CS 8803 Social Computing Assignment 2

Revant Kumar
GT ID: 903024989
Master's in Computer Science
Georgia Institute of Technology

1 Introduction

For this project, I have implemented two classifiers, namely, Multinomial Naive Bayes Classifier with Laplace Smoothing and Support Vector Machines with Linear Kernel. For the features, I have tried each of the classifiers with the unigram model and bigram model. Finally, I have reported the mean accuracy, precision, recall and F1 score for the classifiers. This project is implemented in Python by using libraries such as Numpy, Scikit-learn, NLTK, csv, and random. The code can be seen at my github: "https://gist.github.com/revantkumar/4bba0255a2f085234c08"

2 Feature Construction

For the feature construction part of the project, I have constructed both the unigram and bi-gram model. Each of the model is applied on both the classifiers, namely, Multinomial Naive Bayes Classifier with Laplace Smoothing and Support Vector Machines with Linear Kernel.

The unigram model is basically called the "bag of words" representation. In this model, each message is represented by a feature vector which is formed by counts of the words found in the vocabulary of the given file. For example, if a message is "Hello Hello I am Revant" and vocabulary of the model is given by ["I", "am", "Hello", "Revant", "Awesome"], then the feature vector to represent the given message will be [1,1,2,1,0]. Unigram Model is chosen because it is easy to implement, accuracy achieved is quite good, plus the words in the vocabulary are independent to each other. By this way, I have represented each of the messages given in the file in terms of their feature vectors.

For the bigram model, the range of n-gram is set to 2, so two consecutive words can be chosen. It is seen that the bigram model performs better than the unigram model because the bigram models take into account the probability of the next word. And thus they perform better than unigram model.

The length of the message was also used as a feature in both models. The intuition behind entering the length of message as a feature is that the cost of sending a text message is the same as long as it is contained below 160 characters, so marketers would prefer to use most of the space available to them as long as it doesn?t exceed the limit.

Note: In order to obtain the features, we need to do some preprocessing on the data such as tokenization and removal of stop-words. I have explained this in detail in the Implementation Section.

3 Description of Classifier

I have implemented two classifiers for this project, namely, Multinomial Naive Bayes Classifier with Laplace Smoothing and Support Vector Machines with Linear Kernel.

- Naive Bayes Classifier is a supervised learning algorithm based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features. Given a class variable y and a dependent feature vector x_1 through x_n, Bayes' theorem states the following relationship:

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) P(x_1, \dots x_n \mid y)}{P(x_1, \dots, x_n)} \tag{1}$$

MultinomialNB implements the naive Bayes algorithm for multinomially distributed data. The distribution is parametrized by vectors $\theta_y = (\theta_{y1}, \dots, \theta_{yn})$ for each class y, where n is the number of features and θ_{yi} is the probability $P(x_i \mid y)$ of feature i appearing in a sample belonging to class y.

The parameters θ_y is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n} \tag{2}$$

where $N_{yi} = \sum_{x \in T} x_i$ is the number of times feature i appears in a sample of class y in the training set T, and $N_y = \sum_{i=1}^{|T|} N_{yi}$ is the total count of all features for class y. The smoothing priors $\alpha \geq 0$ accounts for features not present in the learning samples and prevents zero probabilities in further computations. Setting $\alpha = 1$ is called Laplace smoothing

The speed and simplicity along with high accuracy of this algorithm makes it a desirable classifier for spam detection problems.

- The second classifier that I implemented is Support Vector Machine (SVM) with linear kernel. Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier.

  I used SVMs for the various advantages that they have such as effective in high dimensional spaces, effective in cases where number of dimensions is greater than the number of samples. Also, the algorithm is memory efficient as it uses a subset of training points in the decision function (called support vectors). It is also versatile as different Kernel functions can be specified for the decision function.

4 Evaluation Technique

In order to judge the performance of the classifiers, I have reported the mean accuracy, precision, recall and F1 score of the classifiers. Precision, recall and F1 score is calculated for both the classes ”Ham” and ”Spam” along with the average precision, recall and F1 scores of the classifiers.

Regarding k-fold cross validation, I have used the 10-fold cross validation. Using this method, I split the data set into k parts, hold out one, combine the others and train on them, then validate against the held-out portion. I repeated that process k times (each fold), holding out a different portion each time. Then I averaged the score measured for each fold to get a more accurate estimation of the model's performance. I have chosen k=10 because the value of k=10 was obtained by running the algorithm multiple times and then seeing for which k the results come out the best.

5 Implementation

5.1 Pre-Processing of Data

For pre-processing of data, I used python libraries such as NLTK. The tokenization of the message was done by using “from nltk.tokenize import RegexpTokenizer” and stop-words were removed from the vocabulary by using “from nltk.corpus import stopwords”. Once the data is pre-processed, it is stored in a dictionary in the this format $data = \{"text" : [], "class" : []\}$ where $data["text"]$ holds the pre-processed messages and $data["class"]$ holds the class of the corresponding messages in the same order.

