

Fall 2018

Jupyter: Intro to Data Science - Lecture 11 Movie Reviews

Grant Long
CUNY City College

NYC Tech-in-Residence Corps

Follow this and additional works at: https://academicworks.cuny.edu/cc_oers



Part of the [Computer Sciences Commons](#)

[How does access to this work benefit you? Let us know!](#)

Recommended Citation

Long, Grant and NYC Tech-in-Residence Corps, "Jupyter: Intro to Data Science - Lecture 11 Movie Reviews" (2018). *CUNY Academic Works*.
https://academicworks.cuny.edu/cc_oers/159

This Lecture or Presentation is brought to you for free and open access by the City College of New York at CUNY Academic Works. It has been accepted for inclusion in Open Educational Resources by an authorized administrator of CUNY Academic Works. For more information, please contact AcademicWorks@cuny.edu.

Data Dive 11: Movie Reviews

Today, we'll build a naive bayes classifier to judge whether a movie review is positive or negative. In today's exercise, we'll be using a sample of movie reviews [compiled by Stanford University's CS Department \(http://ai.stanford.edu/~amaas/data/sentiment/\)](http://ai.stanford.edu/~amaas/data/sentiment/).



Courtesy of [@AmznMovieRevws \(https://twitter.com/AmznMovieRevws\)](https://twitter.com/AmznMovieRevws)

```
In [ ]: import os
        %matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import KFold, train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier

random_state = 20181118
```

```
In [ ]: corpus_df = pd.read_csv('https://grantmlong.com/data/sentiment_corpus.csv')
        corpus_df.shape
```

Inspect df contents.

```
In [ ]: for i in np.random.choice(corpus_df.index.values, 4):
        print(corpus_df.label.values[i])
        print(corpus_df.text.values[i], '\n\n')
```

Type *Markdown* and LaTeX: α^2

```
In [ ]:
```

```
In [ ]:
```

Process Input Data

Split Train and Holdout Data Sets

```

In [ ]: x_train, x_holdout, y_train, y_holdout = train_test_split(
        corpus_df['text'],
        corpus_df['label'], test_size=0.1,
        random_state=random_state
    )

train_df = pd.DataFrame(x_train, columns=['text'])
train_df['label'] = y_train

holdout_df = pd.DataFrame(x_holdout, columns=['text'])
holdout_df['label'] = y_holdout

train_df.reset_index(inplace=True)
holdout_df.reset_index(inplace=True)

print(train_df.shape[0], np.mean((train_df.label=='positive')*1))
print(holdout_df.shape[0], np.mean((holdout_df.label=='positive')*1))

```

Create Function to Vectorize Text

The functionality scikit-learn's [CountVectorizer](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html) (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html) makes it really easy to transform words to text.

```

In [ ]: def featurize_text(df, n_features=1000, max_df=0.50, min_df=50):

        vectorizer = CountVectorizer(max_df=max_df,
                                    min_df=min_df,
                                    max_features=n_features,
                                    stop_words='english')

        X = (vectorizer.fit_transform(df.text)>0)*1
        Y = (df.label=='positive')*1

        return X, Y, vectorizer

```

Inspect Vectorized Data

```

In [ ]: X, Y, vectorizer = featurize_text(train_df, n_features=1000)

```

```
In [ ]: word_df = pd.DataFrame(
        data = X.todense(),
        columns=vectorizer.get_feature_names()
    )

count_df = pd.DataFrame(
    data = [word_df.sum(),
           word_df.loc[train_df.label=='positive'].sum(),
           word_df.loc[train_df.label=='negative'].sum()],
    index = ['total_count', 'pos_count', 'neg_count']
).transpose()

count_df['pos_share'] = count_df.pos_count/count_df.total_count*100
count_df['neg_share'] = count_df.neg_count/count_df.total_count*100
```

```
In [ ]: (count_df[['total_count', 'pos_share', 'neg_share']]
        .sort_values(by='total_count', ascending=False)[:30])
```

Cross Validate Vectorization Parameters

Here we use `scikit-learn` to train a naive bayes classifier to rate the sentiment of each movie review. For more details of `scikit-learn` see [here \(https://scikit-learn.org/stable/modules/naive_bayes.html\)](https://scikit-learn.org/stable/modules/naive_bayes.html).

- What are the hyperparameters for this model?
- Does it makes sense to train more than one?
- What should we expect to see from our learning curves? Will more words overfit?

```
In [ ]: k_fold = KFold(n_splits=5, random_state=random_state)
```

```
In [ ]: def get_cv_results(df, n_features=1000, max_df=0.80, min_df=50):

    nbayes = BernoulliNB()
    X, Y, _ = featurize_text(df, n_features, max_df, min_df)

    results = []
    for train, test in k_fold.split(X):
        nbayes.fit(X[train, :], Y[train])
        y_predicted = nbayes.predict(X[test, :])
        accuracy = accuracy_score(Y[test], y_predicted)
        results.append(accuracy)

    return np.mean(results), np.std(results)
```

```
In [ ]: get_cv_results(train_df)
```

```
In [ ]: hp_values = [100, 250, 500, 750, 1000, 1500, 2000, 2500, 3000, 3500, 4000]
all_mu = []
all_sigma = []

for m in hp_values:

    mu, sigma = get_cv_results(train_df, n_features=m)
    all_mu.append(mu)
    all_sigma.append(sigma)

print(m, mu, sigma)
```

```
In [ ]: plt.figure(figsize=(14, 5))
plt.plot(hp_values, all_mu)
plt.ylabel('Cross Validation Accuracy')
plt.xlabel('Max Depth')
```

```
In [ ]:
```

```
In [ ]:
```

Train Optimal Model

```
In [ ]: X, Y, vectorizer = featurize_text(train_df, n_features=2500)
nbayes = BernoulliNB()
nbayes.fit(X, Y)
```

Create Test Feature and Target Variables using the vectorizer we produced above.

