yourselves.

In your report, explain your expected and achieved AUC results. Also compute 0/1 accuracy and F1 scores. What threshold should you use for these scores? Discuss how the achieved 0/1 accuracy and F1 scores compare to those obtained by a trivial classifier that predicts `TARGET_B` $= 0$ for every example.

Chapter 8

Learning to predict conditional probabilities

What is a conditional probability?

Reason 1 to want probabilities: predicting sums and averages.

Ideal versus achievable probability estimates.

Reason 2 to want probabilities: understanding predictive power.

Definition of well-calibrated probability.

Brier score.

Converting scores into probabilities.

8.1 Isotonic regression

Let f_i be prediction scores on a dataset, and let $y_i \in \{0, 1\}$ be the corresponding true labels. Let f^j be the f_i values sorted from smallest to largest, and let y^j be the y_i values sorted in the same order. For each f^j we want to find an output value g_j such that the g_j values are monotonically increasing, and the squared error of g_j relative to the y^j values is minimized. Formally, the optimization problem is

$$\min_{g_1,\ldots,g_n} (y^j - g_j)^2 \text{ subject to } g_j \leq g_{j+1} \text{ for } j = 1 \text{ to } j = n - 1.$$

It is a remarkable fact that if squared error is minimized, then the resulting predictions are well-calibrated probabilities.

There is an elegant algorithm called "pool adjacent violators" (PAV) that solves this problem in linear time. The algorithm is as follows, where pooling a set means replacing each member of the set by the arithmetic mean of the set.

Let $g_j = y^j$ for all j
Find the smallest j such that $g_j \not\leq g_{j+1}$
Pool g_j and g_{j+1}
Move left: If $g_{j-1} \not\leq g_j$ then pool g_{j-1} to g_{j+1}
Continue to the left until monotonicity is satisfied
Proceed to the right

Given a test example x, the procedure to predict a well-calibrated probability is as follows:

Apply the classifier to obtain $f(x)$
Find j such that $f^j \leq f(x) \leq f^{j+1}$
The predicted probability is g_j.

8.2 Univariate logistic regression

The disadvantage of isotonic regression is that it creates a lookup table for converting scores into estimated probabilities. An alternative is to use a parametric model. The most common model is called univariate logistic regression. The model is

$$\log \frac{p}{1-p} = a + bf$$

where f is a prediction score and p is the corresponding estimated probability.

The equation above shows that the logistic regression model is essentially a linear model with intercept a and coefficient b. An equivalent way of writing the model is

$$p = \frac{1}{1 + e^{-(a+bf)}}.$$

As above, let f_i be prediction scores on a training set, and let $y_i \in \{0, 1\}$ be the corresponding true labels. The parameters a and b are chosen to minimize the total loss

$$\sum_i l(p, y_i).$$

The precise loss function l that is typically used in the optimization is called conditional log likelihood (CLL), which is explained below. However, one could also use squared error, which would be consistent with isotonic regression.

8.3 Multivariate logistic regression

We assume that the label y follows a probability distribution that is different for different examples x. Technically, for each x there is a different distribution $p(Y = y|x; w)$ of y, but all these distributions share the same parameters w. In the binary case, the possible values y are $y = 1$ and $y = 0$.

Given training data consisting of $\langle x_i, y_i \rangle$ pairs, the principle of maximum conditional likelihood says to choose a parameter estimate $\hat{w}$ that maximizes the product $\prod_i f(y_i|x_i; w)$. Then, for any specific value of x, $\hat{w}$ can be used to predict the distribution of y given x.

We assume that we never want to predict values of x. If data points x themselves are always known, then there is no need to have a probabilistic model of them. In order to justify the conditional likelihood being a product, we just need to assume that the y_i are independent conditional on the x_i. We do not need to assume that the x_i are independent

When y is binary and x is a real-valued vector, the model

$$p(y = 1|x; \alpha, \beta) = \frac{1}{1 + \exp -[\alpha + \sum_{j=1}^{d} \beta_j x_j]}$$

is called logistic regression. We use j to index over the feature values x_1 to x_d of a single example of dimensionality d, since we use i below to index over training examples 1 to n.

The logistic regression model is easier to understand in the form

$$\log \frac{p}{1-p} = \alpha + \sum_{j=1}^{d} \beta_j x_j$$

where p is an abbreviation for $p(y = 1|x; \alpha, \beta)$. The ratio $p/(1 - p)$ is called the odds of the event $y = 1$ given x, and $\log[p/(1 - p)]$ is called the log odds. Since probabilities range between 0 and 1, odds range between 0 and $+\infty$ and log odds range unboundedly between $-\infty$ and $+\infty$. A linear expression of the form $\alpha + \sum_j \beta_j x_j$ can also take unbounded values, so it is reasonable to use a linear expression as a model for log odds, but not as a model for odds or for probabilities. Essentially, logistic regression is the simplest possible model for a random yes/no outcome that depends linearly on predictors x_1 to x_d.

For each feature j, $\exp(\beta_j x_j)$ is a multiplicative scaling factor on the odds $p/(1-p)$. If the predictor x_j is binary, then $\exp(\beta_j)$ is the extra odds of having the outcome $y = 1$ when $x_j = 1$, compared to when $x_j = 0$. If the predictor x_j is real-valued, then $\exp(\beta_j)$ is the extra odds of having the outcome $y = 1$ when the value of x_j

increases by one unit. A major limitation of the basic logistic regression model is that the probability p must either increase monotonically, or decrease monotonically, as a function of each predictor x_j. The basic model does not allow the probability to depend in a U-shaped way on any x_j.

Given the training set $\{\langle x_1, y_1\rangle, \ldots, \langle x_n, y_n\rangle\}$, training a logistic regression classifier involves maximizing the log conditional likelihood (LCL). This is the sum of the LCL for each training example:

$$LCL = \sum_{i=1}^{n} \log L(w; y_i|x_i) = \sum_{i=1}^{n} \log f(y_i|x_i; w).$$

Given a single training example $\langle x_i, y_i\rangle$, the log conditional likelihood is $\log p_i$ if the true label $y_i = 1$ and $\log(1 - p_i)$ if $y_i = 0$, where $p_i = p(y = 1|x_i; w)$.

$$y_i \log p_i + (1 - y_i) \log(1 - p_i)$$

Now write

$$\begin{aligned} p_i &= \frac{1}{1 + \exp -h_i} \\ 1 - p_i &= 1 - \frac{1}{1 + \exp -h_i} \\ &= \frac{\exp -h_i}{1 + \exp -h_i} \\ &= \frac{1}{(\exp h_i) + 1} \end{aligned}$$

So the LCL for one example is

$$-\log(1 + \exp -y_i h_i)$$

and maximizing the LCL is equivalent to minimizing

$$\sum_i \log(1 + \exp -y_i h_i).$$

The loss function inside the sum is called log loss. It has some similarity in shape to hinge loss.

8.4 Logistic regression and regularization

Consider learning a logistic regression classifier for categorizing documents. Suppose that word number j appears only in documents whose labels are positive. The

partial derivative of the log conditional likelihood with respect to the parameter for this word is

$$\frac{\partial}{\partial \beta_j} LCL = \sum_i (y_i - p_i) x_{ij}.$$

This derivative will always be positive, as long as the predicted probability p_i is not perfectly one for all these documents. Therefore, following the gradient will send β_j to infinity. Then, every test document containing this word will be predicted to be positive with certainty, regardless of all other words in the test document. This over-confident generalization is an example of overfitting.

There is a standard method for solving this overfitting problem that is quite simple, but quite successful. The solution is called regularization. The idea is to impose a penalty on the magnitude of the parameter values. This penalty should be minimized, in a trade-off with maximizing likelihood. Mathematically, the optimization problem to be solved is

$$\hat{\beta} = \underset{\beta}{\operatorname{argmax}} \; LCL - \lambda ||\beta||_2^2$$

where $||\beta||_2$ is the L_2 norm of the parameter vector, and the constant λ quantifies the trade-off between maximizing likelihood and minimizing parameter values.

This type of regularization, which we have seen in previous chapters also, is called quadratic or Tikhonov regularization. A major reason why it is popular is that it can be derived from several different points of view. In particular, it arises as a consequence of assuming a Gaussian prior on parameters. It also arises from theorems on minimizing generalization error, i.e. error on independent test sets drawn from the same distribution as the training set. And, it arises from robust classification: assuming that each training point lies in an unknown location inside a sphere centered on its measured location.

When a model is trained using a regularized loss function, then the predicted conditional probabilities obtained from the model are in theory not well-calibrated. In practice, predictions from regularized logistic regression models remain close to being well-calibrated.

Exercises

1.

The three parts below refer to a classifier learning task with two classes, where the base rate for the positive class $y = 1$ is 5%.

(a) Suppose that a probabilistic classifier predicts $p(y = 1|x) = c$ for some constant c, for all test examples x. Explain why $c = 0.05$ is the best value for c. *The value 0.05 is the only constant that is a well-calibrated probability. "Well-calibrated" means that* $c = 0.05$ *equals the average frequency of positive examples in sets of examples with predicted score* c.

(b) What is the error rate of the classifier from part (a)? What is its MSE? *With a prediction threshold of 0.5, all examples are predicted to be negative, so the error rate is 5%. The MSE is*

$$0.05 \cdot (1 - 0.05)^2 + 0.95 \cdot (0 - 0.05)^2 = 0.05 \cdot 0.95 \cdot 1$$

which equals 0.0475.

(c) Suppose that a well-calibrated probabilistic classifier satisfies $a \leq p(y = 1|x) \leq b$ for all x. What is the maximum possible lift at 10% of this classifier? *If the upper bound* b *is a well-calibrated probability, then the fraction of positive examples among the highest-scoring examples is at most* b. *Hence, the lift is at most* $b/0.05$ *where 0.05 is the base rate. Note that this is the highest possible lift at any percentage, not just at 10%.*

Sample assignment

This assignment uses the KDD98 data again. Also as before, you should train on the entire training set, and measure final success just once on the test set. As appropriate, you should do cross-validation on the training set, and/or create validation sets from it.

The goal is to train a classifier that will predict who will make a donation, that is to predict `TARGET_B`=1. However, the goal now is to make predictions that are accurate probabilities. Many classifiers, including SVMs, give scores that are numerical, but not correct probabilities. Logistic regression does give approximately well-calibrated probabilities. SVMs and other methods can be used also, but you must convert their outputs into reasonable probabilities in some way.

Measure the accuracy of predictions using mean squared error (MSE) Before you apply your final decision-making method to the test set (just once!), estimate a confidence interval for the MSE that you expect to achieve on the test set. This should be a range $[a, b]$ such that the probability is (say) 95% that the observed MSE will lie in this range. State your estimated confidence interval clearly in your report, and explain the reasoning behind it. Note that the central issue here is not any mathematical details such as specific probability distributions. Instead, it is thinking about where uncertainty arises, and how it is propagated through operations such as computing squared error.

Chapter 9

Making optimal decisions

This chapter discusses making optimal decisions based on predictions, and maximizing the value of customers.

9.1 Predictions, decisions, and costs

Decisions and predictions are conceptually very different. For example, a prediction concerning a patient may be "allergic" or "not allergic" to aspirin, while the corresponding decision is whether or not to administer the drug. Predictions can often be probabilistic, while decisions typically cannot.

Suppose that examples are credit card transactions and the label $y = 1$ designates a legitimate transaction. Then making the decision $y = 1$ for an attempted transaction means acting as if the transaction is legitimate, i.e. approving the transaction. The essence of cost-sensitive decision-making is that it can be optimal to act as if one class is true even when some other class is more probable. For example, if the cost of approving a fraudulent transaction is proportional to the dollar amount involved, then it can be rational not to approve a large transaction, even if the transaction is most likely legitimate. Conversely, it can be rational to approve a small transaction even if there is a high probability it is fraudulent.

Mathematically, let i be the predicted class and let j be the true class. If $i = j$ then the prediction is correct, while if $i \neq j$ the prediction is incorrect. The (i, j) entry in a cost matrix c is the cost of acting as if class i is true, when in fact class j is true. Here, predicting i means acting as if i is true, so one could equally well call this deciding i.

A cost matrix c has the following structure when there are only two classes:

	actual negative	actual positive
predict negative	$c(0,0) = c_{00}$	$c(0,1) = c_{01}$
predict positive	$c(1,0) = c_{10}$	$c(1,1) = c_{11}$

The cost of a false positive is c_{10} while the cost of a false negative is c_{01}. We follow the convention that cost matrix rows correspond to alternative predicted classes, while columns correspond to actual classes. In short the convention is row/column = i/j = predicted/actual. (This convention is the opposite of the one in Chapter 6 so perhaps we should switch one of these to make the conventions similar.)

The optimal prediction for an example x is the class i that minimizes the expected cost

$$e(x,i) = \sum_j p(j|x)c(i,j). \tag{9.1}$$

For each i, $e(x,i)$ is an expectation computed by summing over the alternative possibilities for the true class of x. In this framework, the role of a learning algorithm is to produce a classifier that for any example x can estimate the probability $p(j|x)$ of each class j being the true class of x.

9.2 Cost matrix properties

Conceptually, the cost of labeling an example incorrectly should always be greater than the cost of labeling it correctly. For a 2x2 cost matrix, this means that it should always be the case that $c_{10} > c_{00}$ and $c_{01} > c_{11}$. We call these conditions the "reasonableness" conditions.

Suppose that the first reasonableness condition is violated, so $c_{00} \geq c_{10}$ but still $c_{01} > c_{11}$. In this case the optimal policy is to label all examples positive. Similarly, if $c_{10} > c_{00}$ but $c_{11} \geq c_{01}$ then it is optimal to label all examples negative. (The reader can analyze the case where both reasonableness conditions are violated.)

For some cost matrices, some class labels are never predicted by the optimal policy given by Equation (9.1). The following is a criterion for when this happens. Say that row m dominates row n in a cost matrix C if for all j, $c(m,j) \geq c(n,j)$. In this case the cost of predicting n is no greater than the cost of predicting m, regardless of what the true class j is. So it is optimal never to predict m. As a special case, the optimal prediction is always n if row n is dominated by all other rows in a cost matrix. The two reasonableness conditions for a two-class cost matrix imply that neither row in the matrix dominates the other.

Given a cost matrix, the decisions that are optimal are unchanged if each entry in the matrix is multiplied by a positive constant. This scaling corresponds to changing the unit of account for costs. Similarly, the decisions that are optimal are unchanged

if a constant is added to each entry in the matrix. This shifting corresponds to changing the baseline away from which costs are measured. By scaling and shifting entries, any two-class cost matrix

c_{00}	c_{01}
c_{10}	c_{11}

that satisfies the reasonableness conditions can be transformed into a simpler matrix that always leads to the same decisions:

0	c'_{01}
1	c'_{11}

where $c'_{01} = (c_{01} - c_{00})/(c_{10} - c_{00})$ and $c'_{11} = (c_{11} - c_{00})/(c_{10} - c_{00})$. From a matrix perspective, a 2x2 cost matrix effectively has two degrees of freedom.

9.3 The logic of costs

Costs are not necessarily monetary. A cost can also be a waste of time, or the severity of an illness, for example. Although most recent research in machine learning has used the terminology of costs, doing accounting in terms of benefits is generally preferable, because avoiding mistakes is easier, since there is a natural baseline from which to measure all benefits, whether positive or negative. This baseline is the state of the agent before it takes a decision regarding an example. After the agent has made the decision, if it is better off, its benefit is positive. Otherwise, its benefit is negative.

When thinking in terms of costs, it is easy to posit a cost matrix that is logically contradictory because not all entries in the matrix are measured from the same baseline. For example, consider the so-called German credit dataset that was published as part of the Statlog project [Michie et al., 1994]. The cost matrix given with this dataset at `http://www.sgi.com/tech/mlc/db/german.names` is as follows:

	actual bad	actual good
predict bad	0	1
predict good	5	0

Here examples are people who apply for a loan from a bank. "Actual good" means that a customer would repay a loan while "actual bad" means that the customer would default. The action associated with "predict bad" is to deny the loan. Hence, the cashflow relative to any baseline associated with this prediction is the same regardless

of whether "actual good" or "actual bad" is true. In every economically reasonable cost matrix for this domain, both entries in the "predict bad" row must be the same. If these entries are different, it is because different baselines have been chosen for each entry.

Costs or benefits can be measured against any baseline, but the baseline must be fixed. An opportunity cost is a foregone benefit, i.e. a missed opportunity rather than an actual penalty. It is easy to make the mistake of measuring different opportunity costs against different baselines. For example, the erroneous cost matrix above can be justified informally as follows: "The cost of approving a good customer is zero, and the cost of rejecting a bad customer is zero, because in both cases the correct decision has been made. If a good customer is rejected, the cost is an opportunity cost, the foregone profit of 1. If a bad customer is approved for a loan, the cost is the lost loan principal of 5."

To see concretely that the reasoning in quotes above is incorrect, suppose that the bank has one customer of each of the four types. Clearly the cost matrix above is intended to imply that the net change in the assets of the bank is then -4. Alternatively, suppose that we have four customers who receive loans and repay them. The net change in assets is then +4. Regardless of the baseline, any method of accounting should give a difference of 8 between these scenarios. But with the erroneous cost matrix above, the first scenario gives a total cost of 6, while the second scenario gives a total cost of 0.

In general the amount in some cells of a cost or benefit matrix may not be constant, and may be different for different examples. For example, consider the credit card transactions domain. Here the benefit matrix might be

	fraudulent	legitimate
refuse	\$20	$-$\$20
approve	$-x$	$0.02x$

where x is the size of the transaction in dollars. Approving a fraudulent transaction costs the amount of the transaction because the bank is liable for the expenses of fraud. Refusing a legitimate transaction has a non-trivial cost because it annoys a customer. Refusing a fraudulent transaction has a non-trivial benefit because it may prevent further fraud and lead to the arrest of a criminal.

9.4 Making optimal decisions

Given known costs for correct and incorrect predictions, a test example should be predicted to have the class that leads to the lowest expected cost. This expectation is

the predicted average cost, and is computed using the conditional probability of each class given the example.

In the two-class case, the optimal prediction is class 1 if and only if the expected cost of this prediction is less than or equal to the expected cost of predicting class 0, i.e. if and only if

$$p(y=0|x)c_{10} + p(y=1|x)c_{11} \leq p(y=0|x)c_{00} + p(y=1|x)c_{01}$$

which is equivalent to

$$(1-p)c_{10} + pc_{11} \leq (1-p)c_{00} + pc_{01}$$

given $p = p(y=1|x)$. If this inequality is in fact an equality, then predicting either class is optimal.

The threshold for making optimal decisions is p^* such that

$$(1-p^*)c_{10} + p^*c_{11} = (1-p^*)c_{00} + p^*c_{01}.$$

Assuming the reasonableness conditions $c_{10} > c_{00}$ and $c_{01} \geq c_{11}$, the optimal prediction is class 1 if and only if $p \geq p^*$. Rearranging the equation for p^* leads to

$$c_{00} - c_{10} = -p^*c_{10} + p^*c_{11} + p^*c_{00} - p^*c_{01}$$

which has the solution

$$p^* = \frac{c_{10} - c_{00}}{c_{10} - c_{00} + c_{01} - c_{11}}$$

assuming the denominator is nonzero, which is implied by the reasonableness conditions. This formula for p^* shows that any 2x2 cost matrix has essentially only one degree of freedom from a decision-making perspective, although it has two degrees of freedom from a matrix perspective. The cause of the apparent contradiction is that the optimal decision-making policy is a nonlinear function of the cost matrix.

Note that in some domains costs have more impact than probabilities on decision-making. An extreme case is that with some cost matrices the optimal decision is always the same, regardless of what the various outcome probabilities are for a given example x. In general, decisions and predictions may be not in 1:1 correspondence. Multiple different predicted labels may lead to the same decision, while different levels of certainty concerning the same label may lead to different decisions. Indeed, identical predictions concerning the label may lead to different decisions for different examples, if the costs or estimated costs for those examples differ.[1]

[1] The situation where the optimal decision is fixed is well-known in philosophy as Pascal's wager. Essentially, this is the case where the minimax decision and the minimum expected cost decision are

9.5 Limitations of cost-based analysis

The analysis of decision-making above assumes that the objective is to minimize expected cost. This objective is reasonable when a game is played repeatedly, and the actual class of examples is set stochastically.