5.2 Extraction of Features

Once the pre-processing of data was accomplished, I extracted the features using "from sklearn.feature_extraction.text import CountVectorize".

In general terms, Extraction of Features means to reduce the mass of unstructured data into some uniform set of features that our algorithm can learn from. For text classification, that can mean word counts. I produced a table of each word mention in the corpus and it's corresponding frequency for each class.

The code to do this using Scikit Learn's feature_extraction module is pretty minimal. I instantiated a CountVectorizer and then called its instance method fit_transform, which does two things: it learns the vocabulary of our corpus and extracts our word count features.

Data needs to be converted to a Numpy array before it can be used by CountVectorizer. This can be easily achieved as numpy.asarray(data['text']).

Also, for the bigram model, we need to mention the ngram range from 1 to 2 as one of the parameters for CountVectorizer. For unigram model, no such parameter needs to be mentioned.

5.3 Choosing the Classifier

I implemented the bayes classifier using MultinomialNB and trained it by calling fit, passing in the feature vector and target vector (the classes that each example belongs to).

```
from sklearn.naive_bayes import MultinomialNB
classifier = MultinomialNB()
targets = numpy.asarray(data['class'])
classifier.fit(counts, targets)
```

For predicting,

```
counts = count_vectorizer.transform(examples)
predictions = classifier.predict(counts)
```

However, this one step at a time approach can be improved by using a construct provided by Scikit Learn called a Pipeline. We can use it to merge the feature extraction and classification into one operation:

```
from sklearn.pipeline import Pipeline
pipeline = Pipeline([('vectorizer', CountVectorizer()), ('classifier', MultinomialNB()) ])
pipeline.fit(numpy.asarray(data['text']), numpy.asarray(data['class']))
pipeline.predict(examples)
```

For SVM classifier, change the pipeline as follows:

```
pipeline = Pipeline([('vectorizer', CountVectorizer()), ('classifier', SVC(kernel='linear')) ])
```

For the bigram model, pipeline looks like following:

```
pipeline = Pipeline([('vectorizer', CountVectorizer(ngram_range=(1, 2))), ('classifier', MultinomialNB()) ])
```

5.4 k-fold cross validation

I used k=10 and split the data set into 10 parts, by holding out one, combining the others and training on them, then validating against the held-out portion. This process is repeated 10 times (each fold), holding out a different portion each time. Then I averaged the score measured for each fold to get a more accurate estimation of the model's performance.

Scikit Learn's KFold can be used to generate k pairs of bitmasks ?vectors of booleans which we can use to select out randomized portions of the dataset for our train and cross-validation sets.

```
from sklearn.cross_validation import KFold
k_fold = KFold(n=len(data), n_folds=6, indices=False)
```

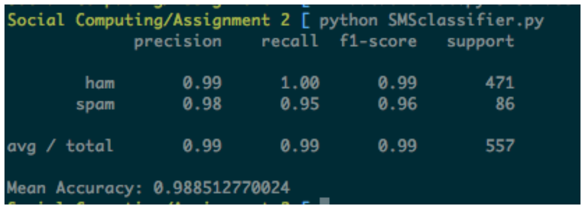
5.5 Performance Evaluation

Scikit Learn models provide a score method, which gives us the mean accuracy of the model on each fold, which we then average together for a mean accuracy on the entire set.

score = pipeline.score(test_text, test_y)

We can also get the precision, recall, F1 score of each class as well as the average ones by using this "metrics.classification_report(test_y, predicted, target_names=['ham', 'spam'])"

Output of this looks like the following:



```
Social Computing/Assignment 2 [ python SMSclassifier.py
             precision    recall  f1-score   support

        ham       0.99      1.00      0.99       471
       spam       0.98      0.95      0.96        86

avg / total       0.99      0.99      0.99       557

Mean Accuracy: 0.988512770024
```

6 Analysis of Results

The following table analyses the mean results obtained:

	Mean Accuracy	Average Precision	Average Recall	Mean F1 Score
MultinomialNB (Unigram model)	0.9879	0.98	0.98	0.98
MultinomialNB (Bigram model)	0.9882	0.99	0.98	0.99
SVM with Linear Kernel (Unigram model)	0.9768	0.96	0.96	0.96
SVM with Linear Kernel (Bigram model)	0.9774	0.96	0.97	0.97

The following shows the results with respect to "Ham" Class:

Ham	Ham Precision	Ham Recall	Ham F1 Score
MultinomialNB (Unigram model)	0.98	1.00	0.98
MultinomialNB (Bigram model)	0.99	1.00	0.99
SVM with Linear Kernel (Unigram model)	0.97	1.00	0.98
SVM with Linear Kernel (Bigram model)	0.98	1.00	0.99

The following shows the results with respect to "Spam" Class:

