RAZAVET Maël

CS 909 – Week 10

Text Classification, clustering and topic models

Technologies used in the project:

Python with many different libraries: NLTK, Pandas, Numpy, Scipy, sklearn, gensim, SGMLlib.

1. Description of the data, any pre-processing performed on it and why.

Description of the data:

The data is composed of 21,578 documents, which are part of the Reuters-21578 dataset. These documents are split into 21 SGML files. Each document within these SGML files is represented with a mix of HTML and XML tags. Here is an example of the structure of a document:

```
<REUTERS TOPICS="NO" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET" OLDID="5545"
NEWID="2">
<DATE>26-FEB-1987 15:02:20.00</DATE>
<TOPICS></TOPICS>
<PLACES><D>usa</D></PLACES>
<PEOPLE></PEOPLE>
<ORGS></ORGS>
<EXCHANGES></EXCHANGES>
<COMPANIES></COMPANIES>
<UNKNOWN>
&#5;&#5;&#5;F Y
&#22;&#22;&#1;f0708&#31;reute
d f BC-STANDARD-OIL-&lt;SRD>-TO   02-26 0082</UNKNOWN>
<TEXT>&#2;
<TITLE>STANDARD OIL &lt;SRD> TO FORM FINANCIAL UNIT</TITLE>
<DATELINE>    CLEVELAND, Feb 26 - </DATELINE><BODY>Standard Oil Co and BP North
America
Inc said they plan to form a venture to manage the money market
borrowing and investment activities of both companies.
    BP North America is a subsidiary of British Petroleum Co
```

Plc <BP>, which also owns a 55 pct interest in Standard Oil.
  The venture will be called BP/Standard Financial Trading
and will be operated by Standard Oil under the oversight of a
joint management committee.

 Reuter
&#3;</BODY></TEXT>
</REUTERS>

As asked in the exercise sheet, we will only consider the following tags to characterise a document: body, topics, reuters, dateline and title. The other tags are not relevant as most of the time, there are empty.

The body is the content of the article. The “topics” tag is the label that we aim at classifying the documents into.

The reuters tag is useful to split the data into the training and testing set, as one of the attribute in this tag is “lewissplit” which can take on either “train” or “test”. After splitting the data into the training and testing set, we can note that the training set represents around 70% of the data and the testing set around 30%.

Furthermore, I noticed that some documents were not having any “body” tags, so I decided not to consider them and delete them from the corpus. After this step, the corpus is composed of 19,043 documents.

Pre-processing performed:

- Tokenize: this aims at transforming the raw data composed of documents into a vector of words. Note that each document is represented as a “bag” of words. Example: “I like England” would be represented as [“I”, “like”, “England”]. This method allows us to represent a document in a vector and allows us to apply different techniques for text classification purposes.
- Punctuation removal: the punctuation is useless and not relevant for text classification. Moreover, this increases the size of the dataset used for classification. This is why it is primordial to remove the punctuation.
- Stopwords removal: many of frequent words are useless when doing text mining such as “the”,”of”, etc... These words are called “stopwords”. The NLTK package provides a tool which removes these words from a database. The advantages of removing these words are that the data would be reduced consequently (around 20 to 30%). Therefore, this improves the efficiency and reduces the computational time of applying any machine learning algorithms.

I chose not to apply any stemming algorithm, which aims at removing suffixes from words or replacing suffixes by another one. Example: The stem of “cooking” is “cook” and “ing” is the suffix. The reason why I did not apply any is because it could replace suffixes from words which would be meaningless afterwards. Example: the suffix “y” for the word “cookery” would be replaced by “cookeri”, which does not mean anything.

Moreover, here, removing links or replacing them by a placeholder is not applicable because there are no links within the corpus.

Concerning the recognition of Named Entities, the tool provided by NLTK package is trained on the ACE corpus. However, this tool would not be able to recognize efficiently named entities. I would have to train my own classifier to perform this task. This is the reason why I chose not to apply this method.

Note that for computational purposes, once the data was extracted from SGML files, I stored the data into a CSV file name “collection.csv”. In this way, I just need to load the data from this file and continue the execution.

2. Explanation of the features used, why these were chosen and how they were obtained. Numbers of different types of features. Details on any feature selection implemented.

