CS909 Coursework - Week 10

Sebastian Marson - 1018677

May 19, 2014

1 Introduction

The aim of this coursework was to take the unfiltered Reuters.sgm files, perform necessary pre-processing and feature selection upon the documents, then attempt to build a classifier to classify new examples of data in this format.

2 Pre-Processing and Feature Selection

The first stage was reading in the information from the sgm files into memory, and storing them in such a way that they could be easily manipulated by the program for pre-processing. Following suggestions from the data mining lectures, as well as further examples from data mining literature, several stages of pre-processing were applied after the document was read in.

The documents were read in using the library *BeautifulSoup*. *BeautifulSoup* allows the program to create a soup object from the document, creating a navigable tree split up using the tags in the document. Each file was read in one at a time, pre-processed and then added to one larger object to hold all of the information for further feature selection.

Several different forms of pre-processing and Feature selection were performed on the data as it was read in. Different sections of the document had different formatting done to them as was necessary, the order may have changed for some of them as well, as a different order is an important factor in the pre-processing stage. Some of the pre-processing stages were handled using natural language processing libraries that already exist in python, the one used at this point was the nltk library, for POS tagging and lemmaisation.The following processes were done:

1. **Remove Sections** - Some tags were deemed unnecessary in the brief, namely the DATE, PLACES, PEOPLE, ORGS, EXCHANGES, COMPANIES and UNKNOWN tags.
2. **Remove Unused Documents** - Documents will later be sorted into whether they are intended for training or testing purposes during the classification stage. This is done using the LEWISSPLIT tag, but some of the documents lack any information there, so these can be pruned out of the final data set entirely.
3. **Remove Articles** - Some articles have a body section that is blank. These articles are removed from the data set because they don't provide any useful information to the classifier. These articles have no data that can be classified.

4. **Process Punctuation** - This is the first process that occurs after the information has been read in. All of the different sections go through this process. All punctuation marks are removed from the text, as well as capital letters. This section is because punctuation would only interfere with later stages of breaking the words up into unigrams.

5. **POS Tagging** - This process analyses a sentence, then tags each word in a string with a 'Part of Speech' tag. This part of speech tag denotes what purpose the word has in a sentence, such as nouns, verbs, etc. This is an important part of the feature selection process because it adds more information for the classifiers to use to increase accuracy. When humans read a story they can tell the semantic differences between words that are otherwise the same, based on where they are in the sentence. By adding the POS tags it means the classifier can also use this information that a human reader would have.

6. **Lemmatization** - Where words such as 'cars' or 'excited' are reduced to simpler forms like 'car' or 'excite' respectively. Pluralisation or different conjugations of the same word can hold the same meaning, so this step can help remove noise in textual data.

7. **Remove Words** - Some words, known as stop words, are extremely common and appear often enough in all of the different possible topics that they provide no additional data for sorting and categorising. These words can therefore be removed. Examples of these words are: and, the, it, etc.

Each different type of information in the document may be handled differently to the other tags, although there is consistency between all of the different articles in how a particular tag is handled. For instance, the topic tag has none of its information changed in any way. Because this is the feature that we are trying to classify, it might lead to poor results if the topic is processed and they are changed inconsistently, or changed with a loss of information. The ID and the lewissplit tag data are also extracted and not changed. The ID and lewissplit data are already very concise, they don't have irregular or noisy data, so they can simply be copied over into the new refined article.

Titles are the next tag to be refined by the pre-processor. The title tags information has the same level of pre-processing applied to it as the body tags information. The title first has all the punctuation removed, then POS tagging begins. The string is split up into individual words, and each word is passed through the nltk.POS tagger. These words are then passed into a new list as a tuple, where the first index holds the word, and the second holds the tag for the word, based on its position and meaning in a sentence. After being tagged, but before these tuples are merged for the final list, the word will be lemmatised.