Spam	Spam Precision	SpamRecall	Spam F1 Score
MultinomialNB (Unigram model)	1.00	0.96	0.98
MultinomialNB (Bigram model)	0.99	0.96	0.97
SVM with Linear Kernel (Unigram model)	0.94	0.96	0.97
SVM with Linear Kernel (Bigram model)	0.95	0.91	0.95

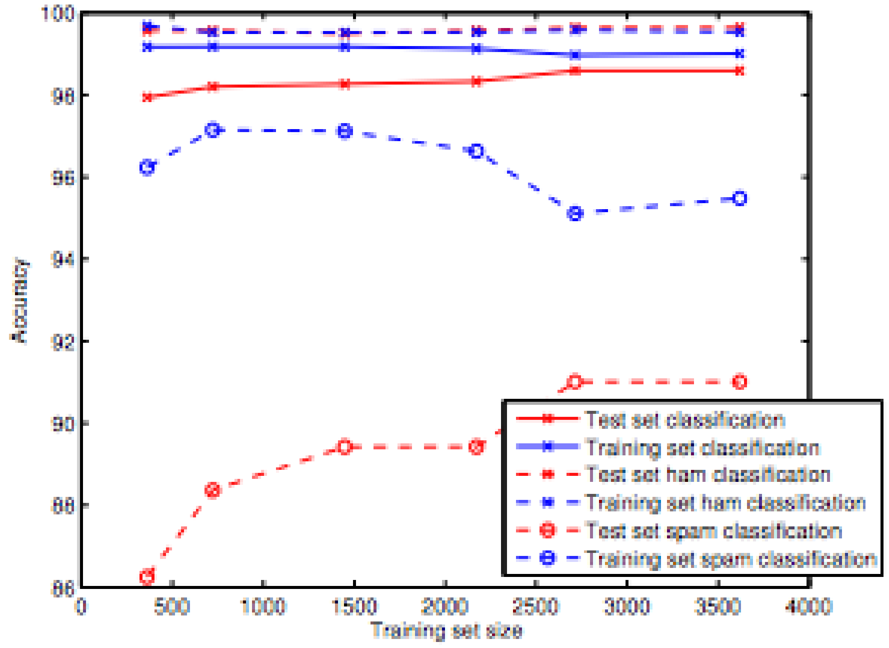
Learning curve for multinomial NB algorithm applied to final features (Figure 1 on next page):

Learning curve for SVM algorithm applied to final features (Figure 2 on next page):

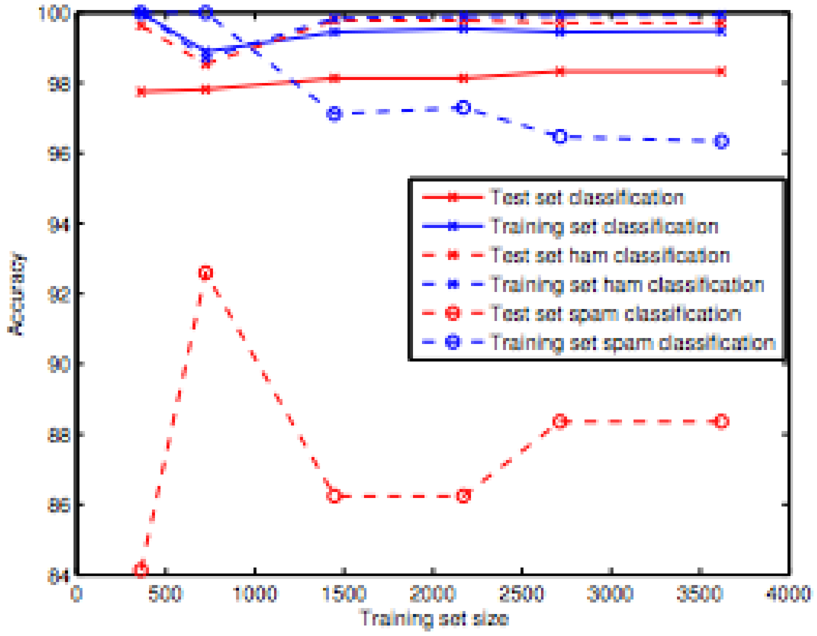
So, we observe that the both the classifiers almost give the same result. However, **the best classifier among is MultinomialNB (Bigram model) as it has the highest mean accuracy.**

7 References

- Houshmand Shirani-Mehr: SMS Spam Detection using Machine Learning Approach
- Scikit-Learn
- https://gist.github.com/zacstewart/5978000
- NLTK



100
98
96
94
92
90
88
86
Accuracy
0
500
1000
1500
2000
2500
3000
3500
4000
Training set size
Test set classification
Training set classification
Test set ham classification
Training set ham classification
Test set spam classification
Training set spam classification



100
98
96
94
92
90
88
86
84
Accuracy
0
500
1000
1500
2000
2500
3000
3500
4000
Training set size
Test set classification
Training set classification
Test set ham classification
Training set ham classification
Test set spam classification
Training set spam classification