Python Data Products

Course 3: Making Meaningful Predictions from Data

Lecture: Implementing a regularization pipeline in Python

Learning objectives

In this lecture we will...

- Show how to implement the training/validation/testing pipeline in Python
- Show how to select regularization parameters, and evaluate model performance using a validation set

Validation pipeline

To summarize our validation pipeline so far, our goal is to:

- Split the data into train/validation/test fractions
- Consider several different values of our hyperparameters (e.g. lambda)
- For each of these values, train a model on the **training set**
- Evaluate each model's performance on the **validation set**
- For the model that performs best on the validation set, evaluate its performance on the **test set**

Code: data and problem setup

First, let's set up our prediction problem (which is mostly code we've seen before):

In [49]:
```
# Training / validation / test pipeline
```

In [50]:
```
import gzip
from collections import defaultdict
import string
import random
```

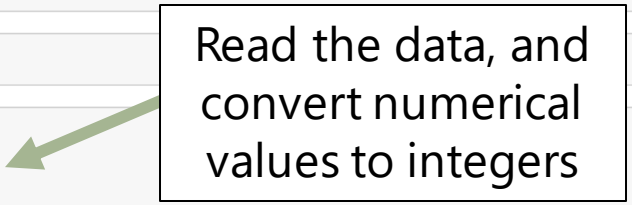
In [51]:
```
path = "/home/jmcauley/datasets/mooc/amazon/amazon_reviews_us_Gift_Card_v1_00.tsv.gz"
```

In [52]:
```
f = gzip.open(path, 'rt', encoding="utf8")
```

In [53]:
```
header = f.readline()
header = header.strip().split('\t')
```

In [54]:
```
dataset = []
```

In [55]:
```
for line in f:
    fields = line.strip().split('\t')
    d = dict(zip(header, fields))
    d['star_rating'] = int(d['star_rating'])
    d['helpful_votes'] = int(d['helpful_votes'])
    d['total_votes'] = int(d['total_votes'])
    dataset.append(d)
```



Code: data and problem setup

Next we extract features (again, much as we did in previous examples):

In [57]:
```
wordCount = defaultdict(int)
punctuation = set(string.punctuation)

for d in dataset:
  r = ''.join([c for c in d['review_body'].lower() if not c in punctuation])
  for w in r.split():
    wordCount[w] += 1

counts = [(wordCount[w], w) for w in wordCount]
counts.sort()
counts.reverse()

words = [x[1] for x in counts[:1000]]

wordId = dict(zip(words, range(len(words))))
wordSet = set(words)
```

Counting instances of words in each review

In [58]:
```
def feature(datum):
    feat = [0]*len(words)
    r = ''.join([c for c in datum['review_body'].lower() if not c in punctuation])
    for w in r.split():
        if w in words:
            feat[wordId[w]] += 1
    feat.append(1) #offset
    return feat
```

Code: splitting the data

Now, our first task is to split the data into training, validation, and test samples:

```
In [59]: random.shuffle(dataset)

In [60]: X = [feature(d) for d in dataset]

In [61]: y = [d['star_rating'] for d in dataset]

In [62]: N = len(X)
         X_train = X[:N//2]
         X_valid = X[N//2:3*N//4]
         X_test = X[3*N//4:]
         y_train = y[:N//2]
         y_valid = y[N//2:3*N//4]
         y_test = y[3*N//4:]

In [63]: len(X), len(X_train), len(X_valid), len(X_test)
Out[63]: (149086, 74543, 37271, 37272)
```

Remember to **shuffle** the dataset, so that our train/valid/test sets are i.i.d. samples

This example uses 50%/25%/25% (non-overlapping) splits, though other ratios would also be possible

Code: regression model

Again, we'll use the "Ridge" model from sklearn, which allows us to implement regression with a regularizer

```
In [64]: from sklearn import linear_model

In [65]: help(linear_model.Ridge)
```

```
Help on class Ridge in module sklearn.linear_model.ridge:

class Ridge(_BaseRidge, sklearn.base.RegressorMixin)
 |  Linear least squares with l2 regularization.
 |
 |  This model solves a regression model where the loss function is
 |  the linear least squares function and regularization is given by
 |  the l2-norm. Also known as Ridge Regression or Tikhonov regularization.
 |  This estimator has built-in support for multi-variate regression
 |  (i.e., when y is a 2d-array of shape [n_samples, n_targets]).
 |
 |  Read more in the :ref:`User Guide <ridge_regression>`.
 |
 |  Parameters
 |  ----------
 |  alpha : {float, array-like}, shape (n_targets)
 |      Small positive values of alpha improve the conditioning of the problem
 |      and reduce the variance of the estimates.  Alpha corresponds to
 |      ``C^-1`` in other linear models such as LogisticRegression or
 |      LinearSVC. If an array is passed, penalties are assumed to be specific
```

Regularization parameter

Code: regression model

Set up a quick utility function to calculate the MSE for a particular model:

In [66]:
```
def MSE(model, X, y):
    predictions = model.predict(X)
    differences = [(a-b)**2 for (a,b) in zip(predictions, y)]
    return sum(differences) / len(differences)
```

Code: regression model

Train the model for a range of regularization parameters:

In [67]:
```
bestModel = None
bestMSE = None
```

Keep track of which model worked the best

In [68]:
```
for lamb in [0.01, 0.1, 1, 10, 100]:
    model = linear_model.Ridge(lamb, fit_intercept=False)
    model.fit(X_train, y_train)

    mseTrain = MSE(model, X_train, y_train)
    mseValid = MSE(model, X_valid, y_valid)

    print("lambda = " + str(lamb) + ", training/validation error = " +
          str(mseTrain) + '/' + str(mseValid))
    if not bestModel or mseValid < bestMSE:
        bestModel = model
        bestMSE = mseValid
```

Fit a model for each lambda value

Report the training and validation error (but not the test error yet!)

```
lambda = 0.01, training/validation error = 0.4187297059927889/0.4481159192463995
lambda = 0.1, training/validation error = 0.41872971449864577/0.44810067260735315
lambda = 1, training/validation error = 0.41873055597795/0.44795079625281725
lambda = 10, training/validation error = 0.41880648152201755/0.44668116707720923
lambda = 100, training/validation error = 0.42244982510706414/0.4437512972171302
```

Code: regression model

Finally, report the **test error** for the model that had the best performance on the **validation set**

In [69]:
```
mseTest = MSE(bestModel, X_test, y_test)
print("test error = " + str(mseTest))
```

```
test error = 0.44550653361941783
```

Summary of concepts

- Showed a full pipeline of model selection and evaluation on a real dataset

On your own...

- Reproduce this pipeline for a different task, e.g. for a classification experiment