The analysis may not be appropriate when the agent is risk-averse and can make only a few decisions.

Also not appropriate when the actual class of examples is set by a non-random adversary.

Costs may need to be estimated via learning.

We may need to make repeated decisions over time about the same example.

Decisions about one example may influence other examples.

When there are more than two classes, we do not get full label information on training examples.

9.6 Evaluating the success of decision-making

In order to evaluate whether a real-world campaign is proceeding as expected, we need to measure the profit achieved by the campaign, and then to compute whether or not this observed profit is consistent with what was anticipated. What was anticipated is formalized as a confidence interval for the achieved profit. A confidence interval is a range $[\alpha, \beta]$ such that there is a specified probability that the observed value lies within the range. It is common to consider 95% confidence intervals.

In general, let x be an example, and let $f(x)$ be some function of x. A test set is a collection of examples x_i. We want to predict the observed sum

$$s = \sum_{i=1}^{n} f(x_i)$$

where n is the number of examples in the test set. There is randomness in this sum, because there is randomness in which specific examples x_i occur in the test set. Given

always the same. An argument analogous to Pascal's wager provides a justification for Occam's razor in machine learning. Essentially, a PAC theorem says that if we have a fixed, limited number of training examples, and the true concept is simple, then if we pick a simple concept then we will be right. However if we pick a complex concept, there is no guarantee we will be right.

	nature simple	nature complex
predict simple	succeed	fail
predict complex	fail	fail

Here "simple" means that the hypothesis is drawn from a space with low cardinality, while complex means it is drawn from a space with high cardinality.

the randomness, we also want to estimate a confidence interval for the observed sum. Alternatively, we might want to estimate the average of all observed $f(x)$ values, and a confidence interval for the average. Whether the average or the sum is estimated, the size n of the test set is a critical parameter. Intuitively, the bigger n is, the smaller the overall effect of randomness.

For each x, $f(x)$ is a different outcome of a random variable. Let the name of this random variable be F. We want to predict the sum of F outcomes that will be observed. Let this sum be the random variable S. Suppose that we know the expectation $m = E[F]$ and the variance $s^2 = Var[F]$. The expectation of S is then

$$E[S] = \sum_{i=1}^{n} m = nm$$

and the variance of S is

$$Var[S] = \sum_{i=1}^{n} s^2 = ns^2.$$

The first equation above is based on the fact that the expectation of a sum is the sum of the expectations, which is always true. The second equation is based on the fact that the variance of a sum is the sum of the variances. This fact is true only if the random variables being added are independent. Here, we assume that each example x is probabilistically independent of all the others.

The definition of variance is $Var[Z] = E[(Z - E[Z])^2]$, but it is often easier to compute variances using the alternative form

$$\begin{aligned} Var[Z] &= E[Z^2 - 2ZE[Z] + E[Z]^2] \\ &= E[Z^2] - 2E[Z]E[Z] + E[Z]^2 \\ &= E[Z^2] - E[Z]^2. \end{aligned}$$

The definition of standard deviation is $std[Z] = \sqrt{Var[Z]}$. Applying the definition to S above gives $std[S] = \sqrt{n}s$. If the distribution of S is approximately Gaussian, then a 95% confidence interval for it is approximately $[nm - 2\sqrt{n}s, nm + 2\sqrt{n}s]$.

In general, the distribution of F is not Gaussian in the slightest. However, the central limit theorem says that for any large collection of independent random variables, their sum (and also their mean) tends towards following a Gaussian distribution, if each of the individual random variables has finite mean and variance. Here, if the examples x in the test set are independent, then their values $f(x)$ are independent random variable outcomes. Each of these values has a finite mean m and a finite variance s^2, so the central limit theorem applies.

The following is an example of applying the results above. Let $F = Y \cdot A - c$ where Y is a random 0/1 label, A is a random real value, and c is a constant. For

concreteness, assume that $E[Y] = 0.05$ as is true approximately for the KDD 98 task. Similarly, assume that $E[A] = 15$ and $Var[A] = 200$. The value of Y is binary, so $Y = Y^2$ always. Hence $E[Y^2] = E[Y]$ and $Var[Y] = E[Y^2] - E[Y]^2 = 0.05 - 0.0025 = 0.0475$.

In general, if the random variables A and B are independent, then $E[AB] = E[A]E[B]$. Assuming that Y and A are independent here, we get $E[F] = 0.75 - c$. From $E[A^2] = E[A]^2 + Var[A] = 425$, we get $E[Y^2A^2] = 0.05 \cdot 425 = 21.25$. Hence $Var[F] = 21.25 - (0.75 - c)^2$. If c is close to 0.75, then the term $(0.75 - c)^2$ is negligible, and $std[F] = \sqrt{21.25} = 4.6$ approximately.

Suppose that S is the sum of 100,000 outcomes of F, and $c = 0.68$. Clearly $E[S] = 7000$. The standard deviation of S is approximately $4.6\sqrt{100,000} = 1450$. The 95% confidence interval is then around $[4100, 9900]$.

The method above can be applied to obtain a confidence interval for the total profit achieved after making decisions using probabilistic models of Y and A, but there are additional complications. First, Y and A are empirically not independent: the outcome $Y = 1$ is correlated with a smaller outcome of A. Second, the decision-making process should be modeled: whether or not an example x_i is included in the test set depends on the estimated values $E[Y|x_i]$ and $E[A|x_i]$. The complications do not change the basic process of creating a confidence interval, and the standard deviation of F remains about the same. However, the expectation $E[F]$ is much higher, since the point of decision-making is to increase this value.

9.7 Rules of thumb for evaluating data mining campaigns

This section is preliminary, incomplete, and contains original research. It is not required for CSE 255.

Let q be a fraction of the test set. There is a remarkable rule of thumb that the lift attainable at q is around $\sqrt{1/q}$. For example, if $q = 0.25$ then the attainable lift is around $\sqrt{4} = 2$.

The rule of thumb is not valid for very small values of q, for two reasons. The first reason is mathematical: an upper bound on the lift is $1/t$, where t is the overall fraction of positives; t is also called the target rate, response rate, or base rate. The second reason is empirical: lifts observed in practice tend to be well below 10. Hence, the rule of thumb can reasonably be applied when $q > 0.02$ and $q > t^2$.

The fraction of all positive examples that belong to the top-ranked fraction q of the test set is $q\sqrt{1/q} = \sqrt{q}$. The lift in the bottom $1 - q$ is $(1 - \sqrt{q})/(1 - q)$. We have

$$\lim_{q \to 1} \frac{1 - \sqrt{q}}{1 - q} = 0.5.$$

This says that the examples ranked lowest are positive with probability equal to $t/2$. In other words, the lift for the lowest ranked examples cannot be driven below 0.5.

Let c be the cost of a contact and let b be the benefit of a positive. This means that the benefit matrix is

	actual negative	actual positive
decide negative	0	0
decide positive	$-c$	$b-c$

Let n be the size of the test set. The profit at q is the total benefit minus the total cost, which is

$$nqbt\sqrt{1/q} - nqc = nc(bt\sqrt{q}/c - q) = nc(k\sqrt{q} - q)$$

where $k = tb/c$. Profit is maximized when

$$0 = \frac{d}{dq}(kq^{0.5} - q) = k(0.5)q^{-0.5} - 1.$$

The solution is $q = (k/2)^2$.

This solution is in the range zero to one only when $k \leq 2$. When $tb/c > 2$ then maximum profit is achieved by soliciting every prospect, so data mining is pointless. The maximum profit that can be achieved is

$$nc(k(k/2) - (k/2)^2) = nck^2/4 = ncq.$$

Remarkably, this is always the same as the cost of running the campaign, which is nqc.

As k decreases, the attainable profit decreases fast, i.e. quadratically. If $k < 0.4$ then $q < 0.04$ and the attainable profit is less than $0.04nc$. Such a campaign may have high risk, because the lift attainable at small q is typically worse than suggested by the rule of thumb. Note however that a small adverse change in c, b, or t is not likely to make the campaign lose money, because the expected revenue is always twice the campaign cost.

It is interesting to consider whether data mining is beneficial or not from the point of view of society at large. Remember that c is the cost of one contact from the perspective of the initiator of the campaign. Each contact also has a cost or benefit for the recipient. Generally, if the recipient responds, one can assume that the contact was beneficial for him or her, because s/he only responds if the response is beneficial. However, if the recipient does not respond, which is the majority case, then one can assume that the contact caused a net cost to him or her, for example a loss of time.

From the point of view of the initiator of a campaign a cost or benefit for a respondent is an externality. It is not rational for the initiator to take into account

these benefits or costs, unless they cause respondents to take actions that affect the initiator, such as closing accounts. However, it *is* rational for society to take into account these externalities.

The conclusion above is that the revenue from a campaign, for its initiator, is roughly twice its cost. Suppose that the benefit of responding for a respondent is λb where b is the benefit to the initiator. Suppose also that the cost of a solicitation to a person is μc where c is the cost to the initiator. The net benefit to respondents is positive as long as $\mu < 2\lambda$.

The reasoning above clarifies why spam email campaigns are harmful to society. For these campaigns, the cost c to the initiator is tiny. However, the cost to a recipient is not tiny, so μ is large. Whatever λ is, it is likely that $\mu > 2\lambda$.

In summary, data mining is only beneficial in a narrow sweet spot, where

$$tb/2 \leq c \leq \alpha tb$$

where α is some constant greater than 1. The product tb is the average benefit of soliciting a random customer. If the cost c of solicitation is less than half of this, then it is rational to contact all potential respondents. If c is much greater than the average benefit, then the campaign is likely to have high risk for the initiator.

As an example of the reasoning above, consider the scenario of the 1998 KDD contest. Here $t = 0.05$ about, $c = \$0.68$, and the average benefit is $b = \$15$ approximately. We have $k = tb/c = 75/68 = 1.10$. The rule of thumb predicts that the optimal fraction of people to solicit is $q = 0.30$, while the achievable profit per person $cq = \$0.21$. In fact, the methods that perform best in this domain achieve profit of about $\$0.16$, while soliciting about 70% of all people.

Exercises

1.

Suppose that your lab wants to crystallize a protein. You can try experimental conditions x that differ on temperature, salinity, etc. The label $y = 1$ means crystallization is successful, while $y = 0$ means failure. Assume (not realistically) that the results of different experiments are independent. You have a classifier that predicts $p(y = 1|x)$, the probability of success of experiment x. The cost of doing one experiment is \$60. The value of successful crystallization is \$9000.

(a) Write down the benefit matrix involved in deciding rationally whether or not to perform a particular experiment. *The benefit matrix is*

	success	*failure*
do experiment	*9000-60*	*-60*
don't	*0*	*0*

(b) What is the threshold probability of success needed in order to perform an experiment? *It is rational to do an experiment under conditions x if and only if the expected benefit is positive, that is if and only if*

$$(9000 - 60) \cdot p + (-60) \cdot (1 - p) > 0.$$

where $p = p(\textit{success}|x)$. The threshold value of p is $60/9000 = 1/150$.

(c) Is it rational for your lab to take into account a budget constraint such as "we only have a technician available to do one experiment per day"? *No. A budget constraint is an alternative rule for making decisions that is less rational than maximizing expected benefit. The optimal behavior is to do all experiments that have success probability over 1/150.*

Additional explanation: The \$60 cost of doing an experiment should include the expense of technician time. If many experiments are worth doing, then more technicians should be hired. If the budget constraint is unavoidable, then experiments should be done starting with those that have highest probability of success. If just one successful crystallization is enough, then experiments should also be done in order of declining success probability, until the first actual success.

2.

Suppose that you work for a bank that wants to prevent criminal laundering of money. The label $y = 1$ means that a money transfer is criminal, while $y = 0$ means the

transfer is legal. You have a classifier that estimates the probability $p(y = 1|x)$ where x is a vector of feature values describing a money transfer.

Let z be the dollar amount of the transfer. The matrix of costs (negative) and benefits (positive) involved in deciding whether or not to deny a transfer is as follows:

	criminal	legal
deny	0	$-0.10z$
allow	$-z$	$0.01z$

Work out the rational policy based on $p(y = 1|x)$ for deciding whether or not to allow a transfer.

3.

Suppose that you are looking for a rare plant. You can search geographical locations x that differ on rainfall, urbanization, etc. The label $y = 1$ means the plant is found, while $y = 0$ means failure. Assume (not realistically) that the results of different searches are independent.

You have a classifier that predicts $p(y = 1|x)$, the probability of success when searching at location x. The cost of doing one search is \$200. The value of finding the plant is \$7000. For every two unsuccessful searches, on average, the information gained is expected to save you one future search.

(a) Write down the benefit matrix involved in deciding rationally whether or not to perform a particular search. *The benefit matrix is*

	success	*failure*
do search	*7000-200*	*-200 + 200/2*
don't	*0*	*0*

(b) What is the threshold probability of success needed in order to perform a search? *It is rational to search at location x if and only if the expected benefit is positive, that is if and only if*

$$(6800) \cdot p + (-100) \cdot (1 - p) > 0.$$

where $p = p(\textit{success}|x)$. The threshold value of p is $100/6900 = 1/69$.

(c) Suppose that you are constrained to do only one search per day. Where should you search each day?

CSE 255 assignment due on May 7, 2013

As before, you should work in a team of two students, and you are free to change partners or not. This assignment is the last one to use the KDD98 data. Again, you should measure final success just once on the hold-out instances in the file `cup98val.zip`. Remember that the hold-out labels in `valtargt.txt` are sorted by `CONTROLN`, unlike the hold-out instances.

Remember that each example corresponds to a person who may or may not make a donation. We assume that if the person is not asked, she or he will not donate. The goal is to ask an optimal subset of people. The measure of success to maximize is total donations received minus \$0.68 for every solicitation. Train a regression function to predict donation amounts, and a classifier to predict donation probabilities. For a person x, let the predicted donation amount be $a(x)$ and let the predicted donation probability be $p(x)$. It is rational to send a donation request to person x if and only if

$$p(x) \cdot a(x) \geq 0.68.$$

This rule is the decision-making method that you apply to hold-out examples. Total profit is a sum over all hold-out examples. If a person is selected according to the rule, and she or he actually does make a donation, then total profit increases by the donation amount minus 68 cents. If a person is selected, but does not donate, then 68 cents is subtracted from total profit.

Many classifiers, including SVMs, give scores that are not correct probabilities. Logistic regression does give approximately well-calibrated probabilities. SVMs and other methods can be used, but you must convert their outputs into reasonable probabilities in some way.

Before you apply your final decision-making method to the hold-out set (just once!), estimate a confidence interval for the profit you expect to achieve on the hold-out set. This should be a range in dollars $[a, b]$ such that the probability is (say) 95% that the observed profit will lie in this range. State your estimated confidence interval clearly in your report, and explain the reasoning behind it. Notice that the maximum single donation is \$500, so total profit varies by \$499.32 depending just on whether this one person is solicited or not. The confidence interval that you obtain will be quite wide.

Intuitively, there are two sources of uncertainty for each person: whether s/he will donate, and if so, how much. Work out the logic of how to quantify these uncertainties, and then the logic of combining the uncertainties into an overall confidence interval. The central issue here is not any mathematical details such as Student distributions versus Gaussians. It is thinking about where uncertainty arises, and how it is propagated.

Chapter 10

Learning in nonstandard labeling scenarios

This chapter discusses how to learn predictive models from nonstandard training data. Specifically, we consider three different but related scenarios where labels are unknown for some training examples.

10.1 The standard scenario for learning a classifier

In the standard situation, the training and test sets are two different random samples from the same population. Being from the same population means that both sets follow a common probability distribution. Let x be a training or test instance, and let y be its label. The common distribution is $p(x, y)$.[1] We can always write

$$p(x, y) = p(x)p(y|x)$$

and also

$$p(x, y) = p(y)p(x|y)$$

without loss of generality and without making any assumptions. These equations are sometimes called the chain rule of probability. They are true both when the probability values are probability masses, and when they are probability densities.

[1] Remember that the notation $p(x, y)$ is an abbreviation for $p(X = x \text{ and } Y = y)$ where X and Y are random variables, and x and y are possible values of these variables. If X and Y are both discrete-valued random variables, then this distribution is a probability mass function (pmf). Otherwise, it is a probability density function (pdf). For a classification task, Y is discrete. For a regression task, Y is real-valued.

10.2 Sample selection bias in general

Suppose that training examples (x, y) are random samples from the population as before, but the label y is sometimes missing. In other words, y is observed for some instances x, but not for others. The instances x for which y is known are in general not a random sample from the population. An important question is whether the unlabeled training examples can be exploited in some way. However, here we focus on a different question: how can we adjust for the fact that the labeled training examples are not representative of the test examples?

We can describe the scenario formally as follows. Let s be a new random variable with value $s = 1$ if and only if x is selected. This means that y is observed if and only if $s = 1$. We can identify three possibilities for s. The easiest case is when $p(s = 1|x, y)$ is a constant. In this case, the labeled training examples are a fully random subset. We have fewer training examples, but otherwise we are in the usual situation. This case is called "missing completely at random" (MCAR).

A more difficult situation arises if the value of s is not completely random, i.e., it is associated in some way with the value of x and/or of y. This case where $p(s = 1) \neq p(s = 1|x, y)$ has two subcases. First, suppose that s depends on x but, given x, not on y. In other words, when x is fixed then s is independent of y. This means that $p(s = 1|x, y) = p(s = 1|x)$ for all x. In this case, Bayes' rule gives that

$$p(y|x, s = 1) = \frac{p(s = 1|x, y)p(y|x)}{p(s = 1|x)} = \frac{p(s = 1|x)p(y|x)}{p(s = 1|x)} = p(y|x)$$

assuming that $p(s = 1|x) > 0$ for all x. Therefore we can learn a correct model of $p(y|x)$ from just the labeled training data, without using the unlabeled data in any way.

This case $p(s = 1|x, y) = p(s = 1|x)$ is rather misleadingly called "missing at random" (MAR). It is not the case that labels are missing in a totally random way, because missingness does depend on x. It is also not the case that s and y are independent. However, it is true that s and y are conditionally independent, conditional on x. Concretely, for each value of x the equation $p(y|x, s = 0) = p(y|x) = p(y|x, s = 1)$ holds.

The assumption that $p(s = 1|x) > 0$ is important. The real-world meaning of this assumption is that label information must be available with non-zero probability for every possible instance x. Otherwise, it might be the case for some x that no labeled training examples are available from which to estimate $p(y|x, s = 1)$.

Suppose that even when x is known, there is still some correlation between s and y. In this case the label y is said to be "missing not at random" (MNAR) and inference is much more difficult. We do not discuss this case further here.

10.3 Importance weighting

Suppose that we have MAR labeled data and we want to learn something more general than $p(y|x)$. Specifically, let the goal be to estimate $E[f(x,y)]$ where $x, y \sim p(x,y)$ for any function f, using data drawn from the distribution $p(x,y|s=1)$.

Generalizing, we want to estimate $E[f(z)]$ where z follows the distribution $p(z)$, but we can only draw samples of z from the distribution $q(z)$. A useful fact is that

$$E[f(z); z \sim p(z)] = E[f(z)\frac{p(z)}{q(z)}; z \sim q(z)]$$

assuming $q(z) > 0$ for all z. More generally the requirement is that $q(z) > 0$ whenever $p(z) > 0$, assuming the definition $0/0 = 0$. The equation above is called the importance weighting identity.

