Evaluating Classifiers
Last lecture...

How can we predict **binary** or **categorical** variables?

\[ f(\text{data}) \rightarrow \text{labels} \]

\{0,1\}, \{True, False\}

\{1, ..., N\}
Last lecture...

**Will I purchase this product?**
(yes)

**Will I click on this ad?**
(no)
Last lecture...

• **Naïve Bayes**
  - Probabilistic model (fits $p(label|data)$)
  - Makes a conditional independence assumption of the form $(feature_i \perp feature_j | label)$ allowing us to define the model by computing $p(feature_i | label)$ for each feature
  - Simple to compute just by counting

• **Logistic Regression**
  - Fixes the “double counting” problem present in naïve Bayes

• **SVMs**
  - Non-probabilistic: optimizes the classification error rather than the likelihood
1) Naïve Bayes

\[
p(label|features) = \frac{p(label)p(features|label)}{p(features)}
\]

due to our conditional independence assumption:

\[
p(label|features) = \frac{p(label) \prod_i p(feature_i|label)}{p(features)}
\]
2) logistic regression

sigmoid function: \[ \sigma(t) = \frac{1}{1+e^{-t}} \]
Logistic regression

- Logistic regressors don’t optimize the number of “mistakes”
- No special attention is paid to the “difficult” instances – every instance influences the model
- But “easy” instances can affect the model (and in a bad way!)
- How can we develop a classifier that optimizes the number of mislabeled examples?
Q: Where would a logistic regressor place the decision boundary for these features?
3) Support Vector Machines

Try to optimize the **misclassification error** rather than maximize a probability.
Support Vector Machines

This is essentially the intuition behind Support Vector Machines (SVMs) – train a classifier that focuses on the “difficult” examples by minimizing the misclassification error.

We still want a classifier of the form

\[ y_i = \begin{cases} 
1 & \text{if } X_i \cdot \theta - \alpha > 0 \\
-1 & \text{otherwise} 
\end{cases} \]

But we want to minimize the number of misclassifications:

\[ \arg \min_{\theta} \sum_i \delta(y_i(X_i \cdot \theta - \alpha) \leq 0) \]
Simple (seperable) case: there exists a perfect classifier
Support Vector Machines

The classifier is defined by the hyperplane $\theta \mathbf{x} - \alpha = 0$
Q: Is one of these classifiers preferable over the others?
A: Choose the classifier that maximizes the distance to the nearest point
Support Vector Machines

Distance from a point to a line?
Support Vector Machines

\[ \theta x - \alpha = 1 \]
\[ \theta x - \alpha = 0 \]
\[ \theta x - \alpha = -1 \]

\[ \arg \min_{\theta, \alpha} \frac{1}{2} \|\theta\|_2^2 \]
\[ \text{such that} \]
\[ \forall i \ y_i (\theta \cdot X_i - \alpha) \geq 1 \]
This is known as a "quadratic program" (QP) and can be solved using "standard" techniques.

\[
\begin{align*}
\arg \min_{\theta, \alpha} & \quad \frac{1}{2} \|\theta\|^2 \\
\text{such that} & \\
\forall_i y_i (\theta \cdot X_i - \alpha) & \geq 1
\end{align*}
\]

See e.g. Nocedal & Wright ("Numerical Optimization"), 2006
But: is finding such a separating hyperplane even possible?
Support Vector Machines

Or: is it actually a good idea?
Support Vector Machines

Want the margin to be as wide as possible

While penalizing points on the wrong side of it
Support Vector Machines

Soft-margin formulation:

$$\arg \min_{\theta, \alpha} \frac{1}{2} \|\theta\|_2^2$$

such that

$$\forall_i y_i (\theta \cdot X_i - \alpha) \geq 1$$
The classifiers we’ve seen this week all attempt to make decisions by associating weights (theta) with features (x) and classifying according to

\[ y_i = \begin{cases} 
1 & \text{if } X_i \cdot \theta > 0 \\
0 & \text{otherwise}
\end{cases} \]
Summary

• **Naïve Bayes**
  • Probabilistic model (fits $p(\text{label}|\text{data})$)
  • Makes a conditional independence assumption of the form $(\text{feature}_i \perp \text{feature}_j | \text{label})$ allowing us to define the model by computing $p(\text{feature}_i | \text{label})$ for each feature
  • Simple to compute just by counting