Lemmatisation occurs using the nltk.lemmatizer package. The first part of the tuple, the word, is passed in on its own, then run through the lemmatizer. The resultant word, which has all inflections removed and is known as the lexeme of the original word, will then be placed back in the tuple with its original forms POS tag. Finally this list of tuples will be turned back into one long string, so that its the same format as the original text read in.

The final pre-processing step that is applied to the title of an article is the removal of certain stop words. These are words that are short and dont add any meaning to a sentence. They are extremely common and do not appear often enough in any one particular type of classification for it to make any kind of statistical difference. An example of words like this are: and, the, or it.

These are all removed, along with their POS tag.

The final output of the pre-processing for each document looks like the following:

```
<story>
<ID>3</ID>
<LEWISSPLIT>TRAIN</LEWISSPLIT>
<topics>money-fx</topics>
<title>  texas-NNS commerce-VBP bancshares-NNS tcb-VBP file-NNS
   plan-NN
</title>
<dateline>  houston-NN feb-NN NUMBER-CD</dateline>
<body>  texas-NNS commerce-VBP bancshares-NNS inc-NNS texas-VBP
   commerce-NN bank-houston-NNP say-VBD file-VBD application-NN
   with-IN comptroller-NN the-DT currency-NN  effort-NN create-VB
   largest-JJS bank-VBG network-NN  harris-NN county-NN bank-NN
   say-VBD network-NN would-MD link-VB NUMBER-CD bank-NNS have-VBG
   NUMBER-CD billion-CD dlrs-NNS  asset-NNS NUMBER-CD billion-CD
   dlrs-NNS  deposit-NNS
</body>
</story>
```

After all of these steps had been carried out, each document consisted of these word/POS tag pairs. Further filtering to remove any words that were used less than 4 times was implemented, because these words, out of the approximately 40,000 different types there were originally, would be so statistically insignificant they may reduce the accuracy of the final model through overfitting. After this threshold was applied to the program, there were only approximately 10,000 different word/POS tag pairs left.

3 Supervised Classification

The data from the reuters documents, after being preprocessed, was now ready for classification. After being read in by BeautifulSoup from the pre-processed.sgm file, the data needs to be split into either training data or test data. The data already contains the <lewissplit> tag, which indicates which of the two sets the data belongs in. There are three possible values this data might take: TRAIN, TEST, or NOT-USED. The first two designate which of the two sets the item belongs to while the third means that this story shouldnt be placed into either set.

K-cross fold validation is used so that models can be created and tested with a variety of random subsets of data, to reduce the chance of the data being over fitted and reducing overall accuracy. 10 folds are used by the program, and the fold that achieved the greatest accuracy is saved. This saved model will then be used to test its accuracy on the much larger just test data set, to further gauge its accuracy.

The following classifiers were used in this process:

- **Naive Bayes** - This classifier is a popular probability supervised learning classifier. It is particularly often used for text classification problems.

- **SVM** - A multiclass variant of the support vector machine classifier was used in this project. A normal SVM classifier can only handle two different classes, but this project required 10.
- **Random Tree** - This is a rule based classifier that constructs decision trees to use for classification.

ScikitLean libraries were used for the classifiers. Below is an example of how the classifiers were run. The gaussian naive bayes library was used, which didn't require any special parameters to run properly. The linearSVC model was used for the SVM classifier, because this allowed multiple classes, rather than the two traditionally allowed by a SVM. The RandomForestClassifier was used for the Random Forest section, the only parameter this required was the number of classes that exist, which is 10 for this limited subsection of the training data used. In the appendix the code listed at point 1 is an example of the naive bayes section, although the other two classifiers were run almost identically, only where the model was called did it change.

Micro and macro statistics are calculated for the most accurate models, when used on the test data, for precisions, recall and f-measure. These can then be used later in the evaluation section to calculate the 95% confidence interval, and allow more detailed comparison of the models.

