HW1

October 15, 2018

```
In [1]: import urllib
        import numpy as np
        from sklearn.svm import LinearSVC
```

1 Regression

```
In [2]: def parseData(fname):
            for l in urllib.request.urlopen(fname):
                yield eval(l)

        data = list(parseData("http://jmcauley.ucsd.edu/cse255/data/beer/beer_50000.json"))

        reviews = {}
        for d in data:
            reviews[d['review/taste']] = reviews.get(d['review/taste'], 0) + 1

        reviews

Out[2]: {1.0: 211,
         1.5: 343,
         2.0: 1099,
         2.5: 1624,
         3.0: 4137,
         3.5: 8797,
         4.0: 16575,
         4.5: 12883,
         5.0: 4331}
```

```
In [3]: feature1 = [d['beer/style']=='Hefeweizen' for d in data]
        feature2 = [d['beer/ABV'] for d in data]
        target = [d['review/taste'] for d in data]

        X = np.stack((np.ones(len(data)), feature1, feature2), -1)
        Y = np.array(target).reshape(-1, 1)

        theta,residuals,rank,s = np.linalg.lstsq(X, Y, rcond=None)
        theta
        # Theta0 represents bias, theta1 and theta2 represent how much
        # weight 'beer are a Hefeweizen' and 'beer/ABV' contribute
        # to taste prediction.
```

```
Out[3]: array([[ 3.11795084],
               [-0.05637406],
               [ 0.10877902]])
```

```
In [4]: def split_data(X, Y, holdout_ratio, shuffle=False):
            m = X.shape[0]
            m_holdout = int(m * holdout_ratio)

            if shuffle:
                permutation = np.random.permutation(m)
                X = X[permutation, :]
                Y = Y[permutation, :]

            return X[:-m_holdout, :], Y[:-m_holdout, :], X[-m_holdout:, :], Y[-m_holdout:, :]

        X_train, y_train, X_test, y_test = split_data(X, Y, 0.5)
        theta,residuals,rank,s = np.linalg.lstsq(X_train, y_train, rcond=None)

        mse_train = np.mean((np.dot(X_train, theta) - y_train)**2)
        mse_test = np.mean((np.dot(X_test, theta) - y_test)**2)

        print('Train MSE: %f Test MSE: %f' % (mse_train, mse_test))
```

```
Train MSE: 0.483968 Test MSE: 0.423707
```

```
In [5]: X_train, y_train, X_test, y_test = split_data(X, Y, 0.5, shuffle=True)
        theta,residuals,rank,s = np.linalg.lstsq(X_train, y_train, rcond=None)

        mse_train = np.mean((np.dot(X_train, theta) - y_train)**2)
        mse_test = np.mean((np.dot(X_test, theta) - y_test)**2)

        print('Train MSE: %f Test MSE: %f' % (mse_train, mse_test))

        # In previous experiment, first half of data may be harder to predict than the
        # second half. Random split may reduce bias in original dataset distribution.
```

```
Train MSE: 0.454771 Test MSE: 0.444600
```

```
In [6]: feature1 = [d['beer/ABV'] if d['beer/style']=='Hefeweizen' else 0 for d in data]
        feature2 = [d['beer/ABV'] if d['beer/style']!='Hefeweizen' else 0 for d in data]
        X = np.stack((np.ones(len(data)), feature1, feature2), -1)
        X_train, y_train, X_test, y_test = split_data(X, Y, 0.5)
        theta,residuals,rank,s = np.linalg.lstsq(X_train, y_train, rcond=None)

        mse_train = np.mean((np.dot(X_train, theta) - y_train)**2)
        mse_test = np.mean((np.dot(X_test, theta) - y_test)**2)

        print('Train MSE: %f Test MSE: %f' % (mse_train, mse_test))
```

```
Train MSE: 0.483967 Test MSE: 0.423697
```

```
In [7]: # The features are different. In Q2-4, 'beer/style is Hefeweizen' is showed
        # explicitly, while in Q5 it is implied from both features.
```