• **Logistic Regression**
  • Fixes the “double counting” problem present in naïve Bayes

• **SVMs**
  • Non-probabilistic: optimizes the classification error rather than the likelihood
Pros/cons

• **Naïve Bayes**
  ++ Easiest to implement, most efficient to “train”
  ++ If we have a process that generates feature that *are* independent given the label, it’s a very sensible idea
  -- Otherwise it suffers from a “double-counting” issue

• **Logistic Regression**
  ++ Fixes the “double counting” problem present in naïve Bayes
  -- More expensive to train

• **SVMs**
  ++ Non-probabilistic: optimizes the classification error rather than the likelihood
  -- More expensive to train
Judging a book by its cover

Images features are available for each book on
http://jmcauley.ucsd.edu/cse258/data/amazon/book_images_5000.json

4096-dimensional image features
Judging a book by its cover

Example: train an SVM to predict whether a book is a children’s book from its cover art

(code available on)
http://jmcauley.ucsd.edu/cse258/code/week2.py
Judging a book by its cover

• The number of errors we made was extremely low, yet our classifier doesn’t seem to be very good – why?
Evaluating classifiers
Which of these classifiers is best?
Which of these classifiers is best?

The solution which minimizes the #errors may not be the best one.
Which of these classifiers is best?

1. When data are highly imbalanced

If there are far fewer positive examples than negative examples we may want to assign additional weight to negative instances (or vice versa)

e.g. will I purchase a product? If I purchase 0.00001% of products, then a classifier which just predicts “no” everywhere is 99.99999% accurate, but not very useful
Which of these classifiers is best?

2. When mistakes are more costly in one direction
   False positives are nuisances but false negatives are disastrous (or vice versa)

  e.g. which of these bags contains a weapon?
Which of these classifiers is best?

3. When we only care about the “most confident” predictions

e.g. does a relevant result appear among the first page of results?
Evaluating classifiers

decision boundary

negative positive
Evaluating classifiers

TP (true positive): Labeled as positive, predicted as positive.
Evaluating classifiers

TN (true negative): Labeled as negative, predicted as negative
Evaluating classifiers

FP (false positive): Labeled as negative, predicted as positive
Evaluating classifiers

**FN (false negative):** Labeled as negative, predicted as positive.
Evaluating classifiers

<table>
<thead>
<tr>
<th>Label</th>
<th>Prediction</th>
<th>true</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true positive</td>
<td>false positive</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>false negative</td>
<td>true negative</td>
<td></td>
</tr>
</tbody>
</table>

Classification accuracy = \frac{\text{correct predictions}}{\#\text{predictions}}

Error rate = \frac{\text{incorrect predictions}}{\#\text{predictions}}
Evaluating classifiers

<table>
<thead>
<tr>
<th>Prediction</th>
<th>Label</th>
<th>Prediction</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true positive</td>
<td>true positive</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false negative</td>
<td>true negative</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false positive</td>
<td>false negative</td>
</tr>
</tbody>
</table>

True positive rate (TPR) = true positives / #labeled positive

True negative rate (TNR) = true negatives / #labeled negative
Evaluating classifiers

<table>
<thead>
<tr>
<th>Label</th>
<th>true</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>true positive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>false positive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>false negative</td>
<td></td>
<td></td>
</tr>
<tr>
<td>true negative</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Balanced Error Rate (BER) = $\frac{1}{2} \ (FPR + FNR)$

= $\frac{1}{2}$ for a random/naïve classifier, 0 for a perfect classifier
Evaluating classifiers

e.g.

\[ y = [1, -1, 1, 1, 1, -1, 1, 1, -1, 1] \]

Confidence = [1.3, -0.2, -0.1, -0.4, 1.4, 0.1, 0.8, 0.6, -0.8, 1.0]
Evaluating classifiers

How to optimize a balanced error measure:

\[ L_\theta(y|X) = \prod_{y_i=1} p_\theta(y_i|X_i) \prod_{y_i=0} (1 - p_\theta(y_i|X_i)) \]
Evaluating classifiers – ranking

The classifiers we’ve seen can associate **scores** with each prediction.