Write $E[f(x,y); x, y \sim p(x,y)]$ as $E[f(x,y)]$ and write $E[f(x,y); x, y \sim p(x,y|s=1)]$ as $E[f(x,y)|s=1]$. We have

$$\begin{aligned} E[f(x,y)] &= E[f(x,y)\frac{p(x,y)}{p(x,y|s=1)}|s=1] \\ &= E[f(x,y)\frac{p(x)p(y|x)}{p(x|s=1)p(y|x,s=1)}|s=1] \\ &= E[f(x,y)\frac{p(x)}{p(x|s=1)}|s=1] \end{aligned}$$

since $p(y|x) = p(y|x, s=1)$ using the MAR assumption. Applying Bayes' rule to $p(x|s=1)$ gives

$$\begin{aligned} E[f(x,y)] &= E[f(x,y)\frac{p(x)}{p(s=1|x)p(x)/p(s=1)}|s=1] \\ &= E[f(x,y)\frac{p(s=1)}{p(s=1|x)}|s=1]. \end{aligned}$$

The constant $p(s=1)$ can be estimated as r/n where r is the number of labeled training examples and n is the total number of training examples. Let $\hat{p}(s=1|x)$ be a trained model of the conditional probability $p(s=1|x)$. The estimate of $E[f(x,y)]$ is then

$$\frac{1}{r}\sum_{i=1}^{r} f(x_i,y_i)\frac{r/n}{\hat{p}(s=1|x_i)} = \frac{1}{n}\sum_{i=1}^{r}\frac{f(x_i,y_i)}{\hat{p}(s=1|x_i)}$$

where the sums range over the r examples $\langle x, y\rangle$ with known labels y. This estimate is called a "plug-in" estimate because it is based on plugging the observed values of

$\langle x, y \rangle$ into a formula that would be correct if based on integrating over all values of $\langle x, y \rangle$.

More generally, the result above on estimating $E[f(x, y)]$ can be used to modify most learning algorithms in order to make them work with biased training data. Each training example is given an individual weight equal to $p(s = 1)/p(s = 1|x)$. If only the relative weights are important, then they can be just $1/p(s = 1|x)$. Intuitively, the example x has a high weight if its probability $p(s = 1|x)$ of being selected is small because it must "stand in" for many examples that were not selected.

The weighting approach just explained is correct in the statistical sense that it is unbiased, if the estimated probabilities $\hat{p}(s = 1|x)$ are correct.[2] However, this approach typically has high variance since a few labeled examples with small values of $\hat{p}(s = 1|x)$ have dominant influence. A simple heuristic to reduce variance is to place a ceiling on the values $1/\hat{p}(s = 1|x)$. For example the ceiling 1000 is used by [Huang et al., 2006]. However, no good method for selecting the ceiling value is known. An alternative approach is to use regularization when training the model $\hat{p}(s = 1|x)$, which will tend to make values of $\hat{p}(s = 1|x)$ be close to the base rate $p(s = 1)$.

10.4 Covariate shift

Imagine that the available training data come from one hospital, but we want to learn a classifier to use at a different hospital. Let x be a patient and let y be a label such as "has bird flu." The distribution $p(x)$ of patient characteristics is different at the two hospitals, but we are willing to assume that the relationship to be learned, which is $p(y|x)$, is unchanged.

In a scenario like the one above, labeled training examples from all classes are available, but the distribution of instances is different for the training and test sets. This scenario is called "covariate shift" where covariate means feature, and shift means change of distribution. The features x have changed distribution, but the two conditional distributions of the label y given x are the same.

Following the argument in Section 10.2, the assumption that $p(y|x)$ is unchanged means that $p(y|x, s = 1) = p(y|x, s = 0)$ where $s = 1$ refers to the first hospital, where labels are known, and $s = 0$ refers to the second hospital. The simple approach to covariate shift is thus to train the classifier on the data $(x, y, s = 1)$ in the usual way, and to apply it to patients from the second hospital directly.

This simple approach is correct mathematically only if the model used for $p(y|x)$ is correctly specified and enough training examples are available. In practice it is

[2] Note that the meaning of "biased" here is different from the meaning of the same word in the previous paragraph.

often better to weight the available training examples in such a way that those that are typical for the case $s = 0$ have more influence. Adapting the argument of Section 10.3, each available training example (x, y) with known label y should be assigned the weight $p(s = 0|x)/p(s = 1|x)$. This says that the more x is typical of examples from the second hospital, the higher its weight should be.

The labels $s = 0$ and $s = 1$ are exhaustive and mutually exclusive, so $p(s = 0|x) = 1 - p(s = 1|x)$. A predictive model of $p(s = 1|x)$, that is a probabilistic classifier for s, can be obtained by using all patients from the first hospital as instances of the class $s = 1$, and all those from the second hospital as instances of the class $s = 0$.

10.5 Reject inference

Suppose that we want to build a model that predicts who will repay a loan. We need to be able to apply the model to the entire population of future applicants (sometimes called the "through the door" population). The usable training examples are people who were actually granted a loan in the past. Unfortunately, these people are not representative of the population of all future applicants. They were previously selected precisely because some decision-maker believed that they were more likely to repay. The problem of "reject inference" is to learn somehow from previous applicants who were not approved, for whom we hence do not have training labels.

This scenario is similar to the covariate shift scenario, but with the difference that unlabeled training instances are available. In the reject inference scenario, but not in the covariate shift scenario, x is known even when the label y is missing. The central similarity is the shared assumption that missingness depends only on the value of x, and not additionally on the value of y. In both scenarios, whether or not the label is missing is expected to be correlated with the value of the label, but this correlation disappears after conditioning on x.

Similar conclusions apply in the covariate shift and reject inference scenarios. A well-specified model of $p(y|x)$ learned from those examples for which y is known is applicable to all future examples. However, it may be better to assign a weight to each training example that is higher if it is typical of examples for which y is not known.

If missingness does depend on y, even after conditioning on x, then we are in the MNAR situation and in general we are can draw no firm conclusions. An example of this situation is "survivor bias." Suppose that our analysis is based on historical records, and those records are more likely to exist ($s = 1$) for survivors ($y = 1$), everything else being equal. Then $p(s = 1|y = 1, x) > p(s = 1|y = 0, x)$ where "everything else being equal" means that x is the same on both sides.

The MNAR situation can also arise because of unrecorded information, Suppose that (i) some loan applicants are interviewed, (ii) those who do well in the interview are more likely to be approved, (iii) the interview is predictive of a good outcome, and (iv) the interview is not recorded in the available database. Let $v = 1$ mean that the person x did well in the interview. Then $p(y = 1|s = 1, x) > p(y = 1|s = 0, x)$ because $p(y = 1|v = 1, x) > p(y = 1|v = 0, x)$ and $p(s = 1|v = 1, x) > p(s = 1|v = 0, x)$.

10.6 Outcomes of treatment

The remainder of this chapter is preliminary, and not required for CSE 255.

A scenario that is related to reject inference, but different, is the task of learning to predict the benefit of a treatment. In medicine, a treatment may be a procedure provided to a patient, while in business, it may be an action such as sending a coupon to a customer.

Let $y = 1$ mean that the outcome is a success, and let $t = 1$ mean that the treatment was provided. Suppose that the treatment in fact has no benefit: $p(y = 1|t = 1, x) = p(y = 1|t = 0, x)$. Nevertheless, if doctors have historically given the treatment only to patients who were particularly likely to benefit, then the observed success rate $p(y = 1|t = 1)$ of the treatment is likely to be higher than its success rate $p(y = 1)$ on "through the door" patients. Conversely, if doctors have historically used the treatment only on patients who were especially ill, then $p(y = 1)$ may be higher than $p(y = 1|t = 1)$.

What is relevant for decision-making is individualized prognoses $p(y = 1|t = 1, x)$ and $p(y = 1|t = 0, x)$, not group averages such as $p(y = 1)$ or $p(y = 1|t = 1)$.

Treatment is fundamentally different from selection because we expect that in general $p(y = 1|t = 1, x) \neq p(y = 1|t = 0, x)$ whereas the the MAR assumption says that $p(y = 1|s = 1, x) = p(y = 1|s = 0, x)$.

10.7 Positive and unlabeled examples

Suppose that only positive examples are labeled. This fact can be stated formally as the equation

$$p(s = 1|x, y = 0) = 0. \tag{10.1}$$

Without some assumption about which positive examples are labeled, it is impossible to make progress. A common assumption is that the labeled positive examples are chosen completely randomly from all positive examples. Let this be called the

"selected at random" assumption. Stated formally, it is that

$$p(s = 1|x, y = 1) = p(s = 1|y = 1) = c. \tag{10.2}$$

Another way of stating the assumption is that s and x are conditionally independent given y.

A training set consists of two subsets, called the labeled ($s = 1$) and unlabeled ($s = 0$) sets. Suppose we provide these two sets as inputs to a standard training algorithm. This algorithm will yield a function $g(x)$ such that $g(x) = p(s = 1|x)$ approximately. The following lemma shows how to obtain a model of $p(y = 1|x)$ from $g(x)$.

Lemma 1. Suppose the "selected completely at random" assumption holds. Then $p(y = 1|x) = p(s = 1|x)/c$ where $c = p(s = 1|y = 1)$.

Proof. Remember that the assumption is $p(s = 1|y = 1, x) = p(s = 1|y = 1)$. Now consider $p(s = 1|x)$. We have that

$$\begin{aligned} p(s = 1|x) &= p(y = 1 \wedge s = 1|x) \\ &= p(y = 1|x)p(s = 1|y = 1, x) \\ &= p(y = 1|x)p(s = 1|y = 1). \end{aligned}$$

The result follows by dividing each side by $p(s = 1|y = 1)$. ■

Several consequences of the lemma are worth noting. First, f is an increasing function of g. This means that if the classifier f is only used to rank examples x according to the chance that they belong to class $y = 1$, then the classifier g can be used directly instead of f.

Second, $f = g/p(s = 1|y = 1)$ is a well-defined probability $f \leq 1$ only if $g \leq p(s = 1|y = 1)$. What this says is that $g > p(s = 1|y = 1)$ is impossible. This is reasonable because the labeled (positive) and unlabeled (negative) training sets for g are samples from overlapping regions in x space. Hence it is impossible for any example x to belong to the positive class for g with a high degree of certainty.

The value of the constant $c = p(s = 1|y = 1)$ can be estimated using a trained classifier g and a validation set of examples. Let V be such a validation set that is drawn from the overall distribution $p(x, y, s)$ in the same manner as the nontraditional training set. Let P be the subset of examples in V that are labeled (and hence positive). The estimator of $p(s = 1|y = 1)$ is the average value of $g(x)$ for x in P. Formally the estimator is $e_1 = \frac{1}{n} \sum_{x \in P} g(x)$ where n is the cardinality of P.

We shall show that $e_1 = p(s = 1|y = 1) = c$ if it is the case that $g(x) = p(s = 1|x)$ for all x. To do this, all we need to show is that $g(x) = c$ for $x \in P$. We can

show this as follows:

$$\begin{aligned} g(x) &= p(s=1|x) \\ &= p(s=1|x,y=1)p(y=1|x) \\ &\quad + p(s=1|x,y=0)p(y=0|x) \\ &= p(s=1|x,y=1)\cdot 1 + 0\cdot 0 \ \text{ since } \ x \in P \\ &= p(s=1|y=1). \end{aligned}$$

Note that in principle any single example from P is sufficient to determine c, but that in practice averaging over all members of P is preferable.

There is an alternative way of using Lemma 1. Let the goal be to estimate $E_{p(x,y,s)}[h(x,y)]$ for any function h, where $p(x,y,s)$ is the overall distribution. To make notation more concise, write this as $E[h]$. We want an estimator of $E[h]$ based on a positive-only training set of examples of the form $\langle x, s\rangle$.

Clearly $p(y=1|x,s=1)=1$. Less obviously,

$$\begin{aligned} p(y=1|x,s=0) &= \frac{p(s=0|x,y=1)p(y=1|x)}{p(s=0|x)} \\ &= \frac{[1-p(s=1|x,y=1)]p(y=1|x)}{1-p(s=1|x)} \\ &= \frac{(1-c)p(y=1|x)}{1-p(s=1|x)} \\ &= \frac{(1-c)p(s=1|x)/c}{1-p(s=1|x)} \\ &= \frac{1-c}{c}\frac{p(s=1|x)}{1-p(s=1|x)}. \end{aligned}$$

By definition

$$\begin{aligned} E[h] &= \int_{x,y,s} h(x,y)p(x,y,s) \\ &= \int_x p(x)\sum_{s=0}^{1} p(s|x)\sum_{y=0}^{1} p(y|x,s)h(x,y) \\ &= \int_x p(x)\Big(p(s=1|x)h(x,1) \\ &\qquad + p(s=0|x)[p(y=1|x,s=0)h(x,1) \\ &\qquad\qquad + p(y=0|x,s=0)h(x,0)]\Big). \end{aligned}$$

The plug-in estimate of $E[h]$ is then the empirical average

$$\frac{1}{m}\Big(\sum_{\langle x,s=1\rangle} h(x,1) + \sum_{\langle x,s=0\rangle} w(x)h(x,1) + (1-w(x))h(x,0)\Big)$$

where

$$w(x) = p(y=1|x,s=0) = \frac{1-c}{c}\frac{p(s=1|x)}{1-p(s=1|x)} \tag{10.3}$$

and m is the cardinality of the training set. What this says is that each labeled example is treated as a positive example with unit weight, while each unlabeled example is treated as a combination of a positive example with weight $p(y=1|x,s=0)$ and a negative example with complementary weight $1-p(y=1|x,s=0)$. The probability $p(s=1|x)$ is estimated as $g(x)$ where g is the nontraditional classifier explained in the previous section.

The result above on estimating $E[h]$ can be used to modify a learning algorithm in order to make it work with positive and unlabeled training data. One method is to give training examples individual weights. Positive examples are given unit weight and unlabeled examples are duplicated; one copy of each unlabeled example is made positive with weight $p(y=1|x,s=0)$ and the other copy is made negative with weight $1-p(y=1|x,s=0)$.

10.8 Further issues

Spam filtering scenario.

Observational studies.

Causation.

Uplift.

Concept drift.

Moral hazard.

Adverse selection.

Exercises

1.

(a) Consider the importance weighting approach to deal with unlabeled examples. What are the minimum and maximum possible values of the weights? *Let x be a labeled example, and let its weight be $p(s = 1)/p(s = 1|x)$. Intuitively, this weight is how many copies are needed to allow the one labeled example to represent all the unlabeled examples that are similar. The conditional probability $p(s = 1|x)$ can range between 0 and 1, so the weight can range between $p(s = 1)$ and infinity.*

(b) Consider the weighting approach to learn from positive and unlabeled examples. What are the minimum and maximum possible values of the weights? *In this scenario, weights are assigned to unlabeled examples, not to labeled examples as above. The weights here are probabilities $p(y = 1|x, s = 0)$, so they range between 0 and 1.*

(c) Explain intuitively what can go wrong if the "selected at random" assumption is false, when learning from positive and unlabeled examples. *The "selected at random" assumption says that the positive examples with known labels are perfectly representative of the positive examples with unknown labels. If this assumption is false, then there will be unlabeled examples that in fact are positive, but that we treat as negative, because they are different from the labeled positive examples. The trained model of the positive class will be too narrow.*

2.

The importance weighting identity is the equation

$$E[f(z); z \sim p(z)] = E[f(z)w(z); z \sim q(z)]$$

where

$$w(z) = \frac{p(z)}{q(z)}.$$

We assume that $q(z) > 0$ for all z such that $p(z) > 0$, and we define $0/0 = 0$.

Suppose that the training set consists of values z sampled according to the probability distribution $q(z)$. Explain intuitively which members of the training set will have greatest influence on the estimate of $E[f(z); z \sim p(z)]$.

CSE 255 assignment due on May 14, 2013

Important: The printed version of this assignment that was distributed in class on May 7 has been replaced by the version here.

The goal of this assignment is to train a useful classifier in each of two different non-standard scenarios: (i) covariate shift and (ii) reject inference.

The file at `cseweb.ucsd.edu/users/elkan/255/dmfiles.zip` contains one shared test set and a training set for each scenario. Training set (i) has 5,164 examples, set (ii) has 11,305 examples, and the test set has 11,307 examples. Each example describes one person. The label to be predicted is whether or not the person earns over $50,000 per year. This is an interesting label to predict because it is analogous to a label describing whether or not the person is a customer that is desirable in some way. Each dataset has 13 features and is derived from the Adult dataset that is available at `archive.ics.uci.edu/ml/datasets/Adult`. (Many thanks to Aditya Menon for creating the datasets. There is a third training set that is not used in this assignment. Also, we are not using the published weightings, so the `fnlwgt` feature has been omitted.)

Training set (i) is a case of covariate shift because it does not follow the same distribution as the population of test examples. Training set (ii) requires reject inference because the training examples are a random sample from the test population, but the true label is known only for some of these examples. The persons with known labels, on average, have higher income than the ones with unknown labels.

First, do cross-validation on the test set to establish the accuracy that is achievable when all training set labels are known. In the report, show a learning curve, that is a figure with accuracy on the vertical axis as a function of the number of training examples on the horizontal axis. Try training subsets of size 1000, 2000, etc. Next, for each of the two scenarios, compare learning $p(y|x)$ with and without reweighting. Include in the report a learning curve for each approach. Discuss the extent to which each scenario reduces achievable accuracy.

In order to do reweighting, you need to train a model of $p(s = 1|x)$ for each scenario. Use this model to try to understand the selection process for each training set. Because these sets are artificial, the selection process for each was in fact quite simple.

Sample assignment

This assignment is to participate in the PAKDD 2010 data mining contest. Details of the contest are at `http://sede.neurotech.com.br/PAKDD2010/`. The contest requires reject inference as explained in this chapter.

Your first goal should be to understand the data and submission formats, and to submit a correct set of predictions. Make sure that when you have good predictions later, you will not run into any technical difficulties.

Your second goal should be to understand the contest scenario and the differences between the training and two test datasets. Do some exploratory analysis of the three datasets. In your written report, explain your understanding of the scenario, and your general findings about the datasets.

Next, based on your general understanding, design a sensible approach for achieving the contest objective. Implement this approach and submit predictions to the contest. Of course, you may refine your approach iteratively and you may make multiple submissions.

The contest rules ask each team to submit a paper of four pages.

> The manuscript should be in the scientific paper format of the conference detailing the stages of the KDD process, focusing on the aspects which make the solution innovative. The manuscript should be the basis for writing up a full paper for an eventual post-conference proceeding (currently under negotiation). The authors of top solutions and selected manuscripts with innovative approaches will have the opportunity to submit to that forum. The quality of the manuscripts will not be used for the competition assessment, unless in the case of ties.

You can find template files for LaTeX and Word at `http://www.springer.com/computer/lncs?SGWID=0-164-6-793341-0`. Do not worry about formatting details.

Chapter 11

Recommender systems

A recommender system is any software that suggests items to users. For example, the users may be people who rent movies, and the items may be movies. The technical core of many recommender systems is a collaborative filtering (CF) method. The CF task is to predict the rating that a user will give to an item, based on ratings for different items provided by the same user and on ratings provided by other users. The general assumption underlying most CF methods is that users who give similar ratings to some items are likely also to give similar ratings to other items.

From a formal perspective, the input to a collaborative filtering algorithm is an incomplete matrix of ratings. Each row corresponds to a user, while each column corresponds to an item. If user $1 \leq i \leq m$ has rated item $1 \leq j \leq n$, the matrix entry x_{ij} is the value of this rating. Often rating values are integers between 1 and 5. If a user has not rated an item, the corresponding matrix entry is missing. Missing ratings are often represented as 0, but this should not be viewed as an actual rating value.

The output of a collaborative filtering algorithm is a prediction of the value of each missing matrix entry. Typically the predictions are not required to be integers. Given these predictions, many different real-world tasks can be performed. For example, a recommender system might suggest to a user those items for which the predicted ratings by this user are highest.

There are two main general approaches to the formal collaborative filtering task: nearest-neighbor-based and model-based. Given a user and item for which a prediction is wanted, nearest neighbor (NN) approaches use a similarity function between rows, and/or a similarity function between columns, to pick a subset of relevant other users or items. The prediction is then some sort of average of the known ratings in this subset.

Model-based approaches to collaborative filtering construct a low-complexity

representation of the complete x_{ij} matrix. This representation is then used instead of the original matrix to make predictions. Typically each prediction can be computed in $O(1)$ time using only a fixed number of coefficients from the representation.

The most popular model-based collaborative filtering algorithms are based on standard matrix approximation methods such as the singular value decomposition (SVD), principal component analysis (PCA), or nonnegative matrix factorization (NNMF). Of course these methods are themselves related to each other.