In order to obtain my features (the terms used) for classification, I chose to use the top 1000 most frequent words in the entire corpus. To achieve this, I used the “TfidfVectorizer” class which is able to convert a raw document to a sparse matrix of TF-IDF features. See Code 1 in appendices. Note that in the argument of this function, there are the function “tokenize” and stop_words='english'. These functions are part of the pre-processing step described in the previous question. I selected the 1000 most frequent words as features because they are mainly representing the corpus and we will have a better chance to obtain probabilities for each document. Indeed, the choice of features is important here, as if a document contains none of the features selected, then the row representing the document in the sparse matrix would be only having zero probabilities. This is what we want to avoid, else this document would not be useful in the classification task. The number of features is also important. I chose 1000 because when using more, the classifiers are less accurate. Besides, I would like not to take fewer features because my intuition is that it would overfit the model. Indeed, by having fewer features, as these are the most frequent terms, the probability of finding each term in each document increases, which could lead to overfitting. Moreover, the sparse matrix would be smaller in terms of (columns or features) and it would be computationally a better choice than representing every single words of the corpus as features. The Figure 6 shows a feature representation obtained with the td-idf weighting method in a sparse matrix. Between brackets, we have the coordinates in the sparse matrix, and the number

next to the brackets is the weight for a given document and a given term (feature) computed by the tf-idf method.

The second part of the feature selection asked to perform is topic modelling. During the lectures, one of the topic modelling discussed was the LDA (Latent Dirichlet Algorithm). Basically, this allows selecting the top most frequent words for a given number of topics. The number of topics is a parameter of our algorithm. I could perform this task by using the gensim library. The code for this algorithm is given in the appendices (Code 2) with an example of the output (Figure 1). As we can see, the number of topics chosen in this example was 100 and the top 10 words (features) for each of the 100 topics are printed (Figure 1). The advantage of the topic modelling is that we need not any labels for documents. This algorithm can be seen as a clustering technique as we set the number of topics (number of clusters) and the words (members of clusters) are assigned to a cluster.

Therefore, depending on the number of features we want to select, we can vary the number of topics and the number of words selected for each topic. As I wish to have 1000 features, I chose to model 100 topics and to select the top 10 words for each topic. However, when selecting the top 1000 features extracted with the LDA, the results of the classifiers (detailed in the next question) are not good at all (between 45 and 65%).

As asked in the exercise sheet, I also combined the top 1000 features extracted from LDA and the top 1000 most frequent features. The results were acceptable. Actually, this method performs slightly worse than the first method described above.

3. Details on which classification algorithms were used and why. What parameters were used in each case and why.

As asked in the exercise sheet, the topics kept from the corpus are the 10 most populous classes, which are: earn, acquisitions (acq in the data), money-fx, grain, crude, trade, interest, ship, wheat, corn. The number of documents left is 8015. After selecting only these topics, the training set is composed of 5,557 documents (25%) and the testing set is composed of 2,458 documents (10%).

The algorithms that I chose to implement are the followings:

- Naïve Bayes
- SVM
- Perceptron with 500 iterations which according to me is enough to obtain a good classifier as we have 1000 features for about 5000 documents.

Note that, no specific parameters have been chosen to train the classifiers.

I chose to apply SVM (with the linear kernel) and Perceptron as my assumption is that the dataset (the corpus) is linearly separable. Indeed, these two algorithms can be applied when the data is linearly separable.

Perceptron has the capability of updating the weights of each node in the layers in order to reduce the misclassification of the instances until converging to the best linear classifier. Indeed, as seen in the lectures notes, Perceptron will converge to a linear classifier according the "Convergence Theorem". The only tuning parameter is number of iteration that will accomplish the algorithm. It is an important parameter because if this parameter is too small, then the algorithm will not have time to converge to the best linear classifier. Therefore, we need to choose a value which is high enough.

Concerning SVM, this algorithm is known to be effective and insensitive to outliers. Like Perceptron, SVM will find the best linear classifier if the data is linearly separable. As we are assuming that the data is linearly separable, this is the reason why I chose to apply the linear kernel.

Furthermore, I chose Naïve Bayes as a classifier because my assumption is that the terms are independent from the documents. Moreover, according to the lectures notes, NB is an effective method for text classification. Note that the Naïve Bayes classifier used in my work is multinomial NB, which is an extension to the regular Naïve Bayes. This uses a multinomial distribution instead of normal distribution. This is known to perform very well for text classification combined with tf-idf sparse matrices.

Finally, according to our lecture notes, these algorithms are supposed to achieve good results when having sparse matrices in input for text classification purposes.

4. Full evaluation of the classification algorithms. Compare different algorithms as appropriate and provide a final choice of an algorithm.

As asked in the exercise sheet, the evaluation metrics used are accuracy, precision, recall, micro average precision and recall, and macro average precision and recall. Note that here the use of micro and macro averaging metrics are relevant as we are performing multi-class classification.