- Furthest from decision boundary in negative direction = lowest score/least confident
- Furthest from decision boundary in positive direction = highest score/most confident
The classifiers we’ve seen can associate **scores** with each prediction

- In ranking settings, the actual labels assigned to the points (i.e., which side of the decision boundary they lie on) **don’t matter**
- All that matters is that positively labeled points tend to be at **higher ranks** than negative ones
The classifiers we’ve seen can associate **scores** with each prediction

- For naïve Bayes, the “score” is the ratio between an item having a positive or negative class
- For logistic regression, the “score” is just the probability associated with the label being 1
- For Support Vector Machines, the score is the distance of the item from the decision boundary (together with the sign indicating what side it’s on)
Evaluating classifiers – ranking

The classifiers we’ve seen can associate **scores** with each prediction

e.g.

\[
\mathbf{y} = [1, -1, 1, 1, 1, -1, 1, 1, 1, -1, 1]
\]

**Confidence** = [1.3, -0.2, -0.1, -0.4, 1.4, 0.1, 0.8, 0.6, -0.8, 1.0]

Sort **both** according to confidence:
Evaluating classifiers – ranking

The classifiers we’ve seen can associate **scores** with each prediction

Labels sorted by confidence:

\[
[1, 1, 1, 1, 1, -1, 1, -1, 1, -1]
\]

Suppose we have a fixed budget (say, six) of items that we can return (e.g. we have space for six results in an interface)

- Total number of **relevant** items =
- Number of items we returned =
- Number of **relevant items** we returned =
Evaluating classifiers – ranking

The classifiers we’ve seen can associate **scores** with each prediction.

\[
\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}
\]

“fraction of retrieved documents that are relevant”

\[
\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}
\]

“fraction of relevant documents that were retrieved”
Evaluating classifiers – ranking

The classifiers we’ve seen can associate **scores** with each prediction.

\[
\text{precision}@k = \text{precision when we have a budget of } k \text{ retrieved documents}
\]

e.g.

- Total number of **relevant** items = 7
- Number of items we returned = 6
- Number of **relevant items** we returned = 5

\[
\text{precision}@6 =
\]
Evaluating classifiers – ranking

The classifiers we’ve seen can associate **scores** with each prediction.

\[
F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}
\]

(harmonic mean of precision and recall)

\[
F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}
\]

(weighted, in case precision is more important (low beta), or recall is more important (high beta))
How does our classifier behave as we “increase the budget” of the number retrieved items?

- For budgets of size 1 to N, compute the precision and recall
- Plot the precision against the recall
When data are highly imbalanced

If there are far fewer positive examples than negative examples we may want to assign additional weight to negative instances (or vice versa).

e.g. will I purchase a product? If I purchase 0.00001% of products, then a classifier which just predicts “no” everywhere is 99.99999% accurate, but not very useful.

Compute the true positive rate and true negative rate, and the F_1 score.
2. When mistakes are more costly in one direction
False positives are nuisances but false negatives are disastrous (or vice versa)

Compute “weighted” error measures that trade-off the precision and the recall, like the $F_{\beta}$ score

e.g. which of these bags contains a weapon?
3. When we only care about the "most confident" predictions e.g. does a relevant result appear among the first page of results?

Compute the precision@k, and plot the signature of precision versus recall
So far: Regression

How can we use **features** such as product properties and user demographics to make predictions about **real-valued** outcomes (e.g. star ratings)?

How can we prevent our models from **overfitting** by favouring simpler models over more complex ones?

How can we assess our decision to optimize a particular error measure, like the MSE?
So far: Classification

Next we adapted these ideas to **binary** or **multiclass** outputs.

What animal is in this image?  Will I **purchase** this product?  Will I **click on** this ad?

Combining features using naïve Bayes models

![Graph of logistic regression function](image)

Support vector machines
So far: supervised learning

Given **labeled training data** of the form

\[ \{(\text{data}_1, \text{label}_1), \ldots, (\text{data}_n, \text{label}_n)\} \]

Infer the function

\[ f(\text{data}) \rightarrow \text{labels} \]
So far: supervised learning

We’ve looked at two types of prediction algorithms:

**Regression**

\[ y_i = X_i \cdot \theta \]

**Classification**

\[ y_i = \begin{cases} 
1 & \text{if } X_i \cdot \theta > 0 \\
0 & \text{otherwise}
\end{cases} \]
Further reading:

• “Cheat sheet” of performance evaluation measures:

• Andrew Zisserman’s SVM slides, focused on computer vision:
  http://www.robots.ox.ac.uk/~az/lectures/ml/lect2.pdf