4 Evaluation

4.1 Supervised Classification

The following tables are the results from classification using three different models, which used 10-cross fold verification to find the best model and then classify the test data.

Table 1: Naive Bayes Results.

Naive Bayes				
1				
	precision	recall	f1-score	support
corn	0.63	0.63	0.63	38
crude	0.9	0.91	0.91	203
earn	0.89	0.76	0.82	42
grain	0.05	0.33	0.08	6
interest	0.81	0.39	0.53	66
money	0.5	0.33	0.4	6
ship	0.57	0.67	0.62	12
avg / total	0.83	0.75	0.77	373
2				
	precision	recall	f1-score	support
corn	0.6	0.75	0.67	24
crude	0.98	0.92	0.95	256
earn	0.9	0.75	0.82	24
grain	0.18	0.6	0.27	10
interest	0.76	0.76	0.76	34

money	1	0.4	0.57	10
ship	0.8	0.57	0.67	14
avg / total	0.91	0.85	0.87	372
3				
	precision	recall	f1-score	support
corn	0.43	0.33	0.38	18
crude	0.96	0.92	0.94	268
earn	0.85	0.81	0.83	42
grain	0	0	0	2
interest	0.57	0.44	0.5	18
money	0	0	0	0
ship	0.56	0.45	0.5	22
trade	0	0	0	2
avg / total	0.87	0.82	0.84	372
4				
	precision	recall	f1-score	support
corn	0.74	0.64	0.68	44
crude	0.95	0.9	0.92	199
earn	0.86	0.79	0.82	47
grain	0	0	0	8
interest	0.58	0.58	0.58	38
money	0.67	0.67	0.67	6
ship	0.67	0.53	0.59	30
avg / total	0.83	0.77	0.8	372
5				
	precision	recall	f1-score	support
corn	0.44	0.78	0.56	18
crude	0.98	0.89	0.93	253
earn	0.85	0.74	0.79	31
grain	0.05	0.5	0.09	4
interest	0.64	0.47	0.55	38
money	1	0.75	0.86	8
ship	0.5	0.2	0.29	20
avg / total	0.87	0.78	0.82	372
6				
	precision	recall	f1-score	support
corn	0.67	0.8	0.73	40
crude	0.89	0.9	0.89	159
earn	0.94	0.88	0.91	34
grain	0.32	0.37	0.34	38
interest	0.75	0.67	0.71	61
money	1	0.53	0.7	30
ship	0.38	0.6	0.46	10
avg / total	0.78	0.76	0.76	372
7				
	precision	recall	f1-score	support
corn	0.68	0.76	0.72	34

crude	0.88	0.84	0.86	179
earn	0.73	0.8	0.76	40
grain	0.22	0.29	0.25	28
interest	0.66	0.59	0.62	59
money	0.75	0.33	0.46	18
ship	0.27	0.43	0.33	14
avg / total	0.73	0.71	0.72	372
8				
	precision	recall	f1-score	support
crude	0	0	0	2
earn	0.71	0.88	0.79	34
grain	0.99	0.92	0.95	242
interest	0.73	0.79	0.76	28
money	0.17	0.67	0.27	6
ship	0.62	0.89	0.73	18
trade	1	0.43	0.6	14
wheat	0.7	0.5	0.58	28
avg / total	0.89	0.84	0.86	372
9				
	precision	recall	f1-score	support
corn	0.83	0.66	0.73	58
crude	0.97	0.9	0.93	193
earn	0.86	0.71	0.77	34
grain	0.21	0.88	0.34	16
interest	0.92	0.56	0.7	41
money	1	0.5	0.67	8
ship	0.5	0.55	0.52	22
avg / total	0.87	0.77	0.8	372
10				
	precision	recall	f1-score	support
corn	0.74	0.64	0.68	44
crude	0.94	0.9	0.92	194
earn	0.91	0.84	0.87	25
grain	0.11	0.29	0.16	14
interest	0.84	0.78	0.81	55
money	0.62	0.56	0.59	18
ship	0.64	0.64	0.64	22
avg / total	0.83	0.79	0.81	372

Table 2: SVM Results.

