Machine Learning Engineer Nanodegree

Capstone Project

Chester Fung
January 21st, 2016

I. Definition

Domain Background

The project of this capstone is on the entertainment film industry. The global film industry shows healthy projections for the coming years, as the global box office revenue is forecast to increase from about 38 billion U.S. dollars in 2016 to nearly 50 billion U.S. dollar in 2020. The U.S. is the third largest film market in the world in terms of tickets sold per year, only behind China and India. More than 1.2 billion movie tickets were sold in the U.S. in 2015. Many websites offer portals for users to give them feedback or reviews of the movies they watch. These reviews include both positive and negative. Being a big movie fan, I visit these review sites often to look for which movies I should watch.

Several papers have been published before on this topic, including
Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). "Learning Word Vectors for Sentiment Analysis." *The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011)*. (link)

The paper and also this project is based on a Kaggle Competition.

Problem Statement

This capstone project is to classify the user reviews from IMDB.

Specifically, it is to classify the sentiment of sentences from the IMDB dataset. Label sentiment of the review; 1 for positive reviews and 0 for negative reviews

We'll then use feature extraction module from scikit-learn to create bag-of-words features.

Metrics

Evaluation Metrics

Evaluation metric that will be used in this capstone project is F1 score

The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

```
F1 = 2 * (precision * recall) / (precision + recall)
```

II. Analysis

Data Exploration

Datasets and Inputs

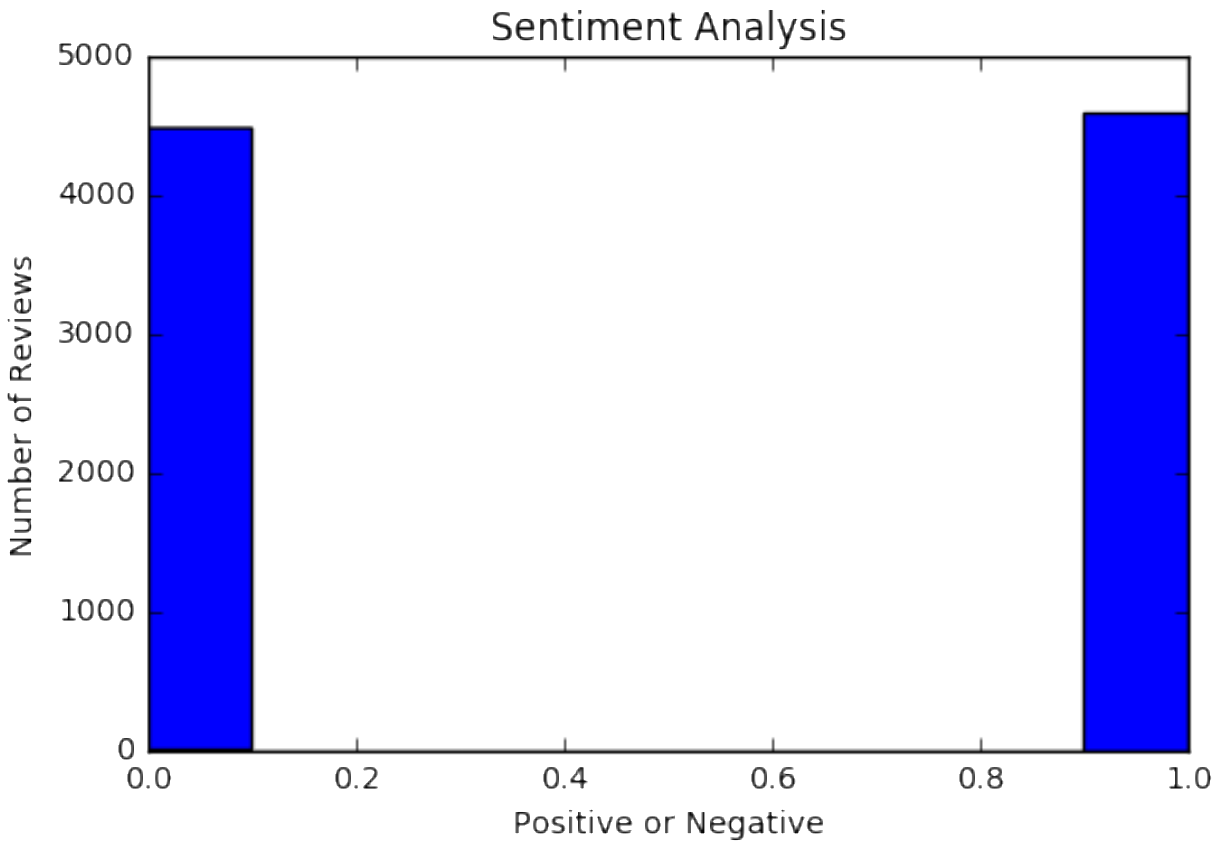
Data set is from IMDB

Size of the data set has 50,000 movie reviews that are labeled. And another 50,000 that are unlabeled.

