Lecture: Implementing a regularization pipeline in Python
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
First, let's set up our prediction problem (which is mostly code we've seen before):

```python
# Training / validation / test pipeline

import gzip
from collections import defaultdict
import string
import random

path = "~/home/jmcauley/datasets/mooc/amazon/amazon_reviews_us_Gift_Card_v1_00.tsv.gz"

f = gzip.open(path, 'rt', encoding="utf8")

header = f.readline()
header = header.strip().split('"

dataset = []

for line in f:
    fields = line.strip().split('"
    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)
```

Read the data, and convert numerical values to integers
Next we extract features (again, much as we did in previous examples):

```python
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)

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
```

Counting instances of words in each review
Now, our first task is to split the data into training, validation, and test samples:

```python
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:]
```

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.

```python
In [63]: len(X), len(X_train), len(X_valid), len(X_test)
Out[63]: (149086, 74543, 37271, 37272)
```
Again, we'll use the "Ridge" model from sklearn, which allows us to implement regression with a regularizer.
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)
```
Train the model for a range of regularization parameters:

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

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

lambda = 0.01, training/validation error = 0.418729765927889/0.4481159192463995
lambda = 0.1, training/validation error = 0.41872971449864577/0.44810067260735315
lambda = 1, training/validation error = 0.41873055597795/0.44795079625281725
lambda = 10, training/validation error = 0.418806431226175/0.44668116707726923
lambda = 100, training/validation error = 0.42244982510706414/0.4437512972171302
```
Finally, report the **test error** for the model that had the best performance on the **validation set**.

```python
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