SVM				
1				
	precision	recall	f1-score	support
corn	1	0.89	0.94	38
crude	0.86	1	0.92	203

earn	1	0.95	0.98	42
grain	1	1	1	6
interest	1	0.58	0.73	66
money	1	1	1	6
ship	1	1	1	12
avg / total	0.92	0.91	0.9	373
2				
	precision	recall	f1-score	support
corn	1	0.92	0.96	24
crude	0.94	1	0.97	256
earn	1	0.83	0.91	24
grain	1	0.2	0.33	10
interest	0.94	0.88	0.91	34
money	1	1	1	10
ship	1	1	1	14
avg / total	0.95	0.95	0.94	372
3				
	precision	recall	f1-score	support
corn	1	0.67	0.8	18
crude	0.96	1	0.98	268
earn	1	0.95	0.98	42
grain	1	1	1	2
interest	1	0.89	0.94	18
money	1	0.91	0.95	22
ship	1	1	1	2
avg / total	0.97	0.97	0.97	372
4				
	precision	recall	f1-score	support
corn	1	0.82	0.9	44
crude	0.91	1	0.95	199
earn	1	0.87	0.93	47
grain	1	1	1	8
interest	1	0.89	0.94	38
money	1	1	1	6
ship	1	0.93	0.97	30
avg / total	0.95	0.95	0.95	372
5				
	precision	recall	f1-score	support
corn	1	1	1	18
crude	0.91	1	0.95	253
earn	1	0.87	0.93	31
grain	1	0.5	0.67	4
interest	1	0.63	0.77	38
money	1	1	1	8
ship	1	0.8	0.89	20
avg / total	0.94	0.94	0.93	372
6				

	precision	recall	f1-score	support
corn	1	0.9	0.95	40
crude	0.87	1	0.93	159
earn	1	1	1	34
grain	1	0.79	0.88	38
interest	1	0.8	0.89	61
money	1	1	1	30
ship	1	1	1	10
avg / total	0.94	0.94	0.93	372
7				
	precision	recall	f1-score	support
corn	1	0.94	0.97	34
crude	0.9	1	0.95	179
earn	1	0.85	0.92	40
grain	1	0.79	0.88	28
interest	1	0.93	0.96	59
money	1	1	1	18
ship	1	0.86	0.92	14
avg / total	0.95	0.95	0.95	372
8				
	precision	recall	f1-score	support
corn	1	1	1	2
crude	1	0.88	0.94	34
earn	0.96	1	0.98	242
grain	1	0.86	0.92	28
interest	1	1	1	6
money	1	1	1	18
ship	1	1	1	14
trade	1	0.93	0.96	28
avg / total	0.97	0.97	0.97	372
9				
	precision	recall	f1-score	support
corn	1	0.79	0.88	58
crude	0.78	1	0.88	193
earn	1	0.82	0.9	34
grain	1	0.12	0.22	16
interest	1	0.61	0.76	41
money	1	1	1	8
ship	1	0.73	0.84	22
avg / total	0.89	0.85	0.84	372
10				
	precision	recall	f1-score	support
corn	1	0.91	0.95	44
crude	0.91	1	0.95	194
earn	1	0.92	0.96	25
grain	1	0.71	0.83	14
interest	1	0.85	0.92	55

money	1	0.89	0.94	18
ship	1	1	1	22
avg / total	0.95	0.95	0.95	372

Table 3: Random Forest Results.