Difference between macro and micro averaging:

- Macro averaging is giving the same weight to each label (topics in our case) as we are first calculating either the precision or the recall for each single class and then, we are taking the average of these.
- Micro averaging, as opposed to macro averaging, is giving the same weight to each document, as we are calculating the precision or recall for the entire document.

Figure 2 in the appendices shows a plot of the accuracy, precision and recall values obtained from the three algorithms with the top 1000 most frequent terms as the features.

Figure 5 in the appendices shows full details of each of the metrics values asked in the exercise sheet.

According to these values, Multinomial NB and Perceptron obtain similar results, 87% and 85% accuracy respectively. Concerning the precision, Multinomial NB performs with 86% and Perceptron 85%. About the recall measure, once again Multinomial NB performs slightly better with 87%, whereas Perceptron achieves 85%.

However, SVM is achieving much better results than the two previous ones, with an accuracy of 91%, a precision of 91% and a recall of 91%. Therefore, we can conclude that SVM performs better than the other algorithms.

The figure 3 shows the results achieved with the top 1000 features from LDA. As we can observe the measures are below 70%, which is much worse than the previous model.

Finally, the figure 4 shows the results achieved by combining the top 1000 most frequent terms from the corpus and the top 1000 features from LDA. As we can observe, the results are really good but it performs a bit worse than the first model.

5. Details on which clustering algorithms were used and why. Provide an evaluation of the clusters in terms of appropriate quality measures and how they correspond to the original TOPICS tags.

In this part of the exercise, I used the entire data without having any training or testing set as clustering is an unsupervised method.

The three clustering algorithms used here are the following ones:

- KMeans because it is a distance-based clustering technique.
- Ward because it is a hierarchical clustering technique.
- DBSCAN because it is a density-based clustering techniques.

I made the choice to use these three different algorithms in order to be able to compare completely different approaches, such as distance or density approach.

In order to evaluate the clusters in terms of quality measures, I computed the following measures:

- **homogeneity**: each cluster contains only members of a single class.
- **completeness**: all members of a given class are assigned to the same cluster.

These two measures can help us to claim whether the clusters correspond to the original topics tags or not. Indeed, if the values are close to 1 then it means that the clusters represent the topics. So, for each cluster, the members would be only the texts with the same topic. However, in our case, some topics are the combination of many different topics, so we can already assume that some documents would be part of different clusters.

The results of the algorithms are shown in the Figure 7. As we can observe, the results are not really satisfactory, especially for the DBSCAN algorithm (less than 10% for both measures). Therefore, we can conclude that here the density-based method is not relevant to gather documents into their respective topics tags. My intuition is that, as we have sparse matrices in the input, the density-based approach would not be suitable to find the right clusters.

Concerning the two other algorithms, they achieve between 23 and 43% for both measures, which is not acceptable as well.

We can therefore conclude that the clusters do not correspond to the original TOPICS tags.

6. Upload code on github. Provide link in report. Include a short README.txt file with an example on how to run the code.

Link to the code hosted on github:
https://github.com/mael221090/Text-classification

7. Improvement and feature work

When realizing this project, I was aware that many different approaches to achieve the work were possible, especially in terms of pre-processing. This is the reason why I would like to improve my work and add some pre-processing methods, such as lemmatization or get rid of the numbers. In this case, the features selected would be different and would not include any numbers for instance, leading to better and more accurate classifiers presumably.

Appendices

Code 1:

```
tfidf = TfidfVectorizer(tokenizer=self.tokenize, sublinear_tf=True, stop_words='english',
max_features=1000, lowercase=False, decode_error='ignore')
#function that trains the tf idf function
tfs_train = tfidf.fit_transform(data)
#funtion that applies the tf idf method previously trained
tfs_test = tfidf.transform(data)
```

TF-IDF feature representation

Note that the option sublinear_tf=True allows to rescale the term frequencies.

Code 2:

```
print("Topic modelling with LDA in process..")
stoplist = stopwords.words('english')
texts = [nltk.word_tokenize(item['body'].lower().translate(None, string.punctuation+'\xfc'))
for item in data if type(item['body']) != float]

    dictionary = corpora.Dictionary(texts)
    corpus = [dictionary.doc2bow(text) for text in texts]
    tfidf = models.TfidfModel(corpus)
    corpus_tfidf = tfidf[corpus]
    n_topics = 100
    lda = models.ldamodel.LdaModel(corpus_tfidf, id2word=dictionary,
num_topics=n_topics)
print("Topic modelling with LDA successfully completed\n")
return lda
```