From the matrix decomposition perspective, the fundamental issue in collaborative filtering is that the matrix given for training is incomplete. Many algorithms have been proposed to extend SVD, PCA, NNMF, and related methods to apply to incomplete matrices, often based on expectation-maximization (EM). Unfortunately, methods based on EM are intractably slow for large collaborative filtering tasks, because they require many iterations, each of which computes a matrix decomposition on a full m by n matrix. Here, we discuss an efficient approach to decomposing large incomplete matrices.

11.1 Applications of matrix approximation

In addition to collaborative filtering, many other applications of matrix approximation are important also. Examples include the voting records of politicians, the results of sports matches, and distances in computer networks. The largest research area with a goal similar to collaborative filtering is so-called "item response theory" (IRT), a subfield of psychometrics. The aim of IRT is to develop models of how a person answers a multiple-choice question on a test. Users and items in collaborative filtering correspond to test-takers and test questions in IRT. Missing ratings are called omitted responses in IRT.

One conceptual difference between collaborative filtering research and IRT research is that the aim in IRT is often to find a single parameter for each individual and a single parameter for each question that together predict answers as well as possible. The idea is that each individual's parameter value is then a good measure of intrinsic ability and each question's parameter value is a good measure of difficulty. In contrast, in collaborative filtering research the goal is often to find multiple parameters describing each person and each item, because the assumption is that each item has multiple relevant aspects and each person has separate preferences for each of these aspects.

11.2 Measures of performance

Given a set of predicted ratings and a matching set of true ratings, the difference between the two sets can be measured in alternative ways. The two standard measures are mean absolute error (MAE) and mean squared error (MSE). Given a probability distribution over possible values, MAE is minimized by taking the median of the distribution while MSE is minimized by taking the mean of the distribution. In general the mean and the median are different, so in general predictions that minimize MAE are different from predictions that minimize MSE.

Most matrix approximation algorithms aim to minimize MSE between the training data (a complete or incomplete matrix) and the approximation obtained from the learned low-complexity representation. Some methods can be used with equal ease to minimize MSE or MAE.

11.3 Additive models

Let us consider first models where each matrix entry is represented as the sum of two contributions, one from its row and one from its column. Formally, $a_{ij} = r_i + c_j$ where a_{ij} is the approximation of x_{ij} and r_i and c_j are scalars to be learned.

Define the training mean $\bar{x}_{i\cdot}$ of each row to be the mean of all its known values; define the the training mean of each column $\bar{x}_{\cdot j}$ similarly. The user-mean model sets $r_i = \bar{x}_{i\cdot}$ and $c_j = 0$ while the item-mean model sets $r_i = 0$ and $c_j = \bar{x}_{\cdot j}$. A slightly more sophisticated baseline is the "bimean" model: $r_i = 0.5\bar{x}_{i\cdot}$ and $c_j = 0.5\bar{x}_{\cdot j}$.

The "bias from mean" model has $r_i = \bar{x}_{i\cdot}$ and $c_j = (x_{ij} - r_i)_{\cdot j}$. Note that in the latter expression, the subscript $\cdot j$ means the average over all rows, for column number j. Intuitively, c_j is the average amount by which users who provide a rating for item j like this item more or less than the average item that they have rated.

The optimal additive model can be computed quite straightforwardly. Let I be the set of matrix indices for which x_{ij} is known. The MSE optimal additive model is the one that minimizes

$$\sum_{\langle i,j\rangle \in I} (r_i + c_j - x_{ij})^2.$$

This optimization problem is a special case of a sparse least-squares linear regression problem: find z that minimizes $\|Az - b\|_2$ where the column vector $z = \langle r_1, \ldots, r_m, c_1, \ldots, c_n\rangle'$, b is a column vector of x_{ij} values and the corresponding row of A is all zero except for ones in positions i and $m + j$. z can be computed by many methods, including the Moore-Penrose pseudoinverse of A: $z = (A'A)^{-1}A'b$. However this approach requires inverting an $m + n$ by $m + n$ matrix, which is computationally expensive. For linear regression in general, approaches based on QR

factorization are more efficient. For this particular special case of linear regression, Section 11.4 below gives a gradient-descent method to obtain the optimal z that only requires $O(|I|)$ time and space.

The matrix A is always rank-deficient, with rank at most $m+n-1$, because any constant can be added to all r_i and subtracted from all c_j without changing the matrix reconstruction. If some users or items have very few ratings, the rank of A may be less than $m+n-1$. However the rank deficiency of A does not cause computational problems in practice.

11.4 Multiplicative models

Multiplicative models are similar to additive models, but the row and column values are multiplied instead of added. Specifically, x_{ij} is approximated as $a_{ij} = r_i c_j$ with r_i and c_j being scalars to be learned. Like additive models, multiplicative models have one unnecessary degree of freedom, because any single value r_i or c_j can be fixed without changing the space of achievable approximations.

A multiplicative model is a rank-one matrix decomposition, since the rows and columns of the y matrix are linearly proportional. We now present a general algorithm for learning row value/column value models; additive and multiplicative models are special cases. Let x_{ij} be a matrix entry and let r_i and c_j be corresponding row and column values. We approximate x_{ij} by $a_{ij} = f(r_i, c_j)$ for some fixed function f. The approximation error is defined to be $e(f(r_i, c_j), x_{ij})$. The training error E over all known matrix entries I is the pointwise sum of errors.

To learn r_i and c_j by minimizing E by gradient descent, we evaluate

$$\begin{aligned}\frac{\partial}{\partial r_i}E &= \sum_{\langle i,j\rangle\in I}\frac{\partial}{\partial r_i}e(f(r_i,c_j),x_{ij}) \\ &= \sum_{\langle i,j\rangle\in I}\frac{\partial}{\partial f(r_i,c_j)}e(f(r_i,c_j),x_{ij})\frac{\partial}{\partial r_i}f(r_i,c_j).\end{aligned}$$

As before, I is the set of matrix indices for which x_{ij} is known. Consider the special case where $e(u,v) = |u-v|^p$ for $p > 0$. This case is a generalization of MAE and MSE: $p = 2$ corresponds to MSE and $p = 1$ corresponds to MAE. We have

$$\begin{aligned}\frac{\partial}{\partial u}e(u,v) &= p|u-v|^{p-1}\frac{\partial}{\partial u}|u-v| \\ &= p|u-v|^{p-1}\,\mathrm{sgn}(u-v).\end{aligned}$$

Here $\mathrm{sgn}(a)$ is the sign of a, that is -1, 0, or 1 if a is negative, zero, or positive. For computational purposes we can assume the derivative is zero for the non-

differentiable case $u = v$. Therefore

$$\frac{\partial}{\partial r_i} E = \sum_{\langle i,j \rangle \in I} p|f(r_i, c_j) - x_{ij}|^{p-1} \operatorname{sgn}(f(r_i, c_j) - x_{ij}) \frac{\partial}{\partial r_i} f(r_i, c_j).$$

Now suppose $f(r_i, c_j) = r_i + c_j$ so $\frac{\partial}{\partial r_i} f(r_i, c_j) = 1$. We obtain

$$\frac{\partial}{\partial r_i} E = p \sum_{\langle i,j \rangle \in I} |r_i + c_j - x_{ij}|^{p-1} \operatorname{sgn}(r_i + c_j - x_{ij}).$$

Alternatively, suppose $f(r_i, c_j) = r_i c_j$ so $\frac{\partial}{\partial r_i} f(r_i, c_j) = c_j$. We get

$$\frac{\partial}{\partial r_i} E = p \sum_{\langle i,j \rangle \in I} |r_i c_j - x_{ij}|^{p-1} \operatorname{sgn}(r_i c_j - x_{ij}) c_j.$$

Given the gradients above, we apply online (stochastic) gradient descent. This means that for each triple $\langle r_i, c_j, x_{ij} \rangle$ in the training set, we compute the gradient with respect to r_i and c_j based just on this one example, and perform the updates

$$r_i := r_i - \lambda \frac{\partial}{\partial r_i} e(f(r_i, c_j), x_{ij})$$

and

$$c_j := c_j - \lambda \frac{\partial}{\partial c_j} e(f(r_i, c_j), x_{ij})$$

where λ is a learning rate. An epoch is one pass over the whole training set, performing the updates above once for each $\langle i, j \rangle \in I$.

After any finite number of epochs, stochastic gradient descent does not converge fully. The learning rate typically follows a decreasing schedule such as $\lambda = a/(b+e)$ where e is the number of the current epoch. The values a and b are often chosen to be small positive integers. The maximum number of epochs is another parameter that needs to be chosen.

Gradient descent as described above directly optimizes precisely the same objective function (given the MSE error function) that is called "incomplete data likelihood" in EM approaches to factorizing matrices with missing entries. It is sometimes forgotten that EM is just one approach to solving maximum-likelihood problems; incomplete matrix factorization is an example of a maximum-likelihood problem where an alternative solution method is superior.

11.5 Combining models by fitting residuals

Simple models can be combined into more complex models. In general, a second model is trained to fit the difference between the predictions made by the first model and the truth as specified by the training data; this difference is called the residual.

The most straightforward way to combine models is additive. Let a_{ij} be the prediction from the first model and let x_{ij} be the corresponding training value. The second model is trained to minimize error relative to the residual $x_{ij} - a_{ij}$. Let b_{ij} be the prediction made by the second model for matrix entry ij. The combined prediction is then $a_{ij} + b_{ij}$. If the first and second models are both multiplicative, then the combined model is a rank-two approximation. The extension to higher-rank approximations is obvious.

The common interpretation of a combined model is that each component model represents an aspect or property of items and users. The idea is that the column value for each item is the intensity with which it possesses this property, and the row value for each user is the weighting the user places on this property. However this point of view implicitly treats each component model as equally important. It is more accurate to view each component model as an adjustment to previous models, where each model is empirically less important than each previous one, because the numerical magnitude of its contribution is smaller. The aspect of items represented by each model cannot be understood in isolation, but only in the context of previous aspects.

11.6 Regularization

A matrix factorization model of rank k has k parameters for each person and for each movie. For example, consider a training set of 500,000 ratings for 10,000 viewers and 1000 movies. A rank-50 unregularized factor model is almost certain to overfit the training data, because it has $50 \cdot (10,000 + 1000) = 550,000$ parameters, which is more than the number of data points for training.

We can typically improve generalization by using a large value for k while reducing the effective number of parameters via regularization.

The unregularized loss function is

$$E = \sum_{\langle i,j \rangle \in I} e(f(r_i, c_j), x_{ij})$$

where I is the set of ij pairs for which x_{ij} is known. The corresponding L_2-

regularized loss function is

$$E = \mu(\sum_i r_i^2 + \sum_j c_j^2) + \sum_{\langle i,j\rangle \in I} e(f(r_i, c_j), x_{ij})$$

where μ is the strength of regularization. Remember that this expression has only two parameter vectors r and c because it is just for a factorization of rank one. The partial derivative with respect to r_i is

$$\frac{\partial}{\partial r_i} E = 2\mu r_i + \sum_{\langle i,j\rangle \in I} \frac{\partial}{\partial f(r_i, c_j)} e(f(r_i, c_j), x_{ij}) \frac{\partial}{\partial r_i} f(r_i, c_j).$$

Assume that $e(u, v) = (u - v)^2$ and $f(r_i, c_j) = r_i c_j$, so $\frac{\partial}{\partial r_i} f(r_i, c_j) = c_j$. Then

$$\frac{\partial}{\partial r_i} E = 2\mu r_i + 2 \sum_{\langle i,j\rangle \in I} (r_i c_j - x_{ij}) c_j.$$

Using stochastic gradient descent, for each training example $\langle r_i, c_j, x_{ij}\rangle$ we do the updates

$$r_i := r_i - 2\lambda\mu r_i - 2\lambda(r_i c_j - x_{ij}) c_j$$

and similarly for c_j.

When doing regularization, it is reasonable not to regularize the vectors learned for the rank-one factorization. The reason is that these are always positive, if ratings x_{ij} are positive, because $x_{ij} = r_i c_j$ on average. (The vectors could always both be negative, but that would be unintuitive without being more expressive.)

After the rank-one factorization, residuals are zero on average, so learned vectors must have positive and negative entries, and regularization towards zero is reasonable.

11.7 Further issues

Another issue not discussed is any potential probabilistic interpretation of a matrix decomposition. One reason for not considering models that are explicitly probabilistic is that these are usually based on Gaussian assumptions that are incorrect by definition when the observed data are integers and/or in a fixed interval.

A third issue not discussed is any explicit model for how matrix entries come to be missing. There is no assumption or analysis concerning whether entries are missing "completely at random" (MCAR), "at random" (MAR), or "not at random" (MNAR). Intuitively, in the collaborative filtering context, missing ratings are likely to be MNAR. This means that even after accounting for other influences, whether or

not a rating is missing still depends on its value, because people select which items to rate based on how much they anticipate liking them. Concretely, everything else being equal, low ratings are still more likely to be missing than high ratings. This intuition can be confirmed empirically. On the Movielens dataset, the average rating in the training set is 3.58. The average prediction from one reasonable method for ratings in the test set, i.e. ratings that are unknown but known to exist, is 3.79. The average prediction for ratings that in fact do not exist is 3.34.

Amazon: people who looked at x bought y eventually (within 24hrs usually)

Exercises

1.

For each part below, say whether the statement in italics is true or false, and then explain your answer briefly.

(a) For any model, the average predicted rating for unrated movies is expected to be less than the average actual rating for rated movies. *True. People are more likely to like movies that they have actually watched, than random movies that they have not watched. Any good model should capture this fact. In more technical language, the value of a rating is correlated with whether or not it is missing.*

(b) Let the predicted value of rating x_{ij} be $r_i c_j + s_i d_j$ and suppose r_i and c_j are trained first. For all viewers i, r_i should be positive, but for some i, s_i should be negative. *True. Ratings x_{ij} are positive, and $x_{ij} = r_i c_j$ on average, so r_i and c_j should always be positive. (They could always both be negative, but that would be unintuitive without being more expressive.)*

The term $s_i d_j$ models the difference $x_{ij} - r_i c_j$. This difference is on average zero, so it is sometimes positive and sometimes negative. In order to allow $s_i d_j$ to be negative, s_i must be negative sometimes. (Making s_i be always positive, while allowing d_j to be negative, might be possible. However in this case the expressiveness of the model $s_i d_j$ would be reduced.)

(c) We have a training set of 500,000 ratings for 10,000 viewers and 1000 movies, and we train a rank-50 unregularized factor model. This model is likely to overfit the training data. *True. The unregularized model has $50 \cdot (10,000 + 1000) = 550,000$ parameters, which is more than the number of data points for training. Hence, overfitting is practically certain.*

2.

The lecture notes above say

> Given the gradients above, we apply online (stochastic) gradient descent. This means that we iterate over each triple $\langle r_i, c_j, x_{ij} \rangle$ in the training set, compute the gradient with respect to r_i and c_j based just on this one example, and perform the updates

$$r_i := r_i - \lambda \frac{\partial}{\partial r_i} e(f(r_i, c_j), x_{ij})$$

and

$$c_j := c_j - \lambda \frac{\partial}{\partial c_j} e(f(r_i, c_j), x_{ij})$$

where λ is a learning rate.

State and explain what the first update rule is for the special case $e(u, v) = (u - v)^2$ and $f(r_i, c_j) = r_i \cdot c_j$.

3.

Section 11.6 of the lecture notes says "Using stochastic gradient descent, for each training example $\langle r_i, c_j, x_{ij} \rangle$ we do the updates

$$r_i := r_i - 2\lambda\mu r_i - 2\lambda(r_i c_j - x_{ij})c_j$$

and similarly for c_j." In the righthand side of the update rule for r_i, explain the role of λ, of μ, and of $r_i c_j$.

CSE 255 assignment due on May 21, 2013

The goal of this project is to implement a collaborative filtering algorithm for the task of predicting movie ratings. Use the small MovieLens dataset available at `http://www.grouplens.org/node/73`, which has 100,000 ratings given by 943 users to 1682 movies. Implement a matrix factorization method that allows for missing data and has regularization. A simple approach that can be implemented in just a few lines of code is alternating least squares.

For experiments, use the five cross-validation folds provided by the publishers of the data, as described in the README file. Note that cross-validation makes the task easier than it is in reality, because there are (almost certainly) no cold-start users or movies. Moreover, evaluation is biased towards users who have provided more ratings, for whom it is presumably easier to make accurate predictions.

In the report, show mean absolute error and mean squared error graphically, as a function of the rank of the matrix factorization. Also in the report, make a careful list of all the design choices that you have to make. You will not have time to investigate all of these experimentally, but do discuss them briefly. Some of the design issues are whether to subtract user means and/or item means before doing factorization, how to do regularization, how to initialize the optimization process, and whether to train dimensions sequentially or simultaneously. Investigate at least one design issue experimentally.

Investigate the final trained matrix factorization model, and the data. Which movies have the most similar learned latent vectors? Can any of the learned dimensions be interpreted? Are any of the dimensions correlated with the explicit information available about users and movies? Are any of the movies or users outliers? If so, in what way are they outliers?

In the report, explain the time complexity of the method implemented. In a figure, show timing information collected experimentally. What is the most binding computational bottleneck of the implementation? Discuss whether the method could be applied easily to the full Netflix dataset of about 10^8 ratings.

Chapter 12

Text mining

This chapter explains how to do data mining on datasets that are collections of documents. Text mining tasks include

- classifier learning
- clustering,
- topic modeling, and
- latent semantic analysis.

Classifiers for documents are useful for many applications. Major uses for binary classifiers include spam detection and personalization of streams of news articles. Multiclass classifiers are useful for routing messages to relevant recipients in a large organization.

Most classifiers for documents are designed to categorize according to subject matter. However, it is also possible to learn to categorize according to qualitative criteria such as helpfulness for product reviews submitted by consumers, or importance for email messages. It is also possible to use a classifier to identify the author of a document, or to recognize some characteristic of the author such as gender or age. However, authorship can rarely be determined with a degree of certainty that is sufficient, for example, to be used as reliable evidence in a criminal trial.

Classifiers can be even more useful for ranking documents than for dividing them into categories. With a training set of very helpful product reviews, and another training set of very unhelpful reviews, we can learn a scoring function that sorts other reviews according to their degree of helpfulness. There is often no need to pick a threshold, which would be arbitrary, to separate marginally helpful from marginally unhelpful reviews. The Google email feature called Priority Inbox uses a classifier to rank messages according to their predicted importance as perceived by the recipient.

As discussed in Chapter 7, when one class of examples is rare, it is not reasonable to use 0/1 accuracy to measure the success of a classifier. In text mining, it is especially common to use what is called F-measure, which is the harmonic mean of precision and recall:

$$f = \frac{2}{1/p + 1/r}$$

where p and r are precision and recall for the rare class, as explained in Chapter 7 above.

The mistakes made by a binary classifier are either false positives or false negatives. A false positive is a failure of precision, while a false negative is a failure of recall. In domains such as medical diagnosis, a failure of recall is often considered more serious than a failure of precision. The reason is that the latter has mainly a monetary cost, but the former can have a significant health consequence. However, in applications of document classification, failures of recall are often less serious than failures of precision. The central reason is that if the content of a document is genuinely important, many documents are likely to contain different versions of the same content. So a failure of recall on one of these documents will not lead to an overall failure. On the other hand, each document that is classified positive is examined by a human, so each failure of precision has a significant cost.

Figuring out the optimal trade off between failures of recall and precision, given that a failure of recall has a certain probability of being remedied later, is a problem in decision theory that is an extension of the methods explained in Section 9.3.

In many applications of multiclass classification, a single document can belong to more than one category, so it is correct to predict more than one label. This task is specifically called multilabel classification. In standard multiclass classification, the classes are mutually exclusive, i.e. a special type of negative correlation is fixed in advance. In multilabel classification, it is important to learn the positive and negative correlations between classes.

Text mining is often used as a proxy for data mining on underlying entities represented by text documents. For example, employees may be represented by their resumes, images by their captions, songs by their lyrics, and movies by their subtitles. Music, movies, and products may also be represented by reviews, which may be written by consumers or by critics. The correlations between labels may arise from correlations in the underlying entities. For example, the labels “slow” and “romantic” may be positively correlated for songs.