The labels in the dataset include id, sentiment and review. ID is a numeric value assigned to the review. Sentiment (score of 0 to 10) is an integer representing the sentiment score. Review is a string written by users.

25000 reviews from the labeled test data set will be used for training. Another 25000 will be used for testing.

Exploratory Visualization



Algorithms and Techniques

Sentiment of the review should be labeled as 1 for positive and 0 for negative. IMDB rating < 5 results in a sentiment score of 0, and rating >=7 have a sentiment score of 1

We'll use an approach called "Bag of Words". The Bag of Words model learns a vocabulary from all of the documents, then models each document by counting the number of times each word appears. Scikit-learn will then be used to create features from "Bag of Words" Support Vector Machines and Random Forest will then be used to tackle this classification problem. We'll then try different options in CountVectorizer to tune and optimize the model.

The following algorithms will be used and results will then be compared:

Logistic regression - advantages: Good if problem (target variable) is linearly separable. Robust to noise. No distribution requirement

Naive Bayes classifier - Advantages: fast to train, not sensitive to irrelevant features

weaknesses: assumes independence of features

Reason for choosing: Naive Bayes works well for small training set sizes and it's fast, and our dataset is small

SVM (Support Vector Machine) - SVM works great for linear problems

Disadvantage: Inefficient to train. Therefore, it's not good for problems with many training points

Advantages: high accuracy

Reason for choosing: high accuracy

Random Forest Boosting - Advantage (Random Forest): fits well with uneven data sets with missing variables. Lower classification error rate compared to decision tree. Faster training time compared to SVM

Benchmark

Benchmark Model

This is a Kaggle competition. Winner of the competition achieved an ROC curve score of **0.99259**. The goal of this capstone project is to obtain **0.9 or above**

III. Methodology

Data Preprocessing

We'll first preprocess /clean the data by doing the following steps:

- **Use BeatifulSoup to parse the data**

```
In [9]: from bs4 import BeautifulSoup # use BeautifulSoup to parse the text

In [15]: review1 = BeautifulSoup(train_movie_data['review'][0], "html.parser")
```

- **Remove numbers and punctuations, convert all letters to lower case and separate the phrases into individual words.**

```
: import re
  modified_review1 = re.sub("[^a-zA-Z]", " ", review1.get_text()).lower()
  print modified_review1
```

We'll also remove "stop" words. These are words that do not have much meaning, i.e. 'and', 'is', 'a'.

```
words = modified_review1.split()
```

```
import nltk
#nltk.download()
```

```
from nltk.corpus import stopwords
print stopwords.words("english")
```

```
words = [w for w in words if not w in stopwords.words("english")]
print words
```

Implementation

After removing the 'stop' words, next step is to convert the review into words.

```
def convert_review_to_words(review):

    reviews = BeautifulSoup(review).get_text()

    modified_reviews = re.sub("[^a-zA-Z]", " ", reviews)

    words = modified_reviews.lower().split()

    stops = set(stopwords.words("english"))

    meaningful_words = [w for w in words if not w in stops]

    return("".join(meaningful_words))
```

Create an empty list, clean_train_reviews, and add all the "cleaned" train reviews.

We'll split the data into train and test data. Since we've already created a "clean_train_reviews" list and appended all the "clean" reviews to the list, we'll perform the same step against the test dataset, "test_train_reviews".

Create an empty list, test_train_reviews, add all the “cleaned” reviews to it by using convert_review_to_words function.

-

```
# Read the test data
# train_movie_data = pd.read_csv('labeledTrainData.tsv', header=0, sep='\t')
test = pd.read_csv("testData.tsv", header=0, sep='\t')

# Verify that there are 25,000 rows and 2 columns
print test.shape

# Create an empty list and append the clean reviews one by one
clean_test_reviews = []

print "Cleaning and parsing the test set movie reviews...\n"
for i in xrange(0,len(test["review"])):
    clean_test_reviews.append("".join(convert_review_to_words(test["review"][i])))
```

Now, apply the “Bag-of-words” model. The bag-of-words model is a simplifying prepresentation used in natural language processing and information retrieval. In this model, a text is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity. For example, we have the following two documents,

1.John likes to watch movies. Mary likes movies too.

2.John also like to watch football games.

Based on these two text documents, a list is constructed as follows:

[
“John”,
“likes”,
“to”,
“watch”,
“movies”,
“also”,
“football”,
“games”,
“Mary”,
“too”
]

We can then use the Bag-of-words model as a tool of feature generation. After transforming the text into a “bag of words”, we can calculate the term frequency.