Random Forest				
1				
	precision	recall	f1-score	support
corn	1	0.89	0.94	38
crude	0.86	1	0.92	203
earn	1	0.95	0.98	42
grain	1	1	1	6
interest	1	0.58	0.73	66
money	1	1	1	6
ship	1	1	1	12
avg / total	0.92	0.91	0.9	373
2				
	precision	recall	f1-score	support
corn	1	0.92	0.96	24
crude	0.94	1	0.97	256
earn	1	0.83	0.91	24
grain	1	0.2	0.33	10
interest	0.94	0.88	0.91	34
money	1	1	1	10
ship	1	1	1	14
avg / total	0.95	0.95	0.94	372
3				
	precision	recall	f1-score	support
corn	1	0.67	0.8	18
crude	0.96	1	0.98	268
earn	1	0.95	0.98	42
grain	1	1	1	2
interest	1	0.89	0.94	18
money	1	0.91	0.95	22
ship	1	1	1	2
avg / total	0.97	0.97	0.97	372
4				
	precision	recall	f1-score	support
corn	1	0.82	0.9	44
crude	0.91	1	0.95	199
earn	1	0.87	0.93	47
grain	1	0.75	0.86	8
interest	0.94	0.89	0.92	38
money	1	1	1	6

ship	1	0.93	0.97	30
avg / total	0.95	0.94	0.94	372
5				
	precision	recall	f1-score	support
corn	1	1	1	18
crude	0.91	1	0.95	253
earn	1	0.87	0.93	31
grain	1	0.5	0.67	4
interest	1	0.63	0.77	38
money	1	1	1	8
ship	1	0.8	0.89	20
avg / total	0.94	0.94	0.93	372
6				
	precision	recall	f1-score	support
corn	1	0.9	0.95	40
crude	0.87	1	0.93	159
earn	1	1	1	34
grain	1	0.79	0.88	38
interest	1	0.8	0.89	61
money	1	1	1	30
ship	1	1	1	10
avg / total	0.94	0.94	0.93	372
7				
	precision	recall	f1-score	support
corn	1	0.94	0.97	34
crude	0.9	1	0.95	179
earn	1	0.85	0.92	40
grain	1	0.79	0.88	28
interest	1	0.93	0.96	59
money	1	1	1	18
ship	1	0.86	0.92	14
avg / total	0.95	0.95	0.95	372
8				
	precision	recall	f1-score	support
corn	1	1	1	2
crude	1	0.88	0.94	34
earn	0.96	1	0.98	242
grain	1	0.86	0.92	28
interest	1	1	1	6
money	1	1	1	18
ship	1	1	1	14
trade	1	0.93	0.96	28
avg / total	0.97	0.97	0.97	372
9				
	precision	recall	f1-score	support
corn	1	0.79	0.88	58
crude	0.78	1	0.88	193

earn	1	0.82	0.9	34
grain	1	0.12	0.22	16
interest	1	0.61	0.76	41
money	1	1	1	8
ship	1	0.73	0.84	22
avg / total	0.89	0.85	0.84	372
10				
	precision	recall	f1-score	support
corn	1	0.91	0.95	44
crude	0.91	1	0.95	194
earn	1	0.92	0.96	25
grain	1	0.71	0.83	14
interest	1	0.85	0.92	55
money	1	0.89	0.94	18
ship	1	1	1	22
avg / total	0.95	0.95	0.95	372

As can be seen the SVM - *Support Vector Machine* - performed the best overall both in the 10-cross fold validation steps and with the actual testing data. What is interesting to note is that some of the classes were so uncommon they wouldn't appear in every test, because of the nature of the k=cross fold validation only taking a sample of the overall test data. Trade and wheat were these relatively uncommon classes in the data.