12.1 The bag-of-words representation

The first question to is how to represent documents. For genuine understanding of natural language the order of the words in documents must be preserved. Computational linguistics is a large and successful research area focused on analyzing the syntax and semantics of sentences and documents. However, methods from computational linguistics tend to require extensive external resources such as dictionaries, and have difficulty scaling to large collections of documents. For many large-scale data mining tasks, including classifying and clustering documents, it is sufficient to use a superficial representation that loses all information about word order.

Given a collection of documents, the first task to perform is to identify the set of all words used a least once in at least one document. This set is called the vocabulary V. Often, it is reduced in size by keeping only words that are used in at least two documents. (Words that are found only once are often mis-spellings or other mistakes.) Although the vocabulary is a set, we fix an arbitrary ordering for it, so that we can refer to word 1 through word m where $m = |V|$ is the size of the vocabulary.

Once V has been fixed, each document is represented as a vector of length m with integer entries. If this vector is x then its jth component x_j is the number of appearances of word j in the document. The length of the document is $n = \sum_{j=1}^{m} x_j$. For typical documents, n is much smaller than m and $x_j = 0$ for most words j.

Many applications of text mining also eliminate from the vocabulary so-called "stop" words. These are words that are common in most documents and do not correspond to any particular subject matter. In linguistics these words are sometimes called "function" words. They include pronouns (you, he, it) connectives (and, because, however), prepositions (to, of, before), auxiliaries (have, been, can), and generic nouns (amount, part, nothing). It is important to appreciate, however, that generic words carry a lot of information for many tasks, including identifying the author of a document or detecting its genre. Also, within any specific domain, there may be words that are not function words in general, but that behave as such in the domain, such as "network" and "algorithm" in machine learning.

A collection of documents is represented as a two-dimensional matrix where each row describes a document and each column corresponds to a word. Each entry in this matrix is an integer count; most entries are zero. It makes sense to view each column as a feature. It also makes sense to learn a low-rank approximation of the whole matrix; doing this is called latent semantic analysis (LSA) and is discussed in Section 13.3 below.

12.2 The multinomial distribution

Once we have a representation for individual documents, the natural next step is to select a model for a set of documents. This model is a probability distribution. Given a training set of documents, we will choose values for the parameters of the distribution that make the training documents have high probability.

Then, given a test document, we can evaluate its probability according to the model. The higher this probability is, the more similar the test document is to the training set.

The probability distribution that we use is the multinomial. Mathematically, this distribution is

$$p(x;\theta) = \frac{n!}{\prod_{j=1}^{m} x_j!} \prod_{j=1}^{m} \theta_j^{x_j}.$$

where the data x are a vector of non-negative integers and the parameters θ are a real-valued vector. Both vectors have the same length m. The components of θ are non-negative and have unit sum: $\sum_{j=1}^{m} \theta_j = 1$.

Intuitively, θ_j is the probability of word j while x_j is the count of word j. Each time word j appears in the document it contributes an amount θ_j to the total probability, hence the term θ_j to the power x_j. The fraction before the product is called the multinomial coefficient. It is the number of different word sequences that might have given rise to the observed vector of counts x. Note that the multinomial coefficient is a function of x, but not of θ. Therefore, when comparing the probability of a document according to different models, that is according to different parameter vectors θ, we can ignore the multinomial coefficient.

Like any discrete distribution, a multinomial has to sum to one, where the sum is over all possible data points. Here, a data point is a document containing n words. The number of such documents is exponential in their length n: it is m^n. The probability of any individual document will therefore be very small. What is important is the relative probability of different documents. A document that mostly uses words with high probability will have higher relative probability.

At first sight, computing the probability of a document requires $O(m)$ time because of the product over j. However, if $x_j = 0$ then $\theta_j^{x_j} = 1$ so the jth factor can be omitted from the product. Similarly, $0! = 1$ so the jth factor can be omitted from $\prod_{j=1}^{m} x_j!$. Hence, computing the probability of a document needs only $O(n)$ time.

Because the probabilities of individual documents decline exponentially with length n, it is necessary to do numerical computations with log probabilities:

$$\log p(x;\theta) = \log n! - [\sum_{j=1}^{m} \log x_j!] + [\sum_{j=1}^{m} x_j \cdot \log \theta_j]$$

Given a set of training documents, the maximum-likelihood estimate of the jth parameter is

$$\theta_j = \frac{1}{T} \sum_x x_j$$

where the sum is over all documents x belonging to the training set. The normalizing constant is $T = \sum_x \sum_j x_j$ which is the sum of the sizes of all training documents.

If a multinomial has $\theta_j = 0$ for some j, then every document with $x_j > 0$ for this j has zero probability, regardless of any other words in the document. Probabilities that are perfectly zero are undesirable, so we want $\theta_j > 0$ for all j. Smoothing with a constant c is the way to achieve this. We set

$$\theta_j \propto c + \sum_x x_j$$

where the symbol $\propto$ means "proportional to." The constant c is called a pseudocount. Intuitively, it is a notional number of appearances of word j that are assumed to exist, regardless of the true number of appearances. Typically c is chosen in the range $0 < c \leq 1$, often $c = 0.1$. Because the equality $\sum_j \theta_j = 1$ must be preserved, the normalizing constant must be $T' = mc + T$ in

$$\theta_j = \frac{1}{T'}(c + \sum_x x_j).$$

In order to avoid big changes in the estimated probabilities θ_j, one should have $c < T/m$. Remember that T is the sum of the sizes of all documents, and m is the number of words in the vocabulary.

Technically, one multinomial is a distribution over all documents of a fixed size n. Therefore, what is learned by the maximum-likelihood process just described is in fact a different distribution for each size n. These distributions, although separate, have the same parameter values.

12.3 Training Bayesian classifiers

Bayesian learning is an approach to learning classifiers based on Bayes' rule. Let x be an example and let y be its class. Suppose the alternative class values are 1 to K. We can write

$$p(y = k|x) = \frac{p(x|y = k)p(y = k)}{p(x)}.$$

In order to use the expression on the right for classification, we need to learn three items from training data: $p(x|y = k)$, $p(y = k)$, and $p(x)$. We can estimate $p(y = k)$

easily as $n_k / \sum_{k=1}^{K} n_k$ where n_k is the number of training examples with class label k. The denominator $p(x)$ can be computed as the sum of numerators

$$p(x) = \sum_{k=1}^{K} p(x|y=k)p(y=k).$$

In general, the class-conditional distribution $p(x|y = k)$ can be any distribution whose parameters are estimated using the training examples that have class label k.

Note that the training process for each class uses only the examples from that class, and is separate from the training process for all other classes. Usually, each class is represented by one multinomial fitted by maximum-likelihood as described in Section 12.2. In the formula

$$\theta_j = \frac{1}{T'}(c + \sum_x x_j)$$

the sum is taken over the documents in one class. Unfortunately, the exact value of c can strongly influence classification accuracy.

12.4 Burstiness

The multinomial model says that each appearance of the same word j always has the same probability θ_j. In reality, additional appearances of the same word are less surprising, i.e. they have higher probability. Consider the following excerpt from a newspaper article, for example.

> Toyota Motor Corp. is expected to announce a major overhaul. Yoshi Inaba, a former senior Toyota executive, was formally asked by Toyota this week to oversee the U.S. business. Mr. Inaba is currently head of an international airport close to Toyota's headquarters in Japan.
>
> Toyota's U.S. operations now are suffering from plunging sales. Mr. Inaba was credited with laying the groundwork for Toyota's fast growth in the U.S. before he left the company.
>
> Recently, Toyota has had to idle U.S. assembly lines, pay workers who aren't producing vehicles and offer a limited number of voluntary buyouts. Toyota now employs 36,000 in the U.S.

The multinomial distribution arises from a process of sampling words with replacement. An alternative distribution named the the Dirichlet compound multinomial (DCM) arises from an urn process that captures the authorship process better. Consider a bucket with balls of $|V|$ different colors. After a ball is selected randomly,

it is not just replaced, but also one more ball of the same color is added. Each time a ball is drawn, the chance of drawing the same color again is increased. This increased probability models the phenomenon of burstiness.

Let the initial number of balls with color j be β_j. These initial values are the parameters of the DCM distribution. The DCM parameter vector β has length $|V|$, like the multinomial parameter vector, but the sum of the components of β is unconstrained. This one extra degree of freedom allows the DCM to discount multiple observations of the same word, in an adjustable way. The smaller the parameter vales β_j are, the more words are bursty.

12.5 Discriminative classification

There is a consensus that linear support vector machines (SVMs) are the best known method for learning classifiers for documents. The extra expressiveness of nonlinear SVMs is not needed, because the number of features is typically much larger than the number of training examples. For the same reason, choosing the strength of regularization appropriately, typically by cross-validation, is crucial.

When using a linear SVM for text classification, accuracy can be improved considerably by transforming the raw counts. Since counts do not follow Gaussian distributions, it is not sensible to make them have mean zero and variance one. Inspired by the discussion above of burstiness, it is sensible to replace each count x_j by $\log(1 + x_j)$. (The base of the logarithm is immaterial.) This transformation maps 0 to 0, so it preserves sparsity. A more extreme transformation that loses information is to make each count binary, that is to replace all non-zero values by one.

One can also transform counts in a supervised way, that is in a way that uses label information. This is most straightforward when there are just two classes. Experimentally, the following transformation is better than others:

$$x_j \mapsto I(x_j > 0)|\log tp/fn - \log fp/tn|.$$

Above, tp is the number of positive training examples containing word j, and fn is the number of these examples not containing the word, while fp is the number of negative training examples containing the word, and tn is the number of these examples not containing the word. If any of these numbers is zero, we replace it by 0.5, which of course is less than any of these numbers that is genuinely non-zero.

The transformation above is sometimes called logodds weighting. It ignores how often a word appears in an individual document. Instead, it makes each word count binary, and gives the word a weight that is the same for every document, but depends on how predictive the word is. The positive and negative classes are treated in a symmetric way. The value $|\log tp/fn - \log fp/tn|$ is large if tp/fn and fp/tn

have very different values. In this case the word is highly diagnostic for at least one of the two classes.

12.6 Clustering documents

Suppose that we have a collection of documents, and we want to find an organization for these, i.e. we want to do unsupervised learning. The simplest variety of unsupervised learning is clustering.

Clustering can be done probabilistically by fitting a mixture distribution to the given collection of documents. Formally, a mixture distribution is a probability density function of the form

$$p(x) = \sum_{k=1}^{K} \alpha_k p(x; \theta_k).$$

Here, K is the number of components in the mixture model. For each k, $p(x; \theta_k)$ is the distribution of component number k. The scalar α_k is the proportion of component number k.

Each component is a cluster.

12.7 Topic models

Mixture models, and clusterings in general, are based on the assumption that each data point is generated by a single component model. For documents, if components are topics, it is often more plausible to assume that each document can contain words from multiple topics. Topic models are probabilistic models that make this assumption.

Latent Dirichlet allocation (LDA) is the most widely used topic model. It is based on the intuition that each document contains words from multiple topics; the proportion of each topic in each document is different, but the topics themselves are the same for all documents.

The generative process assumed by the LDA model is as follows:

Given: Dirichlet distribution with parameter vector α of length K
Given: Dirichlet distribution with parameter vector β of length V
for topic number 1 to topic number K
    draw a multinomial with parameter vector ϕ_k according to β
for document number 1 to document number M
    draw a topic distribution, i.e. a multinomial θ according to α
    for each word in the document

draw a topic z according to θ
draw a word w according to ϕ_z

Note that z is an integer between 1 and K for each word.

The prior distributions α and β are assumed to be fixed and known, as are the number K of topics, the number M of documents, the length N_m of each document, and the cardinality V of the vocabulary (i.e. the dictionary of all words).

For learning, the training data are the words in all documents. Learning has two goals: (i) to infer the document-specific multinomial θ for each document, and (ii) to infer the topic distribution ϕ_k for each topic. After training, φ is a vector of word probabilities for each topic indicating the content of that topic. The distribution θ of each document is useful for classifying new documents, measuring similarity between documents, and more.

When applying LDA, it is not necessary to learn α and β. Steyvers and Griffiths recommend fixed uniform values $\alpha = 50/K$ and $\beta = .01$, where K is the number of topics.

12.8 Open questions

It would be interesting to work out an extension of the logodds mapping for the multiclass case. One suggestion is

$$x_j \mapsto I(x_j > 0) \cdot \max_k |\log p_{jk}/n_{jk}|$$

where i ranges over the classes, p_{jk} is the number of training examples in class k containing word j, and n_{jk} is the number of these examples not containing the word.

It is also not known if combining the log transformation and logodds weighting is beneficial, as in

$$x_j \mapsto \log(x_j + 1) \cdot |\log tp/fn - \log fp/tn|.$$

Exercises

1.

(a) Explain why, with a multinomial distribution, the probabilities of individual documents decline exponentially with length n.' *The probability of document x of length n according to a multinomial distribution is*

$$p(x;\theta) = \frac{n!}{\prod_{j=1}^{m} x_j!} \prod_{j=1}^{m} \theta_j^{x_j}.$$

Each θ_j value is less than 1. In total $n = \sum_j x_j$ of these values are multiplied together. Hence as n increases the product $\prod_{j=1}^{m} \theta_j^{x_j}$ decreases exponentially. Note the multinomial coefficient $n!/\prod_{j=1}^{m} x_j!$ does increase with n, but more slowly.

(b) Consider the multiclass Bayesian classifier

$$\hat{y} = \operatorname*{argmax}_k \frac{p(x|y=k)p(y=k)}{p(x)}.$$

Simplify the expression inside the argmax operator as much as possible, given that the model $p(x|y=k)$ for each class is a multinomial distribution. *The denominator $p(x)$ is the same for all k, so it does not influence which k is the argmax. Within the multinomial distributions $p(x|y=k)$, the multinomial coefficient does not depend on k, so it is constant for a single x and it can be eliminated also, giving*

$$\hat{y} = \operatorname*{argmax}_k \; p(y=k) \prod_{j=1}^{m} \theta_{kj}^{x_j}.$$

where θ_{kj} is the jth parameter of the kth multinomial.

(c) Consider the classifier from part (b) and suppose that there are just two classes. Simplify the classifier further into a linear classifier. *Let the two classes be $k=0$ and $k=1$ so we can write*

$$\hat{y} = 1 \;\; \textit{if and only if} \;\; p(y=1) \prod_{j=1}^{m} \theta_{1j}^{x_j} > p(y=0) \prod_{j=1}^{m} \theta_{0j}^{x_j}.$$

Taking logarithms and using indicator function notation gives

$$\hat{y} = I\left(\log p(y=1) + \sum_{j=1}^{m} x_j \log \theta_{1j} - \log p(y=0) - \sum_{j=1}^{m} x_j \log \theta_{0j}\right).$$

The classifier simplifies to

$$\hat{y} = I(c_0 + \sum_{j=1}^{m} x_j c_j)$$

where the coefficients are $c_0 = \log p(y = 1) - \log p(y = 0)$ *and* $c_j = \log \theta_{1j} - \log \theta_{0j}$. *The expression inside the indicator function is a linear function of* x.

2.

Consider the task of learning to classify documents into one of two classes, using the bag-of-words representation. Explain why a regularized linear SVM is expected to be more accurate than a Bayesian classifier using one maximum-likelihood multinomial for each class. Specifically, write one or two sentences for each of the following reasons: (a) handling words that are absent in one class in training; (b) learning the relative importance of different words.

3.

Consider the binary Bayesian classifier

$$\hat{y} = \underset{k \in \{0,1\}}{\operatorname{argmax}} \frac{p(x|y = k)p(y = k)}{p(x)}.$$

Simplify the expression inside the argmax operator as much as possible, given that the model $p(x|y = k)$ for each class is a multinomial distribution:

$$p(x|y = 0) = \frac{n!}{\prod_{j=1}^{m} x_j!} \prod_{j=1}^{m} \alpha_j^{x_j}$$

and

$$p(x|y = 1) = \frac{n!}{\prod_{j=1}^{m} x_j!} \prod_{j=1}^{m} \beta_j^{x_j}.$$

Also assume that $p(y = 0) = 0.5$. *The denominator* $p(x)$ *is the same for all* k, *so it does not influence which* k *is the argmax. Within the multinomial distributions, the multinomial coefficient does not depend on the parameters, so it is constant for a single* x *and it can be eliminated also, giving*

$$\hat{y} = \underset{k}{\operatorname{argmax}} \; p(y = k) \prod_{j=1}^{m} \theta_{kj}^{x_j}.$$

where θ_{kj} is either α_j or β_j.

The assumption $p(y = 0) = 0.5$ implies $p(y = 1) = 0.5$, so the $p(y = k)$ factor also does not influence classification. We can write

$$\hat{y} = 1 \text{ if and only if } \prod_{j=1}^{m} \beta_j^{x_j} > \prod_{j=1}^{m} \alpha_j^{x_j}.$$

Using indicator function notation gives

$$\hat{y} = I\Big(\prod_{j=1}^{m} \Big(\frac{\beta_j}{\alpha_j}\Big)^{x_j} > 1\Big).$$

Note that there is no guaranteed relationship between α_j and β_j. We know that $\sum_j \alpha_j = 1$ and similarly for the β parameter vector, but this knowledge does not help in making a further simplification.

CSE 255 assignment due on May 28, 2013

The goal of this assignment is to obtain a useful binary classifier for text documents. The dataset to use is the movie review polarity dataset, version 2.0, as published by Lillian Lee at `http://www.cs.cornell.edu/People/pabo/movie-review-data/`. Be sure to read the README carefully, and to understand the data and task fully.

You should train a discriminative classifier, either a logistic regression model or a support vector machine. Since there are more features than training examples, regularization is vital. To get good performance with an SVM, you need to use strong regularization. Investigate whether you can achieve better accuracy via feature selection, feature transformation, and/or feature weighting. Train classifiers with and without standard preprocessing involving stop words and stemming.

Evaluate the statistical significance of differences in accuracy that you find. Think carefully about any ways in which you may be allowing leakage of information from test subsets that makes estimates of accuracy be biased. Discuss this issue in your report. If you achieve very high accuracy, it is likely that you are a victim of leakage in some way.

Compare the accuracy that you can achieve with accuracies reported in some of the many published papers that use this dataset. You can find links to these papers at `http://www.cs.cornell.edu/People/pabo/movie-review-data/otherexperiments.html`. Discuss whether the comparison papers contain any methodological errors, in classifier training and/or in performance evaluation. Discuss also whether their results could be replicated, or if insufficient detail is provided for reproducibility. Verify that the comparison papers use exactly the same dataset version. Discuss the fairness and usefulness of any additional data preprocessing in these papers.

Analyze your best trained classifier to identify which features of movie reviews are most indicative of the review being favorable or unfavorable. Make a table of the top 20 words that are most indicative of being favorable, and a table of the top 20 words that are most indicative of being unfavorable. Discuss whether there are any surprises in these tables. Has the classifier learned anything that is not superficial? Other visualizations of trained classifiers are welcome also.

Chapter 13

Matrix factorization and applications

Many datasets are essentially two-dimensional matrices. For these datasets, methods based on ideas from linear algebra often provide considerable insight. It is remarkable that similar algorithms are useful for datasets as diverse as the following.

- In many datasets, each row is an example and each column is a feature. For example, in the KDD 1998 dataset each row describes a person. There are 22 features corresponding to different previous campaigns requesting donations. For each person and each campaign there is binary feature value, 1 if the person made a donation and 0 otherwise.
- In a text dataset, each row represents a document while each column represents a word.
- In a collaborative filtering dataset, each row can be a user and each column can be a movie.
- In a social network, each person corresponds to a row and also to a column. The network is a graph, and the matrix is its adjacency matrix.
- In a kernel matrix, each example corresponds to a row and also to a column. The number in row i and column j is the degree of similarity between examples i and j.

The cases above show that the matrix representing a dataset may be square or rectangular. Its entries may be binary, that is 0 or 1, or they may be real numbers. In some cases every entry has a known value, while in other cases some entries are unknown; missing is a synonym for unknown.