- Why do we use the vectorizer produced above instead of creating a new one?

```
In [ ]: X_test = vectorizer.transform(holdout_df.text)
Y_test = (holdout_df.label=='positive')*1

# Generate predicted labels (positive=1)
y_predicted = nbayes.predict(X_test)

# Generate predicted probabilities
y_probpos = nbayes.predict_proba(X_test)[:, 1]

print(X_test.shape)
```

Inspect in-sample performance.

```
In [ ]: np.random.seed(random_state + 0)
for i in np.random.choice(holdout_df.index.values, 4):
    print('Index:          %i' % i)
    print('Prob. Positive:  %0.3f' % nbayes.predict_proba(X_test[i])[0][1])
    print('Label:          %s\n' % holdout_df.label.values[i])
    print(holdout_df.text.values[i], '\n\n')
```

Report Overall Out of Sample Performance

- Based on these results, do you think our classifier is overfitting?

```
In [ ]: def report_performance(classifier):
    y_predicted = classifier.predict(X_test)
    print('No. of test samples:  %i' % len(y_predicted))
    print('Accuracy:            %0.1f%%' % (accuracy_score(Y_test, y_predicted)))
    print('Precision:           %0.1f%%' % (precision_score(Y_test, y_predicted)))
    print('Recall:              %0.1f%%' % (recall_score(Y_test, y_predicted)))

report_performance(nbayes)
```

Examine our false positives

```
In [ ]: np.random.seed(random_state + 0)
for i in np.random.choice(holdout_df.loc[(Y_test==0) & (y_predicted==1)].index, 4):
    print('Index:          %i' % i)
    print('Prob. Positive:  %0.3f' % y_probpos[i])
    print('True Label:      %s\n' % holdout_df.label.values[i])
    print(holdout_df.text.values[i], '\n\n')
```

Examine our false negatives

```
In [ ]: np.random.seed(random_state + 0)
for i in np.random.choice(holdout_df.loc[(Y_test==1) & (y_predicted==0)].index, 4):
    print('Index:          %i' % i)
    print('Prob. Positive:  %0.3f' % nbayes.predict_proba(X_test[i])[0][1])
    print('True Label:      %s\n' % holdout_df.label.values[i])
    print(holdout_df.text.values[i], '\n\n')
```

Examine borderline cases

```
In [ ]: np.random.seed(random_state + 0)

border_threshold = 0.05
borderline = ((y_probpos > (0.5 - border_threshold)) &
              (y_probpos < (0.5 + border_threshold)))
print('No. on border:   %i' % sum(borderline), '\n')

for i in np.random.choice(holdout_df.loc[borderline].index.values, 2):
    print('Index:           %i' % i)
    print('Prob. Positive:  %0.3f' % nbayes.predict_proba(X_test[i])[0][1])
    print('True Label:     %s\n' % holdout_df.label.values[i])
    print(holdout_df.text.values[i], '\n\n')
```

In []:

Benchmark Against Other Models

- How will our performance with Naive Bayes live up to a logistic regression or gradient boosting machine?

```
In [ ]: logreg = LogisticRegression(
        random_state=random_state,
        solver='lbfgs'
    )

logreg.fit(X, Y)
report_performance(logreg)
```

```
In [ ]: gbm = GradientBoostingClassifier(
        random_state=random_state,
        max_depth=6,
        n_estimators=100
    )

gbm.fit(X, Y)
report_performance(gbm)
```

In []:

In []:

Take the classifier for a spin! Review a movie.

- Does our naive bayes classifier perform like we'd expect it to?
- How does it handle things like sarcasm? Negation?

To evaluate our classifier, we'll also rely on some additional reviews helpfully compiled by [@AmznMovieRevws](https://twitter.com/AmznMovieRevws) (<https://twitter.com/AmznMovieRevws>)

```
In [ ]: def score_review(example, model):
        test = vectorizer.transform([example])
        print('Words included: %i' % test.toarray().sum())
        print('Prob. Positive: %0.3f' % nbayes.predict_proba(test)[0][1])
        print(example)
```

```
In [ ]: example = '''
        This was a really great movie. One of the best I've seen in a while!
        '''

        score_review(example, nbayes)
```

```
In [ ]: example = '''
        I hated this movie. The plot was so slow and the acting was terrible.
        '''

        score_review(example, nbayes)
```

 alt text


```
In [ ]: example = '''
        False Advertising. Don't be fooled. Although this movie is called "The Rock
        entertainer Dwayne "The Rock" Johnson.
        '''

        score_review(example, nbayes)
```

 alt text


```
In [ ]: example = '''
        Misleading. There are no wolves in this movie.
        '''

        score_review(example, nbayes)
```

 alt text

```
In [ ]: example = '''
        If you were going to see this movie because, like me, you love chocolate and
        the process prepare to go elsewhere. This movie has no chocolate to offer -
        chocolate, nothing about the abuses in the chocolate industry, and not even
        I wish I were one of the dead characters in the movie, as I would have felt
        '''

        score_review(example, nbayes)
```


 alt text

```
In [ ]: example = '''  
Expected a superhero named the tick to bite people and infect them with a s  
Not impressed.  
'''  
  
score_review(example, nbayes)
```

```
In [ ]: example = '''  
There is nothing funny about a tick being a super hero. Ticks cause lyme di  
'''  
  
score_review(example, nbayes)
```

```
In [ ]:
```