Below are the best models of each classifier tested on the testing data, for easy comparison:

Naive Bayes				
	precision	recall	f1-score	support
corn	0.7	0.73	0.72	360
crude	0.95	0.93	0.94	2176
earn	0.8	0.75	0.78	296
grain	0.17	0.34	0.22	172
interest	0.7	0.53	0.6	424
money	0.71	0.28	0.4	85
ship	0.54	0.62	0.58	208
avg / total	0.82	0.79	0.8	3721
SVM				
	precision	recall	f1-score	support
corn	0.88	0.82	0.85	360
crude	0.89	1	0.94	2176
earn	0.99	0.84	0.91	296
grain	0.9	0.41	0.56	172
interest	0.84	0.74	0.78	424
money	0.96	0.51	0.66	85
ship	0.9	0.89	0.9	208
avg / total	0.9	0.9	0.89	3721

Random Forest				
	precision	recall	f1-score	support
corn	0.87	0.71	0.78	360
crude	0.87	1	0.93	2176
earn	0.81	0.79	0.8	296
grain	0.71	0.26	0.38	172
interest	0.72	0.64	0.68	424
money	0.71	0.34	0.46	85
ship	0.78	0.58	0.66	208
avg / total	0.83	0.84	0.82	3721

Across all of the folds, a table comparing the three different classifiers can be made, to give a calculated example of the differences between them, and which ones present better information:

Fold	Naive Bayes	SVM	Random Forest	NB-SVM	NV-RT	SVM-RT
1	0.748	0.909	0.909	-0.161	-0.161	0
2	0.849	0.952	0.952	-0.103	-0.103	0
3	0.817	0.968	0.968	-0.151	-0.151	0
4	0.769	0.946	0.942	-0.177	-0.173	0.004
5	0.785	0.935	0.935	-0.15	-0.15	0
6	0.758	0.935	0.93	-0.177	-0.172	0.005
7	0.71	0.946	0.946	-0.236	-0.236	0
8	0.844	0.973	0.973	-0.129	-0.129	0
9	0.774	0.855	0.855	-0.081	-0.081	0
10	0.79	0.946	0.946	-0.156	-0.156	0
Average	0.784	0.937	0.936	-0.152	-0.151	0.001
STD	0.043	0.034	0.034	0.043	0.042	0.002
t-Test	N/A	N/A	N/A	-11.178	-11.369	1.581
Confidence	0.026	0.013	0.015	N/A	N/A	N/A

The micro and macro values for all of the classifiers precision, recall, and F1-score were all also calculated for the three different classifiers, the results of which are shown in this table:

	Naive Bayes	SVM	Random Forest
Accuracy	0.793	0.895	0.842
Precision: Micro	0.793	0.895	0.842
Precision: Macro	0.654	0.909	0.807
Recall: Micro	0.793	0.895	0.842
Recall: Macro	0.598	0.744	0.608
F1: Micro	0.793	0.895	0.842
F1: Macro	0.607	0.802	0.66

In the previous table, and the ones preceding it, we can clearly see that the SVM classifier outperforms the others in all of the measured metrics. Normally Naive Bayes classifiers are used when it comes to text classification, because of their very good accuracy, and although the naive bayes classifier did perform well in these experiments, it was outdone by the multi-class SVM classifier. Hence, that is why the SVM model should be recommended as the best one to use.

4.2 Clusters

Cluster analysis allows unsupervised learning to take place on the training and test data, in order to build models for classification. Three different clustering algorithms were used:

- **DBSCAN** - this classifier was chosen because its properties have made it one of the most common types of clustering algorithm used. It didn't require the number of clusters to be defined before beginning, it can find arbitrarily shaped clusters, and it can find clusters that are surrounded by other clusters, which is a property some other clustering algorithms lack.
- **KMeans** - this is another popular clustering algorithm typically used in the academic world, and for solving real world problems.
- **Ward** - again is a popular and generally accurate clustering algorithm. This type of clustering algorithm is also very good at detecting large numbers of clusters in a given space. In this case there should be 10, so ward seemed like a good choice of clustering algorithm.

In the appendix section in the second position is a section of code showing how the clusters were called and their parameters. DBSCAN required two parameters, eps denoted the maximum distance between two samples before they wouldn't be considered in the same neighborhood as each other. Min samples was also used, this denoted the minimum number of samples that would have to exist in a neighborhood before it could be considered a 'core point'. These numbers, of 0.3 and 10 respectively were obtained through trial and error.