13.1 Singular value decomposition

A mathematical idea called the singular value decomposition (SVD) is at the root of most ways of analyzing datasets that are matrices. One important reason why the SVD is so important is that it is defined for *every* matrix. Other related concepts are defined only for matrices that are square, or only for matrices that are invertible. The SVD is defined for every matrix all of whose entries are known. Its extension to matrices with missing entries is discussed elsewhere.

Before explaining the SVD, let us review the concepts of linear dependence and rank. Consider a subset of rows of a matrix. Suppose that the rows in this subset are called a_1 to a_k. Let v be any row vector of length k. A linear combination of these rows is the new row vector that is the matrix product

$$v \begin{pmatrix} a_1 \\ a_2 \\ \cdots \\ a_k \end{pmatrix}$$

A set of rows is called linearly independent if no row is a linear combination of the others.

The rank of a matrix is the largest number of linearly independent rows that it has. If a matrix has rank k, then it has a subset of k rows such that (i) none of these is a linear combination of the others, but (ii) every row not in the subset is a linear combination of the rows in the subset. The rank of a matrix is a measure of its complexity. Consider for example a user/movie matrix. Suppose that its rank is k. This means that there are k users such that the ratings of every other user are a linear function of the ratings of these k users. Formally, these k users are a basis for the whole matrix.

Let A be any matrix, square or rectangular. Let A have m rows and n columns, and let r be the rank of A. The SVD is a product of three matrices named U, S, and V. Specifically,

$$A = USV^T$$

where V^T means the transpose of V, which can also be written V'. The matrix U is m by r, S is r by r, and V is n by r.

Some conditions on U, S, and V make the SVD unique. First, all the off-diagonal entries of S are zero, its diagonal entries σ_i are positive, and they are sorted from largest to smallest. These entries are called singular values.[1] Next, the columns of

[1] They are closely related to eigenvalues, but remember that the SVD is well-defined for rectangular matrices, while only square matrices have eigenvalues.

U, which are vectors of length m, are orthonormal. This means that each column is a unit vector, and the columns are perpendicular to each other. Concretely, let u_i and u_j be any two columns of U where $i \neq j$. We have $u_i \cdot u_i = 1$ and $u_i \cdot u_j = 0$. Similarly, the columns of V, which are vectors of length n, are orthonormal to each other.

Because U and V have orthonormal columns, the matrix product in the SVD can be simplified to

$$A = \sum_{k=1}^{r} \sigma_k u_k v_k^T.$$

Here, u_k is the kth column of U, and v_k^T is the transpose of the kth column of V. The product $u_k v_k^T$ is an m by n matrix of rank 1. It is often called the outer product of the vectors u_k and v_k. This simplification decomposes A into the sum of r matrices of rank 1.

For data mining, the SVD is most interesting because of the following property. Let U_k be the k leftmost columns of U. This means that U_k is an m by k matrix whose columns are the same as the first k columns of U. Similarly, let V_k be the k leftmost columns of V. Let S_k be the k by k diagonal matrix whose diagonal entries are S_{11} to S_{kk}. Define

$$A_k = U_k S_k V_k^T.$$

The rank of A_k is k. The following theorem was published by the mathematicians Carl Eckhart and Gale Young in 1936.

Theorem:

$$A_k = \min_{X:rank(X)\leq k} ||A - X||_2.$$

The theorem says that among all matrices of rank k or less, A_k has the smallest squared error with respect to A. The squared error is simply

$$||A - X||_2 = \sum_{i=1}^{m} \sum_{j=1}^{n} (A_{ij} - X_{ij})^2$$

and is sometimes called the Frobenius norm of the matrix $A - X$. The SVD essentially answers the question, what is the best low-complexity approximation of a given matrix A?

The matrices U_k, S_k, and V_k are called truncated versions of U, S, and V. The omitted columns in the truncated matrices correspond to the smallest diagonal entries in S. The hope is that these correspond to noise in the original observed A.

Notice that the matrices U, S, and V are themselves truncated in our definition to have r columns, where r is the rank of A. This definition is sometimes called the compact SVD. The omitted columns correspond to singular values $\sigma_i = 0$ for $i > r$.

The truncated SVD is called a matrix approximation:

$$A \approx U_k S_k V_k^T.$$

The matrix U_k has k columns, and the same number of rows as A. Each row of U_k can be viewed as a low-dimensional representation of the corresponding row of A. Similarly, each column of V_k^T is a k-dimensional representation of the corresponding column of A. Note that the rows of U_k, and the columns of V_k^T, are not orthonormal.

If an approximation with just two factors is desired, the matrix S_k can be included in U_k and/or V_k.

13.2 Principal component analysis

Standard principal component analysis (PCA) is the most widely used method for reducing the dimensionality of a dataset. The idea is to identify, one by one, the directions along which a dataset varies the most. These directions are called the principal components. The first direction is a line that goes through the mean of the data points, and maximizes the squared projection of each point onto the line. Each new direction is orthogonal to all previous ones.

Let A be a matrix with m rows corresponding to examples and n columns corresponding to features. The first step in PCA is to make each feature have mean zero, that is to subtract the mean of each column from each entry in it. Let this new data matrix be B, with rows b_1 to b_m. A direction is represented by a column vector w of length n with unit L_2 norm. The first principal component of B is

$$w_1 = \underset{||w||_2=1}{\operatorname{argmax}} \sum_{i=1}^{m} (b_i \cdot w)^2$$

Each data point b_i is a row vector of length n. The principal component is a column vector of the same length. The scalar $(b_i \cdot w)$ is the projection of b_i along w, which is the one-dimensional representation of the data point.

Each data point with the first component removed is

$$b_i' = b_i - (b_i \cdot w) w^T.$$

The second principal component can be found recursively as

$$w_2 = \underset{||w||_2=1}{\operatorname{argmax}} \sum_{i=1}^{m} (b_i' \cdot w)^2$$

Principal component analysis is widely used, but it has some major drawbacks. First, it is sensitive to the scaling of features. Without changing the correlations

between features, if some features are multiplied by constants, then the PCA changes. The principal components tend to be dominated by the features that are numerically largest. Second, PCA is unsupervised. It does not take into account any label that is to be predicted. The directions of maximum variation are usually not the directions that differentiate best between examples with different labels. Third, PCA is linear. If the data points lie in a cloud with nonlinear structure, PCA will not capture this structure.

PCA is closely related to SVD, which is used in practice to compute principal components and the re-representation of data points.

13.3 Latent semantic analysis

Polysemy is the phenomenon in language that a word such as "fire" can have multiple different meanings, Another important phenomenon is synonymy, the ability of different words such as "firearm" and "gun" to have similar meanings. Because of synonymy, the bag-of-words vectors representing two documents can be dissimilar, despite similarity in meaning between the documents. Conversely, document vectors may be similar despite different meanings, because of polysemy.

Latent semantic analysis (LSA), also called latent semantic indexing (LSI) is an attempt to improve the representation of documents by taking into account polysemy and synonymy in a way that is induced from the documents themselves. The key idea is to relate documents and words via concepts. These concepts are latent, i.e. hidden, because they are not provided explicitly. Concepts can also be called topics, or dimensions. The hope is that the concepts revealed by LSA do have semantic meaning.

Intuitively, a document is viewed as a weighted combination of concepts. Importantly, each document is created from more than one concept, so concepts are different from, and more expressive than, cluster centers. Synonymy is captured by permitting different word choices within one concept, while polysemy is captured by allowing the same word to appear in multiple concepts. The number of independent dimensions or concepts is usually chosen to be far smaller than the number of words in the vocabulary. A typical number is 300.

Concretely, LSI uses the singular value decomposition of the matrix A where each row is a document and each column is a word. Let $A = USV^T$ as above. Each row of U is interpreted as the representation of a document in concept space. Similarly, each column of V^T is interpreted as the representation of a word. Each singular value in S is the strength of the corresponding concept in the whole document collection.

Remember that the SVD can be written as

$$A = \sum_{k=1}^{r} \sigma_k u_k v_k^T$$

where u_k is the kth column of U and v_k^T is the kth row of V^T. Each vector v_k^T is essentially a concept, with a certain weight for each word. The vector u_k gives the strength of this concept for each document. A concept can be thought of as a pseudo-word; $U_{ik} = (u_k)_i$ is the count of pseudo-word k in document i.

Weighting schemes that transform the raw counts of words in documents are critical to the performance of LSI. A scheme that is popular is called log-entropy. The weight for word j in document i is

$$[1 + \sum_j p_{ij} \log_n p_{ij}] \cdot \log(x_{ij} + 1)$$

where p_{ij} is an estimate of the probability of occurrence of word j in document i, usually a maximum likelihood estimate. Note that words that are equally likely to occur in every document have the smallest weights.

LSI has drawbacks that research has not yet fully overcome. In no particular order, some of these are as follows.

- For large datasets, while A can be represented efficiently owing to its sparse nature, A_k cannot. For example, the TREC-2 collection used in research on information retrieval has 741,696 documents and 138,166 unique words. Since there are only 80 million non-zero entries in A, it can be stored in single precision floating point using 320 megabytes of memory. In contrast, because A_k is dense, storing it for any value of k would require about 410 gigabytes.

- Computing the SVD of large matrices is expensive. However, present day computational power and the efficient implementations of SVD, including stochastic gradient descent methods, make it possible to apply LSI to large collections.

- The first concept found by LSI typically has little semantic interest. Let u be the first column of U and let v be the first row of V^T. The vector u has an entry for every document, while v has an entry for every word in the vocabulary. Typically, the entries in u are roughly proportional to the lengths of the corresponding documents, while each entry in v is roughly proportional to the frequency in the whole collection of the corresponding word.

- Mathematically, every pair of rows of V^T must be orthogonal. This means that no correlations between topics are allowed.

- Apart from the first column, every column of u has about as many negative as positive entries; the same is true for rows of V^T. This means that every document has a negative loading for half the topics, and every topic has a negative loading for half the words.
- The rows of the U_k matrix representing rare words, which are often highly discriminative, typically have small norms. That is, the rare words have small loadings on every dimension. This has a negative impact on LSI performance in applications. Unfortunately, both log-entropy and traditional IDF schemes fail to address this problem.

13.4 Matrix factorization with missing data

Alternating least squares.

Convexity.

Regularization.

13.5 Spectral clustering

13.6 Graph-based regularization

Let w_{ij} be the similarity between nodes i and j in a graph. Assume that this similarity is the penalty for the two nodes having different labels. Let f_i be the label of node i, either 0 or 1. The cut size of the graph is

$$\begin{aligned}
\sum_{i,j:f_i \neq f_j} w_{ij} &= \sum_{i,j} w_{ij}(f_i - f_j)^2 \\
&= \sum_{i,j} w_{ij}(f_i^2 - 2f_i f_j + f_j^2) \\
&= \sum_{i,j} w_{ij}(f_i^2 + f_j^2) - 2f'Wf \\
&= 2\sum_i (\sum_j w_{ij})(f_i^2) - 2f'Wf
\end{aligned}$$

where f is a column vector. Let the sum of row i of the weight matrix W be $v_i = \sum_j w_{ij}$, and let V be the diagonal matrix with diagonal entries v_i. The cut size is then

$$-2f'Wf + 2f'Vf = 2f'(V - W)f.$$

The term involving W resembles an empirical loss function, while the term involving V resembles a regularizer.

Remember that f is a 0/1 vector. We can relax the problem by allowing f to be real-valued between 0 and 1. Finding the optimal f, given W, is then doable efficiently using eigenvectors; see the background section below. This approach is well-known in research on spectral clustering and on graph-based semi-supervised learning; see [Zhu and Goldberg, 2009, Section 5.3].

Let z_i be the suggested label of node i according to the base method mentioned in the previous section. The objective that we want to minimize is the linear combination

$$\begin{aligned}
\sum_i (f_i - z_i)^2 \quad &+ \quad \lambda \sum_{i,j:f_i \neq f_j} w_{ij} \\
&= \sum_i (f_i^2 - 2 f_i z_i + z_i^2) + 2\lambda f'(V - W) f \\
&= f'If - 2f'Iz + z'Iz + 2\lambda f'(V - W) f \\
&= -2f'Iz + z'Iz + 2\lambda f'(V - W + \frac{1}{2\lambda} I) f.
\end{aligned}$$

Minimizing this expression is equivalent to maximizing

$$f'Iz + \lambda f'(W - V - \frac{1}{2\lambda} I) f$$

where the term that does not depend on f has been omitted.

The following result is standard.

Theorem: Let A be symmetric and let B be positive definite. Then the maximum value of $x'Ax$ given $x'Bx = 1$ is the largest eigenvalue of $B^{-1}A$. The vector x that achieves the maximum is the corresponding eigenvector of $B^{-1}A$.

Exercises

1.

The compact singular value decomposition of the matrix A can be written as

$$A = \sum_{k=1}^{r} \sigma_k u_k v_k^T$$

where the vectors u_k are orthonormal, and so are the vectors v_k^T.

(a) What can you say about r in general?

(b) Suppose that one row of A is all zero. What can you say about r?

Chapter 14

Social network analytics

A social network is a graph where nodes represent individual people and links represent relationships. More generally, in a network nodes represent examples and edges represent relationships.

Examples:

- Telephone network. Nodes are subscribers, and edges are phone relationhips. Note that call detail records (CDRs) must be aggregated.
- Facebook.
- Protein-protein interactions.
- Gene-gene regulation.
- Papers citing papers.
- Same-author relationship between citations.
- Credit card applications.

There are two types of data mining one can do with a network: supervised and unsupervised. The aim of supervised learning is to obtain a model that can predict labels for nodes, or labels for edges. For nodes, this is sometimes called collective classification. For edges, the most basic label is existence. Predicting whether or not edges exist is called link prediction. (Predicting whether nodes exist has not been studied much.)

Examples of tasks that involve collective classification: predicting churn of subscribers; detecting fraudulent applicants. recognizing the emotional state of network members, predicting who will play a new game, predicting who will "like" a product or company.

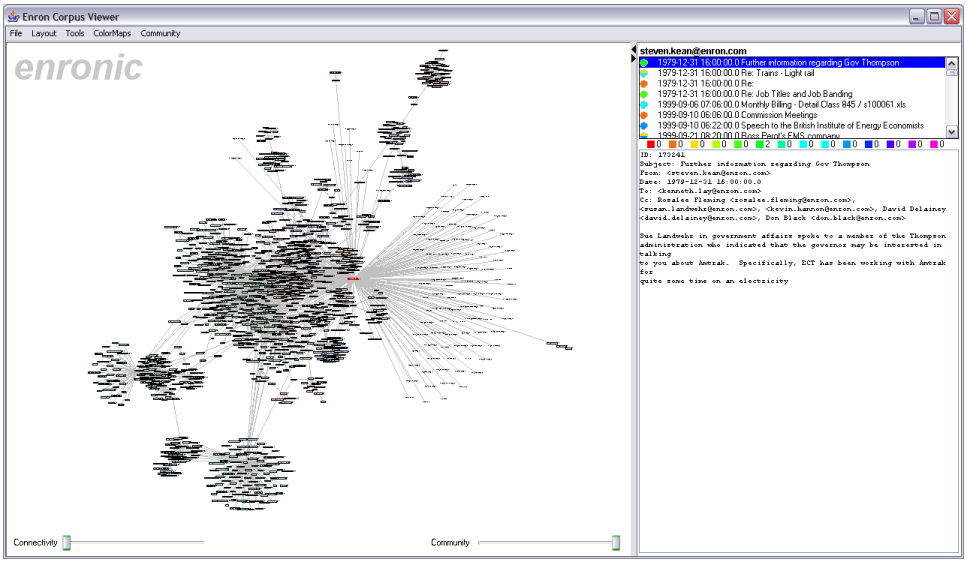

Figure 14.1: Visualization of the Enron email network from `http://jheer.org/enron/v1/enron_nice_1.png`.

Examples of tasks that involve link prediction: suggesting new friends for network members, identifying appropriate citations between scientific papers, predicting which pairs of terrorists know each other.

Structural versus dynamic link prediction: inferring unobserved edges versus forecasting future new edges.

14.1 Issues in network mining

Both collective classification and link prediction are transductive problems. Transduction is the situation where the set of test examples is known at the time a model is to be trained. In this situation, the real goal is not to train a model. Instead, it is simply to make predictions about specific examples. Some methods for transduction can make predictions without having an explicit reusable model. We will look at methods that do involve explicit models. However, these models will be usable only for nodes that belong to the part of the network that is known at training time. We will not solve the cold-start problem of making predictions for nodes that are not known during training.

Social networks, and other graphs in data mining, can be complex. In particular,

edges may be directed or undirected, they can be of multiple types, and/or they can be weighted. The graph can be bipartite, e.g. authors and papers.

Given a social network, nodes have two fundamental characteristics. First, they have identity. This means that we know which nodes are the same and which are different in the graph, but we do not necessarily know anything else about nodes. Second, nodes (and maybe edges) may be be associated with vectors that specify the values of features. These vectors are sometimes called side-information.

Sometimes an edge between two nodes means that they represent the same real-world entity. Recognizing that different records refer to the same person, or that different citations describe the same paper, is sometimes called deduplication or record linkage.

Maybe the most fundamental and most difficult issue in mining network data, including social networks, is the problem of missing information. More often than not, it is only known partially which edges exist in a network.

A network is a graph, and a graph can be represented by its adjacency matrix. Link prediction involves predicting values in the adjacency matrix that are unknown. Hence, link prediction is mathematically similar to collaborative filtering.

14.2 Unsupervised network mining

The aim of unsupervised data mining is often to come up with a model that explains the pattern of the connectivity of the network, in particular its statistics. Mathematical models that explain patterns of connectivity are often time-based, e.g. the "rich get richer" idea. This type of learning can be fascinating as sociology. For example, it has revealed that people who stop smoking tend to stop being friends with smokers, but people who lose weight do not stop being friends with obese people.

Most new friendships are between two people who already have at least one friend in common. Two people with a common friend are a path of length two. Most such paths never become a triangle, so an interesting question is which do? A strong but unpublished pattern is that if persons A and B became friends at a time close to when B and C became friends, then A and C are much more likely to be friends. This pattern highlights the benefit of knowing and using side information, such as time stamps for edges.

Often unsupervised data mining in a network does not allow making useful decisions in an obvious way. But some models of entire networks can lead to predictions, often about the evolution of the network, that can be useful for making decisions. For example, one can identify the most influential nodes, and make special efforts to reach them.

Unsupervised methods are often used for link prediction. Local topological features. The preferential attachment score is

$$a(x, y) = |\Gamma(x)| \cdot |\Gamma(y)|$$

where $\Gamma(x)$ is the set of neighbors of node x. The Adamic-Adar score is

$$a(x, y) = \sum_{z \in \Gamma(x) \bigcap \Gamma(y)} \frac{1}{\log |\Gamma(z)|}$$

while the Katz score is

$$a(x, y) = \sum_{l=1}^{\infty} \beta^l |\text{paths}(x, y, l)|$$

where paths(x, y, l) is the number of paths of length l from node x to node y. This measure gives reduced weight to longer paths. For link prediction, each of these scores is better for some networks and worse for others.

The Laplacian matrix of a graph is $D - A$ where A is the adjacency matrix and D is the diagonal matrix of vertex degrees. The modularity matrix has elements

$$B_{ij} = A_{ij} - \frac{D_{ii} D_{jj}}{2m}$$

where d_i is the degree of vertex i and m is the total number of edges. This is the actual minus the expected number of edges between the two vertices, where the expected number depends only on degrees. Many analysis methods that can be applied to the adjacency matrix can also be applied to the Laplacian or modularity matrix. There do exist alternative definitions of the Laplacian matrix.

14.3 Collective classification

In traditional classifier learning, the labels of examples are assumed to be independent. (This is not the same as assuming that the examples themselves are independent; see the discussion of sample selection bias in Section 10.2.) If labels are independent, then a classifier can make predictions separately for separate examples. However, if labels are not independent, then in principle a classifier can achieve higher accuracy by predicting labels for related examples in a related way. As mentioned, this situation is called collective classification.

