Carl Mueller
MP1 Report
CS165A - Winter 2016

Introduction

My Naive Bayes Classifier program implements a multinomial Naive Bayes Classification algorithm and uses a number of string processing techniques and feature selection technique in an attempt to maximize the number of useful features and minimize the number of useless and artifactual features.

To run: Inside MP1 directory:
$ make clean; $ make; $ java NaiveBayesClassifier training.txt testing.txt

Architecture

My program consists for five Java packages: document, feature, model, naivebayes, and util. I have a unittest package for my own use but does not contribute to the function of the program in anyway. For the purposes of making a simple Makefile, the NaiveBayesClassifier class that contains main() is in the default package.

document

The document package consists of two sets of classes devoted to processing .txt files where each set of classes processes the training.txt and testing.txt files. The AbstractDocumentProcessor is an abstract class used as the base class to define both the TrainingDocumentProcessor and TestDocumentProcessor. These two classes are the core classes that import and process the .txt files for both the training.txt and testing.txt files. These classes utilize a StringProcessor object instantiated from the util package to perform the core string pre-processing functions on each movie review.

For each movie review, or document, the DocumentProcessors create a TrainingDocument or TestDocument object, depending on which .txt file is being processed. These objects are aggregated in TrainingDocumentFiler and TestDocumentFiler objects. Each DocumentProcessor ultimately returns DocumentFilers (the TrainingDocumentProcessor returns an array of DocumentFilers for each classification category). The TrainingDocumentFiler objects are passed into a FrequencyFeatureSelector object whereas the TestDocumentFiler is passed into a Classifier object.

feature

The feature package contains an abstract class called FeatureSelector which serves as the base class for FrequencyFeatureSelector and MutualInformationSelector classes. The FrequencyFeatureSelector class takes in TrainingDocumentFiler array and returns a String array of selected features. It chooses features based on hard cutoffs of frequencies for very common words and bigrams and very rare words and bigrams. FrequencyFeatureSelector returns an array FeatureStatistic objects via a public getter method. The FeatureStatistics class contains information about document counts for each category for the given feature. This array is passed into an instance of the MutualInformationSelector class. The MutualInformationSelector chooses features based on mutual

information and a chosen cutoff of .00036. MutualInformationSelector returns an array of the chosen features.

model

The model package contains two classes, ClassificationModel and ModelBuilder. ModelBuilder takes in the features selected from the FeatureSelector objects. ModelBuilder is the workhorse class that populates the variables of a ClassificationModel objects and then returns this object. The ClassificationModel contains all of the prior probabilities of the chosen features.

naivebayes

The naivebayes package contains one class, Classifier. It takes in a ClassificationModel object and uses the prior probabilities contained with the ClassificaitonModel object to classify each TestDocument object, contained in an arrayList of TestDocuments, into the positive or negative movie review category. It also provides a count of the accuracy by comparing it's classified category and the actual category and keeping a count of those documents correctly classified.

util

The util package contains four classes. One is a implementation of a binary tree node and binary search tree I have written previously in which I stored my stop words. I could have used a JRE System Library hashMap or tree but decided to reuse my old code for further practice (I am not a CS major and so any chance I get to continue building programming/implementation skills, I take). The hashUtilities class contains a number of hashMap oriented methods for manipulating JRE System Library hashMaps. The StringProcessor class has a number of methods used by the DocumentProcessors for preprocessing.

Preprocessing

My program preprocesses both test documents and training documents by doing the following:

1. Converts each document to lowercase
2. Removes labels (0 and 1)
3. Removes all non-alphanumeric characters (
 and other erroneous characters)
4. Parses for negations (not good → not_good, isn't good → isnt_good)
5. Removes stopwords (see stopwords.txt)
6. Creates bigrams (ignores bigrams that contain two stopwords i.e. "in the" would not be include)

Model Building

My model is built using the class ModelBuilder and the information is stored in a ClassificationModel object. The program implements a multinomial Naive Bayes classification algorithm, meaning that each feature's probability is calculated by the frequency of the given feature over the total number of features (including their counts) of the classification category. The set of feature used are based on a frequency filter to remove very common and very rare words and bigrams and then each of those features has their mutual information calculated. Test Documents are represented as hashMaps where only the chosen features are selected from the test document string and put into the hashMap. These are my "feature maps" for each test document.

Results

Here are my results:
3 seconds (training)
2 seconds (labeling)
.914 (training)
.844 (testing)

My accuracy is quite decent for a Naive Bayes binary classifier. I believe that with an increased number training documents the accuracy could increase a few percentage points.

Since my features are based on mutual information, and there are only two classes, then each feature will have the same amount of information for each class (worst likely signifies negative sentiment at the same time it signifies *not* positive sentiment). This means the chosen feature is equally important in for each class.

Top ten features:
Feature: worst; MI: .04411263915
Feature: bad; MI: .03810254486
Feature: great; MI: .02102649055
Feature: stupid; MI: .01973541685
Feature: awful; MI: .01906415310
Feature: wonderful; MI: .01842665398
Feature: terrible; MI: .01724158415
Feature: best; MI: .01710694451
Feature: worse; MI: .01445145022
Feature: boring; MI: .01406742389

Challenges

The biggest challenge I faced was implementing the Mutual Information algorithm. I used the algorithm provided by Stanford NLP: http://nlp.stanford.edu/IR-book/html/htmledition/mutual-information-1.html . I had to create another class: FeatureStatistics in order to keep the document count of each chosen feature in the FrequencyFeatureSelector. Otherwise if I calculated these values for each feature inside MutualInformationSelector without this FeatureStatistics class, my run time became an exponential affair. An early issue I faced was using Scanner to import the text files. BOM characters presented problems and so I switched to BufferedReader and it solved any I/O issues I had.

Weakness

The major weakness in my implementation is too much coupling and too much coding. I am not a CS major by training and so object oriented design patterns are still unfamiliar. I attempted to have each class package together a data type that gets passed along to the next class in the pipeline/process. My design is also very monolithic in that it can only do binary classification for movie sentiment analysis.

As a private project, I plan on trying to make my program more loosely coupled and more generic so that it may process multiple classes and provide greater code legibility and design.