KMeans had only one parameter, which was the number of clusters. Because KMeans requires foresight into the number of clusters that it should be attempting to find. There were approximately 81 classes in the full data set that hadn't been filtered to the 10 most popular classes. Ward also had this parameter to input, which was 81.

The next table shows the results from the three different clustering algorithms when applied to the training and testing data:

	DBSCAN	KMeans	Ward
Homogeneity	0.009	0.141	0.141
Completeness	0.057	0.075	0.076
V-Measure	0.015	0.098	0.098
Adj Random Index	0.017	0.009	0.009
Adj Mutual Information	0.008	0.056	0.056
Silhouette Coefficient	0.603	0.583	0.566

As is evident from the results, the DBSCAN method yielded the most accurate results, with Kmeans being the second most accurate and then Ward. The silhouette coefficient shows how 'well-clustered' points in any cluster are, by demonstrating how large the average distance is between other points in the same cluster and points in neighboring clusters. DBSCAN has a distinct lead in silhouette score over the other two clusters.

A copy of the results along with the source code for the project can be found here on a repository on Github:

https://github.com/SebCompski/DataMining

5 Appendix

1.

```
print "Naive Bayes:"
gnb = GaussianNB()
print "Doing k-fold cross fold validation stuff..."
currentFold = 1
bestAccuracy = 0.0
kf = KFold(len(testClass), 10)

#For each of the folds we want, make a seperate set of training data
for trainkf, testkf in kf:
    print ("Doing fold: %d" % currentFold)
    #trainkf and testkf are a list of indices, so you need to pull
        them out
    splitTrainStory = [trainStory[i] for i in trainkf]
    splitTrainClass = [trainClass[i] for i in trainkf]

    splitTestStory = [trainStory[i] for i in testkf]
    splitTestClass = [trainClass[i] for i in testkf]

    #Make the model
    model = gnb.fit(splitTrainStory, splitTrainClass)

    #Create a list of classes
    labels = model.classes_

    #Do the predictions
    prediction = model.predict(splitTestStory)

    #Output the statistics for this classifier to a text file
    output_csv(splitTestClass, prediction,labels,currentFold,
        "Naive Bayes")
    output_stats(splitTestClass,prediction,labels,currentFold,
        "Naive Bayes")

    if (bestAccuracy < float(metrics.accuracy_score(splitTestClass,
            prediction))):
        bestAccuracy = float(metrics.accuracy_score
            (splitTestClass,prediction))
        bestModel = model
        bestPrediction = prediction
        bestTestClass = splitTestClass

    print ("This model accuracy: %0.3f" % float(metrics.
        accuracy_score(splitTestClass, prediction)))
    print ("Current best Acc: %0.3f" % bestAccuracy)

    currentFold = currentFold + 1

print
print ("Best NB Accuracy: %0.3f" % bestAccuracy)
print "Running against the test data now..."

#For the best data,
prediction = model.predict(testStory)
print_metrics(testClass, prediction)
output_csv(testClass, prediction,labels,"Best","Naive Bayes")
output_stats(testClass,prediction,labels,"Best","Naive Bayes")
```

2.
```
        print
print "DBSCAN:"
classifier = DBSCAN(eps=0.3, min_samples=10)
db = classifier.fit(newStoryArray)
labels = db.labels_
print_cluster(clusterTrainClass, labels, newStoryArray)

print
print "KMeans:"
classifier = KMeans(n_clusters=81)
db = classifier.fit(newStoryArray)
labels = db.labels_
print_cluster(clusterTrainClass, labels, newStoryArray)

print
print "Ward:"
classifier = Ward(n_clusters=81,copy=True)
db = classifier.fit(newStoryArray)
labels = db.labels_
print_cluster(clusterTrainClass, labels, newStoryArray)
```