Intuitively, the labels of neighbors are often correlated. Given a node x, we would like to use the labels of the neighbors of x as features when predicting the label of

x, and vice versa. A principled approach to collective classification would find mutually consistent predicted labels for x and its neighbors. However, in general there is no guarantee that mutually consistent labels are unique, and there is no general algorithm for inferring them.

A different approach has proved more successful. This approach is to extend each node with a fixed-length vector of feature values derived from its neighbors, and/or from the whole network. Then, a standard classifier learning method is applied. Because a node can have a varying number of neighbors, but standard learning algorithms need vector representations of fixed length, features based on neighbors are often aggregates. Aggregates may include sums, means, modes, minimums, maximums, counts, and indicator functions, notably for existence.

Conventional features describing persons may be implicit versions of aggregates based on social networks. For many data mining tasks where the examples are people, there are five general types of feature: personal demographic, group demographic, behavioral, situational, and social. If group demographic features, in particular, have predictive power, it can be because they are aggregates of features of linked people. For example, if zip code is predictive of the brand of automobile a person will purchase, that may be because of contagion effects between people who know each other, or see each other on the streets.

Often, the most useful vector of feature values for a node is obtained by low rank approximation of the adjacency matrix, or of a transformation of the adjacency matrix such as the Laplacian or modularity matrix. For example, the Enron email adjacency matrix shown in Figure 14.1 empirically has rank only 2 approximately.

The simplest case is when the adjacency matrix is fully known, edges are undirected, and they are unweighted or have real-valued weights. In this case, the standard SVD can be used directly. An undirected adjacency matrix A is symmetric, meaning $A = A^T$. For such a matrix the SVD is

$$A = USU^T = \sum_{k=1}^{r} \lambda_k u_k u_k^T$$

where each λ_k is a scalar that is a diagonal entry of the matrix S and the u_k vectors are the orthonormal columns of the matrix U. Each u_k vector has one entry for each node. Then, the vector $\langle U_{i1}, \ldots U_{ik} \rangle = \langle (u_1)_i, \ldots, (u_k)_i \rangle$ can be used as the fixed-length representation of the ith node of the graph. The representation length k can be chosen between 1 and r. Because the λ_i values are ordered from largest to smallest, intuitively the vector u_1 captures more of the overall structure of the network than u_2, and so on. The vectors u_k for large k may represent only random noise.

If edges have discrete labels from more than two classes, then the numerical SVD cannot be used directly. If some edges are unknown, then the standard SVD

is not appropriate. Instead, one should use a variant that allows some entries of the matrix A to be unknown. Approaches that are used for collaborative filtering, such as alternating least squares (ALS), can be applied.

Often, there are no edges that are known to be absent. In other words, if an edge is known, then it exists, but if an edge is unknown, it may or may not exist. This situation is sometimes called "positive only" or "positive unlabeled." In this case, it is reasonable to use the standard SVD where the value 1 means "present" and the value 0 means "unknown."

In many domains, in effect multiple networks are overlaid on the same nodes. For example, if nodes are persons then one set of edges may represent the "same address" relationship while another set may represent the "made telephone call to" relationship.

14.4 The social dimensions approach

The method of [Tang and Liu, 2009a] appears to be the best approach so far for predicting labels of nodes in a social network. The idea is as follows.

1. Compute the modularity matrix from the adjacency matrix.
2. Compute the top k eigenvectors of the modularity matrix.
3. Extend the vector representing each node with a vector of length k containing its component of each eigenvector.
4. Train a linear classifier using the extended vectors.

An intuitive weakness of this approach is that the eigenvectors are learned in an unsupervised way that ignores the specific collective classification task to be solved.

The method of [Tang and Liu, 2009a] has two limitations when applied to very large networks. First, the eigenvectors are dense so storing the extended vector for each node can use too much memory. Second, computing these eigenvectors may need too much time and/or too much memory. The method of [Tang and Liu, 2009b] is an alternative designed not to have these drawbacks. It works as follows.

1. Let n be the number of nodes and let e be the number of edges. Make a sparse binary dataset D with e rows and n columns, where each edge is represented by ones for its two nodes.
2. Use the k-means algorithm with cosine similarity to partition the set of rows into a large number k of disjoint groups. Note that each row represents one node; the number of ones in this row is the degree of the node.

3. Extend the vector representing each node with a sparse binary vector of length k. The jth component of this vector is 1 if and only if the node has at least one edge belonging to cluster number j.

4. Train a linear classifier using the extended vectors.

14.5 Supervised link prediction

Supervised link prediction is similar to collective classification, but involves some additional difficulties. First, the label to be predicted is typically highly unbalanced: between most pairs of nodes, there is no edge. The common response is to subsample pairs of nodes that are not joined by an edge, but a better solution is to use a ranking loss function during training.

The most successful general approach to link prediction is essentially the same as for collective classification: extend each node with a vector of feature values derived from the network, then use a combination of the vector for each node to predict the existence of an edge between the two nodes.

The vectors representing two nodes should not be combined by concatenation; see Section 7.4 above. A reasonable alternative is to combine the vectors buy point-wise multiplication.

A combination of vectors representing nodes is a representation of a possible edge. One can also represent a potential edge with edge-based feature values. For example, for predicting future co-authorship:

1. Keyword overlap is the most informative feature for predicting co-authorship.

2. Sum of neighbors and sum of papers are next most predictive; these features measure the propensity of an individual as opposed to any interaction between individuals.

3. Shortest-distance is the most informative graph-based feature.

Major open research issues are the cold-start problem for new nodes, and how to generalize to new networks.

Exercises

1.

Consider the task of predicting which pairs of nodes in a social network are linked. Specifically, suppose you have a training set of pairs for which edges are known to exist, or known not to exist. You need to make predictions for the remaining edges.

Let the adjacency matrix A have dimension n by n. You have trained a low-rank matrix factorization $A = UV$, where U has dimension n by r for some $r \ll n$. Because A is symmetric, V is the transpose of U.

Let u_i be the row of U that represents node i. Let $\langle j, k \rangle$ be a pair for which you need to predict whether an edge exists. Consider these two possible ways to make this prediction:

1. Simply predict the dot-product $u_j \cdot u_k$.
2. Predict $f([u_j, u_k])$ where $[u_j, u_k]$ is the concatenation of the vectors u_j and u_k, and f is a trained logistic regression model.

Explain an important reason why the first approach is better.

CSE 255 assignment due on June 7, 2013

This assignment is due on Friday, June 7, which is the last day of classes of the quarter. Please place a hardcopy of your report in Eric Christiansen's mailbox on the second floor of the CSE building. As usual, you should be part of a team of two students. (Remember that the final exam will be on Tuesday June 11 from 7pm to 10pm in WLH 2111, the usual classroom for 255.)

The goal is to apply and evaluate methods to predict the presence of connections in a social network, and/or their type. The dataset to use has information about 244 supposed terrorists. The information was compiled originally by the Profiles in Terror project.[1] The dataset[2] was created at the University of Maryland[3] and edited at UCSD by Eric Doi and Ke Tang. This information is far from fully correct or complete, but it is the best public data about terrorism that I know of. If you can find better data, or related data that would be interesting to combine with this data, please post on Piazza.

The dataset has 851 rows and 1227 columns. Each row describes a known link between two terrorists. The first and second columns are id numbers between 1 and 244 for the two terrorists. The last column of the dataset indicates the type of the known link. The value in the last column is 1 if the two terrorists are members of the same organization, and 0 if they have some other relationship (family members, contacted each other, or used the same facility such as a training camp). There are 461 positive same-organization links; see Figure 14.2. Of course, an unknown number of additional unknown links exist.

Columns 3 to 614 contain 612 binary feature values for the person identified by the first column, and columns 615 to 1226 contain values for the same features for the second person. Unfortunately the meaning of these features is not known precisely, and presumably the information is quite incomplete. Zero may mean missing more often than it means false; see Figure 14.3. For more details see a paper by Eric Doi.[4]

There are two different link prediction tasks for this dataset. One task is to predict the existence of edges, regardless of their type. The other task is to distinguish between same-organization links and links of other types. Note that each task involves only binary classification. For the first task the two classes can be labeled "present" and "absent" while for the second task the classes can be labeled "colleague" and "other."

---

[1] www.amazon.com/Profiles-Terror-Middle-Terrorist-Organizations/dp/0742535258

[2] cseweb.ucsd.edu/users/elkan/255/PITdata.csv

[3] www.cs.umd.edu/~sen/lbc-proj/LBC.html

[4] cseweb.ucsd.edu/users/elkan/255/Eric.Doi_report.pdf

When measuring performance for the first task, you can assume that the 851 known edges are the only edges that really exist. Similarly, you can measure performance for the second task using only the edges with known labels 0 and 1. The paper by Eric Doi considers the latter task, for which results are presented in its Figure 6. Design your experiments so that your results can be compared to those in that table.

For each task, do ten-fold cross-validation. For the first task, this means pretending that 85 true edges are unknown. Then, based on the $851 - 85 = 766$ known edges, and on all 244 nodes, you predict a score for each of the $244 \cdot 243/2$ potential edges. The higher the score of the 85 held-out edges, the better.

Use a linear classifier such as logistic regression to compute a score for each potential edge. When computing scores, you may use features obtained from (i) the network structure of known and/or unknown edges, (ii) properties of nodes, and (iii) properties of pairs of nodes. For (i), use matrix factorization to convert a transformation of the adjacency matrix into a vector of feature values for each node, as discussed in class. Investigate whether some transformations, for example some definitions of Laplacian, are better than others.

While designing your experiments, think carefully about the following issues. Discuss these issues in your report.

- During cross-validation, exactly what information is it legitimate to include in test examples?
- How could you take into account the fact that, almost certainly, some edges are present but are not among the 851 known edges?
- How can you take into account the fact that many feature values are likely missing for many nodes?
- How can you avoid the mistakes discussed in Section 7.4 above?
- Are variations in accuracy between different methods reliable, or could they be due to randomness?

In your report, have separate sections for "Design of experiments" and "Results of experiments." In the results section, in order to allow comparisons with previous work, do report 0/1 accuracy for the task of distinguishing between same-organization links and links of other types. However, also discuss whether or not this is a good measure of success.

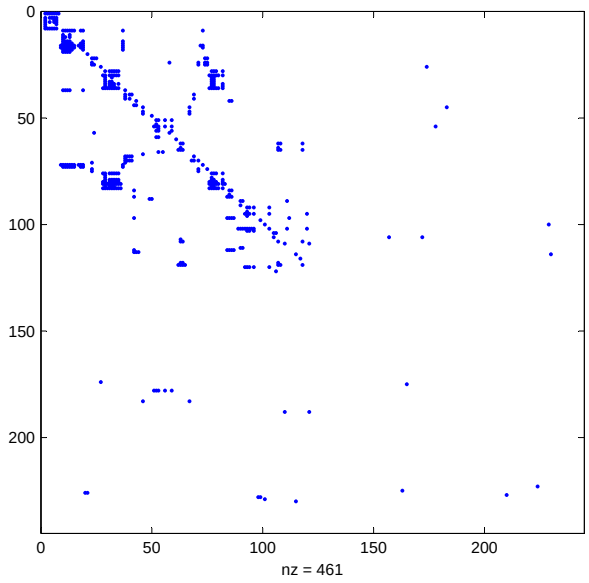

Figure 14.2: Visualization of 461 “same organization” links between 244 supposed terrorists.

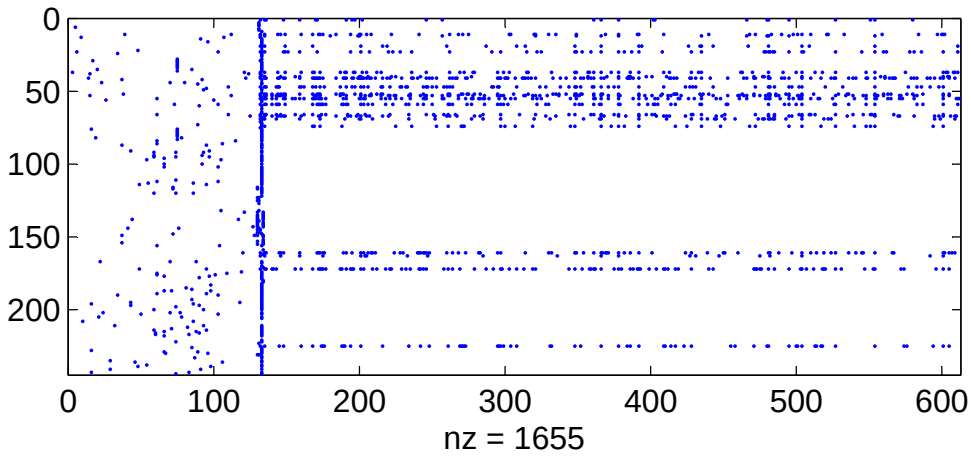

Figure 14.3: Visualization of feature values for the 244 supposed terrorists.

Chapter 15

Interactive experimentation

This chapter will discuss data-driven optimization of actions taken, in particular via A/B testing.

A/B testing can reveal "willingness to pay." Compare profit at prices x and y, based on number sold. Choose the price that yields higher profit.

Formalizing A/B testing requires the classical notions of testing hypotheses and calculating confidence intervals. For the latter, see Section 9.6.

Exercises

1.

The following text is from a New York Times article dated May 30, 2009.

> Mr. Herman had run 27 ads on the Web for his client Vespa, the scooter company. Some were rectangular, some square. And the text varied: One tagline said, "Smart looks. Smarter purchase," and displayed a $0 down, 0 percent interest offer. Another read, "Pure fun. And function," and promoted a free T-shirt.
>
> Vespa's goal was to find out whether a financial offer would attract customers, and Mr. Herman's data concluded that it did. The $0 down offer attracted 71 percent more responses from one group of Web surfers than the average of all the Vespa ads, while the T-shirt offer drew 29 percent fewer.

(a) What basic principle of the scientific method did Mr. Herman not follow, according to the description above?

(b) Suppose that it is true that the financial offer does attract the highest number of purchasers. What is an important reason why the other offer might still be preferable for the company?

(c) Explain the writing mistake in the phrase "Mr. Herman's data concluded that it did." Note that the error is relatively high-level; it is not a spelling or grammar mistake.

Chapter 16

Reinforcement learning

Reinforcement learning is the type of learning done by an agent who is trying to figure out a good policy for interacting with an environment. The framework is that time is discrete, and at each time step the agent perceives the current state of the environment and selects a single action. After performing the action, the agent earns a real-valued reward or penalty, time moves forward, and the environment shifts into a new state. The goal of the agent is to learn a policy for choosing actions that leads to the best possible long-term total of rewards.

Reinforcement learning is fundamentally different from supervised learning because explicit correct input/output pairs are never given to the agent. That is, the agent is never told what the optimal action is in a given state. Nor is the agent ever told what its future rewards will be. However, at each time step the agent does find out what its current short-term reward is.

Reinforcement learning has real-world applications in domains where there is an intrinsic tradeoff between short-term and long-term rewards and penalties, and where there is an environment that can be modeled as unknown and random. Successful application areas include controlling robots, playing games, scheduling elevators, treating cancer, and managing relationships with customers [Abe et al., 2002, Simester et al., 2006]. In the latter three examples, the environment consists of humans, which are modeled collectively as acting probabilistically. More specifically, each person is a different instantiation of the environment. The environment is considered to be a set of states. Different people who happen to be in the same state are assumed to have the same probability distribution of reactions.

Reinforcement learning is appropriate for domains where the agent faces a stochastic but non-adversarial environment. It is not suitable for domains where the agent faces an intelligent opponent, because in adversarial domains the opponent does not change state probabilistically. For example, reinforcement learning methods are not

appropriate for learning to play chess. However, reinforcement learning can be successful for learning to play games that involve both strategy and chance, such as backgammon, if it is reasonable to model the opponent as following a randomized strategy. Game theory is the appropriate formal framework for modeling agents interacting with intelligent opponents. Of course, finding good algorithms is even harder in game theory than in reinforcement learning.

A fundamental distinction in reinforcement learning is whether learning is online or offline. In offline learning, the agent is given a database of recorded experience interacting with the environment. The job of the agent is to learn the best possible policy for directing future interaction with the same environment. Note that this is different from learning to predict the rewards achieved by following the policy used to generate the recorded experience. The learning that is needed is called off-policy since it is about a policy that is different from the one used to generate training data.

In online learning, the agent must balance exploration, which lets it learn more about the environment, with exploitation, where it uses its current estimate of the best policy to earn rewards. Most research on reinforcement learning has concerned online learning, but offline learning is more useful for many applications.

16.1 Markov decision processes

The basic reinforcement learning scenario is formalized using three concepts:

1. a set S of potential states of the environment,
2. a set A of potential actions of the agent, and
3. rewards that are real numbers.

At each time t, the agent perceives the state $s_t \in S$ and its available actions A. The agent chooses one action $a \in A$ and receives from the environment the reward r_t and then the new state s_{t+1}.

Formally, the environment is represented as a so-called Markov decision process (MDP). An MDP is a four-tuple $\langle S, A, p(\cdot|\cdot,\cdot), r(\cdot,\cdot)\rangle$ where

- S is the set of states, which is also called the state space,
- A is the set of actions,
- $r(s,a)$ is the immediate reward received after doing action a from state s.
- $p(s_{t+1} = s'|s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t+1$, and

The states of an MDP possess the Markov property. This means that if the current state of the MDP at time t is known, then transitions to a new state at time $t+1$ are independent of all previous states. In a nutshell, history is irrelevant.

Some extensions of the MDP framework are possible without changing the nature of the problem fundamentally. In particular, the reward may be a function of the next state as well as of the current state and action, that is the reward can be a function $r(s, a, s')$. Or, for maximal mathematical simplicity, the reward can be a function just of the current state, $r(s)$. Or, the MDP definition can be extended to allow r to be stochastic as opposed to deterministic. In this case, $p(r|s, a, s')$ is the probability that the immediate reward is r, given state s, action a, and next state s'. It is crucial that the probability distributions of state transitions and rewards are stationary: transitions and rewards may be stochastic, but their probabilities are the same at all times.

The goal of the agent is to maximize a cumulative function R of the rewards. If the maximum time step is $t = T$ where T is finite, then the goal can be to maximize $R = r_0 + r_1 + \ldots + r_T$. If the maximum time horizon is infinite, then the goal is to maximize a discounted sum that involves a discounting factor γ, which is usually a number just under 1:

$$R = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t).$$

The factor γ makes the sum above finite even though it is a sum over an infinite number of time steps.

In order to maximize its total reward, the agent must acquire a policy, i.e. a mapping $\pi : S \to A$ that specifies which action to take for every state. An optimal policy π^* is one that maximizes the expected value of R, where the expectation is based on the transition probability distribution $p(s_{t+1} = s'|s_t = s, a_t = \pi^*(s_t))$.

Given an MDP, an optimal policy is sometimes called a solution of the MDP. If the time horizon is infinite, then optimal policies have the Markov property also: the best action to choose depends on the current state only, regardless of prior history. With a finite time horizon, the best action may be a function $\pi^*(s, t)$ of the current time t as well as the current state t.

16.2 RL versus cost-sensitive learning

Remember that cost-sensitive learning (CSL) is rather a misnomer; a more accurate name is cost-sensitive decision-making. The differences between CSL and RL include the following.

- In CSL, the customer is assumed to be in a certain state, and then a single characteristic becomes true; this is the label to be predicted. In RL, every

aspect of the state of the customer can change from one time point to the next.

- In CSL, the customer's label is assumed to be independent of the agent's prediction. In RL, the next state depends on the agent's action.
- In CSL, the agent chooses an action based on separate estimates of the customer's label and the immediate benefit to the agent of each label. In RL, the agent chooses an action to maximize long-term benefit.
- In CSL, long-term effects of actions can only be taken into account heuristically. In RL, these are learned; consider priming and annoyance for example.
- In CSL, the benefit must be estimated for every possible action. But the actual benefit is observed only for one action.
- In CSL, a customer is typically represented by a real-valued vector. In RL, the state of a customer is typically one of a number of discrete classes.

Both CSL and RL face the issue of possibly inadequate exploration. Suppose that the agent learns in 2010 never to give a loan to owners of convertibles, based on data from 2009. Then in 2011, when the agent wants to learn from 2010 data, it will have no evidence that owners of convertibles are bad risks.