2 Classification

```
In [8]: feature1 = [d['review/taste'] for d in data]
        feature2 = [d['review/appearance'] for d in data]
        feature3 = [d['review/aroma'] for d in data]
        feature4 = [d['review/palate'] for d in data]
        feature5 = [d['review/overall'] for d in data]
        target = [d['beer/style'] == 'Hefeweizen' for d in data]

        X = np.stack((np.ones(len(data)), feature1, feature2, feature3, feature4, feature5), -1)
        Y = np.array(target, dtype=np.int).reshape(-1, 1)

        X_train, y_train, X_test, y_test = split_data(X, Y, 0.5, shuffle=True)

        clf = LinearSVC(C=1000)
        clf.fit(X_train, y_train.reshape(-1))

        train_accuracy = clf.score(X_train, y_train)
        test_accuracy = clf.score(X_test, y_test)
        print('Train accuracy: %f Test accuracy: %f' % (train_accuracy, test_accuracy))
```

```
Train accuracy: 0.987000 Test accuracy: 0.988280
```

```
/home/fanjin/.local/lib/python3.5/site-packages/sklearn/svm/base.py:922: ConvergenceWarning: Lib
  "the number of iterations.", ConvergenceWarning)
```

```
In [9]: text_hef = [d['review/text'] for d in data if d['beer/style'] == 'Hefeweizen']
        text_non_hef = [d['review/text'] for d in data if d['beer/style'] != 'Hefeweizen']

        import re
        import nltk
        from nltk.corpus import stopwords
```

```
         # nltk.download('stopwords', download_dir='.')
         nltk.data.path.append(".");

         def createCorpus(text):
             corpus = {}
             for t in text:
                 words = re.findall(r'[^\s()!,.?":;0-9]+', t.lower())
                 for w in set(words):
                     corpus[w] = corpus.get(w, 0) + 1

             remove = [k for k in corpus.keys() if k in stopwords.words('english')]
             for k in remove: del corpus[k]
             return corpus

         hef_corpus = createCorpus(text_hef)
         nonhef_corpus = createCorpus(text_non_hef)

         hef_top_words = [w[0] for w in sorted(hef_corpus.items(), key=lambda x: x[1],
                                              reverse=True)[:50]]
         nonhef_top_words = [w[0] for w in sorted(nonhef_corpus.items(),
                                                 key=lambda x: x[1], reverse=True)[:300]]

         token = [w for w in hef_top_words if w not in nonhef_top_words]
         token

         # Extract top 50 most frequent occurrence words in
         # Hefeweizen review text (exclude stopwords)
         # minus top 300 most frequent occurence words in non-Hefeweizen review text.

Out[9]: ['banana', 'wheat', 'clove', 'hefe', 'lemon', 'hefeweizen']

In [10]: def hasToken(text):
             words = re.findall(r'[^\s()!,.?":;0-9]+', text.lower())
             cnt = 0
             for w in set(words):
                 if w in token:
                     cnt += 1
             return cnt

         feature6 = [hasToken(d['review/text']) for d in data]

         # Feature6 counts how many token words occur in review texts.

In [12]: X = np.stack((np.ones(len(data)), feature1, feature2, feature3,
                       feature4, feature5, feature6), -1)
         Y = np.array(target, dtype=np.int).reshape(-1, 1)
```

```
        X_train, y_train, X_test, y_test = split_data(X, Y, 0.5, shuffle=True)

        clf = LinearSVC(C=1000)
        clf.fit(X_train, y_train.reshape(-1))

        train_accuracy = clf.score(X_train, y_train)
        test_accuracy = clf.score(X_test, y_test)
        print('Train accuracy: %f Test accuracy: %f' % (train_accuracy, test_accuracy))
```

```
Train accuracy: 0.991480 Test accuracy: 0.991000
```

```
/home/fanjin/.local/lib/python3.5/site-packages/sklearn/svm/base.py:922: ConvergenceWarning: Lib
  "the number of iterations.", ConvergenceWarning)
```

```
In [ ]:
```