In our case, the following code demonstrates to application of using the “bag-of-words” model.

```
print "Creating the bag of words...\n"
from sklearn.feature_extraction.text import CountVectorizer

# Initialize the "CountVectorizer" object, which is scikit-learn's
# bag of words tool.
vectorizer = CountVectorizer(analyzer = "word",   \
                             tokenizer = None,    \
                             preprocessor = None, \
                             stop_words = None,   \
                             max_features = 5000)

# fit_transform() does two functions: First, it fits the model
# and learns the vocabulary; second, it transforms our training data
# into feature vectors. The input to fit_transform should be a list of
# strings.
train_data_features = vectorizer.fit_transform(clean_train_reviews)

# Numpy arrays are easy to work with, so convert the result to an
# array
train_data_features = train_data_features.toarray()
```

After the train_data_features has been converted to an array, let's look at 1 entry,

```
train_data_features[:1]
```

```
array([[0, 0, 0, ..., 0, 0, 0]])
```

As expected, it is an array containing 0s and 1s.

Now, we'll use the features and use an algorithm to train and fit the training set. Let's start with Random Forest first

```
print "Training the random forest..."
from sklearn.ensemble import RandomForestClassifier

# Initialize a Random Forest classifier with 100 trees
forest = RandomForestClassifier(n_estimators = 100)
print "after forest"
# Fit the forest to the training set, using the bag of words as
# features and the sentiment labels as the response variable
#
# This may take a few minutes to run
forest = forest.fit( train_data_features, train_movie_data["sentiment"] )
#print forest
```

After the Random Forest model is trained, use it to predict on the test dataset

```
# Get a bag of words for the test set, and convert to a numpy array
test_data_features = vectorizer.transform(clean_test_reviews)
test_data_features = test_data_features.toarray()

# Use the random forest to make sentiment label predictions
print "Predicting test labels...\n"
result = forest.predict(test_data_features)
#result = model_LR.predict(test_data_features)
#result = model_NB.predict(test_data_features)
#result = model_SVC.predict(test_data_features)

# Copy the results to a pandas dataframe with an "id" column and
# a "sentiment" column
output = pd.DataFrame( data={"id":test["id"], "sentiment":result} )

# Use pandas to write the comma-separated output file
output.to_csv(os.path.join(os.path.dirname('__file__'), 'data', 'Bag_of_Words_model.csv'), index=False, quoting=3)
print "Wrote results to Bag_of_Words_model.csv"
```

Refer to the “Results” section for score generated by Random Forest.

Next, let’s try on Logistic Regression

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression


model_LR = LogisticRegression()
model_LR.fit(train_data_features, train_movie_data["sentiment"])
```

Pls refer to the “Results” section for score generated by Logistic Regression

Next, we’ll try using Naïve Bayes

```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB

model_NB = GaussianNB()
model_NB.fit(train_data_features, train_movie_data["sentiment"])
```

After training with the train data using Naïve Bayes, let's predict with the test data. Pls refer to the score generated by Naïve Bayes in the "Results" section

Finally, we'll use Support Vector Machine (SVM) model.

```
# Support Vector Machine
from sklearn.svm import SVC
#clf_svm = SVC(gamma=0.001, C=100.)
model_SVC = SVC()
model_SVC.fit(train_data_features, train_movie_data["sentiment"])
```

Once the model finishes using the train data set to fit, we'll predict the result using the test data set. Pls refer to the score generated by SVM in the "Results" section

Refinement

Use GriSearchCV for feature selection:

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
from sklearn.grid_search import GridSearchCV

grid_values = {'C':[30]} # Decide which settings you want for the grid search.

model_LR = GridSearchCV(LogisticRegression(penalty = 'l2', dual = True, random_state = 0),
                        grid_values, scoring = 'roc_auc', cv = 20)

#model_LR.fit(X,y_train) # Fit the model.

#model_LR = LogisticRegression()
model_LR.fit(train_data_features, train_movie_data["sentiment"])
```

IV. Results

The final score is judged on area under the ROC curve. A Receiver Operating Characteristic curve (ROC curve), is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. The results for different models are the following:

Random Forest: 0.50044

Logistic Regression: 0.50044
Naïve Bayes: 0.50044
SVM: 0.50044
LR with GridSearch: 0.50044

Model Evaluation and Validation

Justification

Based on the result, the scores for using different models are still the same, that leads me to believe that there could be something wrong with my code, could be related to training set or test data set parsing.

V. Conclusion

Free-Form Visualization
In this section, you will need to provide some form of visualization that emphasizes an important quality about the project. It is much more free-form, but should reasonably support a significant result or characteristic about the problem that you want to discuss. Questions to ask yourself when writing this section:
- _Have you visualized a relevant or important quality about the problem, dataset, input data, or results?_
- _Is the visualization thoroughly analyzed and discussed?_
- _If a plot is provided, are the axes, title, and datum clearly defined?_

Reflection

One of the difficult/frustrating part of the project is training and predicting results with Random Forest is significantly longer the other algorithms. This could be due to RF using more memory, leading to occasional freeze of my macbook pro.

Improvement

One method that can be tried to improve the score is tf-idf, term frequency –inverse document frequency.

Tf-idf, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in information retrieval and text mining.

Another method to try is to use Neural Networks.