16.3 Algorithms to find an optimal policy

If the agent knows the transition probabilities $p(s'|s,a)$ of the MDP, then it can calculate an optimal policy; no learning is required. To see how to find the optimal policy, consider a real-valued function U of states called the utility function. Intuitively, $U^\pi(s)$ is the expected total reward achieved by starting in state s and acting according to the policy π. Recursively,

$$U^\pi(s) = r(s,\pi(s)) + \gamma \sum_{s'} p(s'|s,\pi(s)) U^\pi(s').$$

The equation above is called the Bellman equation. The Bellman optimality equation describes the optimal policy:

$$U^*(s) = \max_a [r(s,a) + \gamma \sum_{s'} p(s'|s,\pi(s)) U^*(s')].$$

Let the function V be an approximation of the function U^*. We can get an improved approximation by picking any state s and doing the update

$$V(s) := \max_a [r(s,a) + \gamma \sum_{s'} p(s'|s,a) V(s')]. \tag{16.1}$$

The value iteration algorithm of Bellman (1957) is simply to do this single step repeatedly for all states s in any order. If the state space S is finite, then V can be stored as a vector and it can be initialized to be $V(s) = 0$ for each state s.

Lemma: The optimal utility function U^* is unique, and the V function converges to U^* regardless of how V is initialized.

Proof: Assume that the state space S is finite, so the function $V(s)$ can be viewed as a vector. Let V_1 and V_2 be two such vectors, and let V_1' and V_2' be the results of updating V_1 and V_2. Consider the maximum discrepancy between V_1' and V_2':

$$\begin{aligned} d &= \max\{V_1' - V_2'\} \\ &= \max\{\max_{a_1}[r(s,a_1) + \gamma \sum_{s'} p(s'|s,a_1)V_1(s')] \\ &\qquad - \max_{a_2}[r(s,a_2) + \gamma \sum_{s'} p(s'|s,a_2)V_2(s')]\}. \end{aligned}$$

This maximum discrepancy goes to zero. ■

At any point in the algorithm, a current guess for the optimal policy can be computed from the approximate utility function V as

$$\pi(s) = \operatorname*{argmax}_a [r(s,a) + \gamma \sum_{s'} p(s'|s,a)V(s')]. \tag{16.2}$$

Note that (16.2) is identical to (16.1) except for the argmax operator in the place of the max operator. An important fact about the value iteration method is that $\pi(s)$ may converge to the optimal policy long before V converges to the true U.

An alternative algorithm called policy iteration is due to Howard (1960). This method repeats two steps until convergence:

1. Compute V exactly based on π.
2. Update π based on V.

Step 2 is performed using Equation (16.2). Computing V exactly in Step 1 is formulated and solved as a set of simultaneous linear equations. There is one equation for each s in S:

$$V(s) = r(s,\pi(s)) + \gamma \sum_{s'} p(s'|s,a)V(s').$$

Policy iteration has the advantage that there is a clear stopping condition: when the function π does not change after performing Step 1, then the algorithm terminates.

16.4 Q functions

An agent can execute the algorithms from the previous section, i.e. value iteration and policy iteration, only if it knows the transition probabilities of the environment. If these probabilities are unknown, then the agent must learn them at the same time it is trying to discover an optimal policy. This learning problem is obviously more difficult. It is called reinforcement learning.

Rather than learn transition probabilities or an optimal policy directly, some of the most successful RL methods learn a so-called Q function. A Q function is an extended utility function. Specifically, the Q^π function is like the U^π function except that it has two arguments and it only follows the policy π starting in the next state:

$$Q^\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) Q^\pi(s', \pi(s')).$$

In words, the Q^π function is the value of taking the action a and then continuing according to the policy π. Given a Q function, the recommended action for a state s is $\hat{a} = \pi^Q(s) = \text{argmax}_a \, Q(s,a)$.

When using a Q function, at each time step the action a is determined by a maximization operation that is a function only of the current state s. From this point of view, at each time step actions seem to be chosen in a greedy way, ignoring the future. The key that makes the approach in fact not greedy is that the Q function is learned. The learned Q function emphasizes aspects of the state that are predictive of long-term reward.

16.5 The Q-learning algorithm

Experience during learning consists of $\langle s, a \rangle$ pairs together with an observed reward r and an observed next state s'. That is, the agent was in state s, it tried doing a, it received r, and s' happened. The agent can use each such fragment of experience to update the Q function directly. For any state s_t and any action a_t, the agent can update the Q function as follows:

$$Q(s_t, a_t) := (1-\alpha) Q(s_t, a_t) + \alpha [r_t + \gamma \max_a Q(s_{t+1}, a)].$$

In this update rule, r_t is the observed immediate reward, α is a learning rate such that $0 < \alpha < 1$, and γ is the discount rate.

Assume that the state space and the action space are finite, so we can use a two-dimensional table to store the Q function. The Q learning algorithm is as follows:

1. Initialize Q values, and start in some state s_0.

2. Given the current state s, choose an action a.
3. Observe immediate reward r and the next state $s' = s_{t+1}$.
4. Update $Q(s, a)$ using the update rule above.
5. Continue from Step 2.

This algorithm is guaranteed to converge to optimal Q values, if (i) the environment is stationary, (ii) the environment satisfies the Markov assumption, (iii) each state is visited repeatedly, (iv) each action is tried in each state repeatedly, and (v) the learning rate parameter decreases over time. All these assumptions are reasonable. However, the algorithm may converge extremely slowly.

The learning algorithm does not say which action to select at each step. We need a separate so-called exploration strategy. This strategy must trade off exploration, that is trying unusual actions, with exploitation, which means preferring actions with high estimated Q values.

An ϵ-greedy policy selects with probability $1 - \epsilon$ the action that is optimal according to the current Q function, and with probability ϵ it chooses the action from a uniform distribution over all other actions. In other words, with probability ϵ the policy does exploration, while with probability $1 - \epsilon$ it does exploitation.

A softmax policy chooses action a_i with probability proportional to $\exp[bQ(s, a_i)]$ where b is a constant. This means that actions with higher estimated Q values are more likely to be chosen (exploitation), but every action always has a non-zero chance of being chosen (exploration). Smaller values of b make the softmax strategy explore more, while larger values make it do more exploitation.

Q-learning has at least two major drawbacks. First, there is no known general method for choosing a good learning rate α. Second, if $Q(s, a)$ is approximated by a continuous function, for example a neural network, then when the value of $Q(s, a)$ is updated for one $\langle s, a \rangle$ pair, the Q values of other state-action pairs are changed in unpredictable ways.

The Q-learning algorithm is designed for online learning. If historical training data is available, Q-learning can be applied, but the next section explains a better alternative.

16.6 Fitted Q iteration

As mentioned, in the batch setting for RL, an optimal policy or Q function must be learned from historical data [Neumann, 2008]. A single historical training example is a quadruple $\langle s, a, r, s' \rangle$ where s is a state, a is the action actually taken in this state, r is the immediate reward that was obtained, and s' is the state that was observed to come next. The method named Q-iteration [Murphy, 2005, Ernst et al., 2005] is the following algorithm:

Define $Q_0(s, a) = 0$ for all s and a.
For horizon $h = 0, 1, 2, \ldots$
    For each example $\langle s, a, r, s' \rangle$
        let $v = r + \gamma \max_b Q_h(s', b)$
    Train Q_{h+1} with examples $\langle s, a, v \rangle$.

The variable h is called the horizon, because it counts how many steps of lookahead are implicit in the learned Q function. In some applications, in particular medical clinical trials, the maximum horizon is a small fixed integer. In these cases the discount factor γ can be set to one.

Fitted Q-iteration has significant advantages over Q-learning. No learning rate is needed, and all training Q values are fitted simultaneously, so the underlying supervised learning algorithm can make sure that all of them are fitted reasonably well.

Another major advantage of fitted Q-iteration, compared to other methods for offline reinforcement learning, is that the historical training data can be collected in any way that is convenient. There is no need to collect trajectories starting at any specific states. Indeed, there is no need to collect trajectories of consecutive states, that is of the form $\langle s_1, a_1, r_1, s_2, a_2, r_2, s_3, \ldots \rangle$. And, there is no need to know the policy π that was followed to generate historical episodes of the form $\langle s, a = \pi(s), r, s' \rangle$. Q-iteration is an off-policy method, meaning that data collected while following one or more arbitrary non-optimal policies can be used to learn a better policy.

Off-policy methods for RL face the problem of bias in sample selection. The probability distribution of training examples $\langle s, a \rangle$ is different from the probability distribution of optimal examples $\langle s, \pi^*(s) \rangle$. The fundamental reason why Q-iteration is correct as an off-policy method is that it is discriminative as opposed to generative. Q-iteration does not model in any way the distribution of state-action pairs $\langle s, a \rangle$. Instead, it uses a purely discriminative method to learn only how the values to be predicted depend on s and a.

16.7 Representing continuous states and actions

Requiring that actions and states be discrete is restrictive. To be useful for many applications, a reinforcement learning algorithm should allow states and actions to be represented as real-valued vectors. For example, in many business domains the state is essentially the current characteristics of a customer, who is represented by a high-dimensional real-valued vector [Simester et al., 2006].

Using the fitted Q iteration algorithm, or many other methods, there are two basic operations that need to be performed with a Q function. First, finding optimal actions should be efficient. Second, the Q function must be learnable from training examples.

Suppose that a Q function is written as $Q(s, a) = s^T W a$. This is called a bilinear representation. It has important advantages. Given s, the optimal action is

$$a^* = \underset{b}{\text{argmax}}\ Q(s, b) = \underset{b}{\text{argmax}}\ x \cdot b$$

where $x = s^T W$. This maximization, subject to linear restrictions on the components of b, is a linear programming (LP) task. Hence, it is tractable in practice, even for high-dimensional action vectors. The linear constraints may be lower and upper bounds for components (sub-actions) of b. They may also involve multiple components of b, for example a budget limit in combination with different costs for different sub-actions.[1]

With the bilinear approach, the Q function is learnable in a straightforward way by standard linear regression. Let the vectors s and a be of length m and n respectively, so that $W \in \mathbb{R}^{m \times n}$. The key is to notice that

$$s^T W a = \sum_{i=1}^{m} \sum_{j=1}^{n} (W \circ sa^T)_{ij} = \mathit{vec}(W) \cdot \mathit{vec}(sa^T). \tag{16.3}$$

In this equation, sa^T is the matrix that is the outer product of the vectors s and a, and $\circ$ denotes the elementwise product of two matrices, which is sometimes called the Hadamard product. The notation $\mathit{vec}(A)$ means the matrix A converted into a vector by concatenating its columns. Based on Equation (16.3), each training triple $\langle s, a, v \rangle$ can be converted into the pair $\langle \mathit{vec}(sa^T), v \rangle$ and $\mathit{vec}(W)$ can be learned by standard linear regression.

The bilinear representation of Q functions contains an interaction term for every component of the action vector and of the state vector. Hence, it can represent the consequences of each action component as depending on each aspect of the state.

[1] The LP approach can handle constraints that combine two or more components of the action vector. Given constraints of this nature, the optimal value for each action component is in general not simply its lower or upper bound. Even when optimal action values are always lower or upper bounds, the LP approach is far from trivial in high-dimensional action spaces: it finds the optimal value for every action component in polynomial time, whereas naive search might need $O(2^d)$ time where d is the dimension of the action space.

The solution to an LP problem is always a vertex in the feasible region. Therefore, the maximization operation always selects a so-called "bang-bang" action vector, each component of which is an extreme value. In many domains, actions that change smoothly over time are desirable. In at least some domains, the criteria to be minimized or maximized to achieve desirably smooth solutions are known, for example the "minimum jerk" principle [Viviani and Flash, 1995]. These criteria can be included as additional penalties in the maximization problem to be solved to find the optimal action. If the criteria are nonlinear, the maximization problem will be harder to solve, but it may still be convex, and it will still be possible to learn a bilinear Q function representing how long-term reward depends on the current state and action vectors.

Some other representations for Q functions are less satisfactory. The simplest representation, as mentioned above, is tabular: a separate value is stored for each combination of a state value s and an action value a. This representation is usable only when both the state space and the action space are discrete and of low cardinality. A general representation is linear, using fixed basis functions. In this representation $Q(s,a) = w \cdot [\phi_1(s,a), \cdots, \phi_m(s,a)]$ where ϕ_1 to ϕ_m are fixed real-valued functions and w is a vector of length m. The bilinear approach is a special case of this representation where each basis function is the product of a state component and an action component. An intuitive drawback of other cases of this representation is that they ignore the distinction between states and actions: essentially, s and a are concatenated as inputs to the basis functions ϕ_i. A computational drawback is that, in general, no efficient way of performing the maximization operation $\mathrm{argmax}_b \ Q(s,b)$ is known, whether or not there are constraints on feasible action vectors b. Other common representations are neural networks [Riedmiller, 2005] and ensembles of decision trees [Ernst et al., 2005]; these have the same difficulty.

Note that any fixed format for Q functions, including a bilinear representation, in general makes the whole RL process suboptimal, since the true optimization problem that represents long-term reward may not be expressible in this format.

16.8 Inventory management applications

In inventory management applications, at each time step the agent sees a demand vector and/or a supply vector for a number of products. The agent must decide how to satisfy each demand in order to maximize long-term benefit, subject to various rules about the substitutability and perishability of products. Managing the stocks of a blood bank is a typical application of this type. As a concrete illustration, this section describes results using the specific formulation used in [Yu, 2007] and Chapter 12 of [Powell, 2007].

The blood bank stores blood of eight types: AB+, AB-, A+, A-, B+, B-, O+, and O-. Each period, the manager sees a certain level of demand for each type, and a certain level of supply. Some types of blood can be substituted for some others; 27 of the 64 conceivable substitutions are allowed. Blood that is not used gets older, and beyond a certain age must be discarded. With three discrete ages, the inventory of the blood bank is a vector in $\mathbb{R}^{24}$. The state of the system is this vector concatenated with the 8-dimensional demand vector and a unit constant component. The action vector has $3 \cdot 27 + 1 = 82$ dimensions since there are 3 ages, 27 possible allocations, and a unit constant pseudo-action. Each component of the action vector is a quantity of blood of one type and age, used to meet demand for blood of the same or another type. The LP solved to determine the action at each time step has learned coefficients

Table 16.1: Short-term reward functions for blood bank management.

	[Yu, 2007]	new
supply exact blood type	50	0
substitute O- blood	60	0
substitute other type	45	0
fail to meet demand	0	-60
discard blood	-20	0

in its objective function, but fixed constraints such as that the total quantity supplied from each of the 24 stocks must be not be more than the current stock amount.

For training and for testing, a trajectory is a series of periods. In each period, the agent sees a demand vector drawn randomly from a specific distribution. Then, blood remaining in stock is aged by one period, blood that is too old is discarded, and fresh blood arrives according to a different probability distribution. Training is based on 1000 trajectories of length 10 periods in which the agent follows a greedy policy.[2] Testing is based on trajectories of the same length where supply and demand vectors are drawn from the same distributions, but the agent follows a learned policy.

In many real-world situations the objectives to be maximized, both short-term and long-term, are subject to debate. Intuitive measures of long-term success may be not mathematically consistent with intuitive immediate reward functions, or even with the MDP framework. The blood bank scenario illustrates this issue.

Table 16.1 shows two different immediate reward functions. Each entry is the benefit accrued by meeting one unit of demand with supply of a certain nature. The middle column is the short-term reward function used in previous work, while the right column is an alternative that is more consistent with the long-term evaluation measure used previously (described below).

For a blood bank, the percentage of unsatisfied demand is an important long-term measure of success [Yu, 2007]. If a small fraction of demand is not met, that is acceptable because all high-priority demands can still be met. However, if more than 10% of demand for any type is not met in a given period, then some patients

[2] The greedy policy is the policy that supplies blood to maximize one-step reward, at each time step. Many domains are like the blood bank domain in that the optimal one-step policy, also called myopic or greedy, can be formulated directly as a linear programming problem with known coefficients. With a bilinear representation of the Q function, coefficients are learned that formulate the long-term optimal policy as a problem in the same class. The linear representation is presumably not sufficiently expressive to represent the truly optimal long-term policy, but it is expressive enough to represent a policy that is better than the greedy one.

Table 16.2: Success of alternative learned policies.

	average unmet A+ demand	frequency of severe unmet A+ demand
greedy policy	7.3%	46%
bilinear Q-iteration (i)	18.4%	29%
bilinear Q-iteration (ii)	7.55%	12%

may suffer seriously. Therefore, we measure both the average percentage of unmet demand and the frequency of unmet demand over 10%. The A+ blood type is an especially good indicator of success, because the discrepancy between supply and demand is greatest for it: on average, 34.00% of demand but only 27.94% of supply is for A+ blood.

Table 16.2 shows the performance of the greedy policy and of policies learned by fitted Q iteration using the two immediate reward functions of Table 16.1. Given the immediate reward function (i), fitted Q-iteration satisfies on average less of the demand for A+ blood. However, this reward function does not include an explicit penalty for failing to meet demand. The bilinear method correctly maximizes (approximately) the discounted long-term average of immediate rewards. The last column of Table 16.1 shows a different immediate reward function that does penalize failures to meet demand. With this reward function, fitted Q iteration learns a policy that more than cuts in half the frequency of severe failures.

Bibliography

[Abe et al., 2002] Abe, N., Pednault, E., Wang, H., Zadrozny, B., Fan, W., and Apte, C. (2002). Empirical comparison of various reinforcement learning strategies for sequential targeted marketing. In *Proceedings of the IEEE International Conference on Data Mining*, pages 3–10. IEEE.

[Ernst et al., 2005] Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(1):503–556.

[Huang et al., 2006] Huang, J., Smola, A., Gretton, A., Borgwardt, K. M., and Schölkopf, B. (2006). Correcting sample selection bias by unlabeled data. In *Proceedings of the Neural Information Processing Systems Conference (NIPS 2006)*.

[Jonas and Harper, 2006] Jonas, J. and Harper, J. (2006). Effective counterterrorism and the limited role of predictive data mining. Technical report, Cato Institute. Available at `http://www.cato.org/pub_display.php?pub_id=6784`.

[Michie et al., 1994] Michie, D., Spiegelhalter, D. J., and Taylor, C. C. (1994). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood.

[Murphy, 2005] Murphy, S. A. (2005). A generalization error for Q-learning. *Journal of Machine Learning Research*, 6:1073–1097.

[Neumann, 2008] Neumann, G. (2008). Batch-mode reinforcement learning for continuous state spaces: A survey. *ÖGAI Journal*, 27(1):15–23.

[Powell, 2007] Powell, W. B. (2007). *Approximate Dynamic Programming*. John Wiley & Sons, Inc.

[Riedmiller, 2005] Riedmiller, M. (2005). Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the 16th European Conference on Machine Learning (ECML)*, pages 317–328.

[Simester et al., 2006] Simester, D. I., Sun, P., and Tsitsiklis, J. N. (2006). Dynamic catalog mailing policies. *Management Science*, 52(5):683–696.

[Tang and Liu, 2009a] Tang, L. and Liu, H. (2009a). Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 817–826. ACM.

[Tang and Liu, 2009b] Tang, L. and Liu, H. (2009b). Scalable learning of collective behavior based on sparse social dimensions. In *Proceeding of the 18th ACM Conference on Information and Knowledge Management (CIKM)*, pages 1107–1116. ACM.

[Vert and Jacob, 2008] Vert, J.-P. and Jacob, L. (2008). Machine learning for in silico virtual screening and chemical genomics: New strategies. *Combinatorial Chemistry & High Throughput Screening*, 11(8):677–685(9).

[Viviani and Flash, 1995] Viviani, P. and Flash, T. (1995). Minimum-jerk, two-thirds power law, and isochrony: converging approaches to movement planning. *Journal of Experimental Psychology*, 21:32–53.

[Yu, 2007] Yu, V. (2007). Approximate dynamic programming for blood inventory management. Honors thesis, Princeton University.

[Zhu and Goldberg, 2009] Zhu, X. and Goldberg, A. B. (2009). Introduction to semi-supervised learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 3(1):1–130.