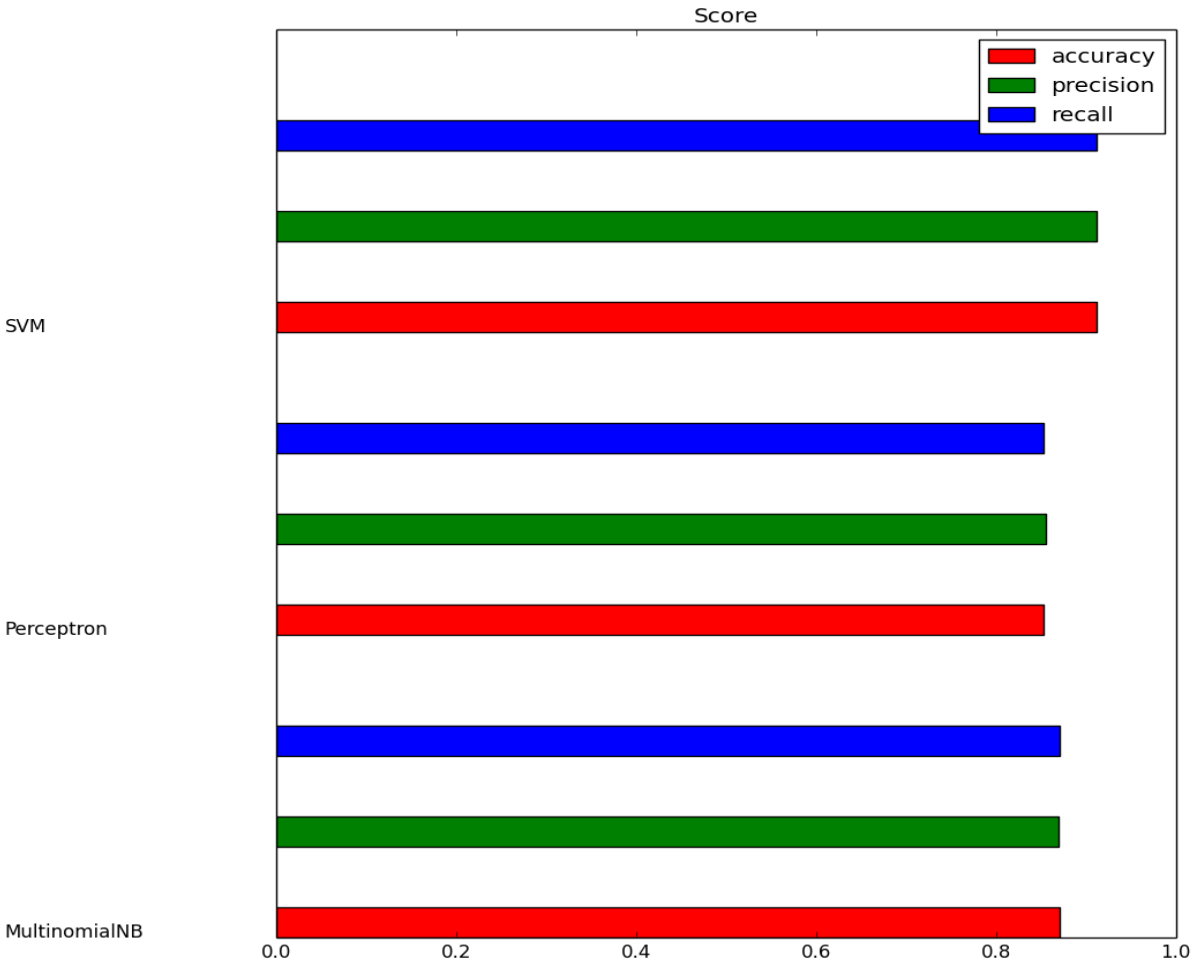
Code for LDA

Figure 1:

Top 10 terms for topic #0: southland, cotton, hercules, disney, joseph, bales, anderson, 483, wholly, freedom
Top 10 terms for topic #1: salomon, lp, lynch, merrill, halted, 126, moodys, dlrs, 188, billion
Top 10 terms for topic #2: gulf, iranian, iran, attack, military, war, copper, forces, kuwait, radio
Top 10 terms for topic #3: l, 295, 343, underwritten, 270, 2112, undervalued, smith, 320000, colorado
Top 10 terms for topic #4: baker, loan, savings, corp, james, federal, inc, insurance, volcker, stability
Top 10 terms for topic #5: dome, 1212, 2500000, plcs, 360, mexican, algeria, differential, 420, ugandas
Top 10 terms for topic #6: officer, chief, executive, president, vice, chairman, named, class, resigned, robert
Top 10 terms for topic #7: repurchased, miles, summit, csr, venice, factor, court, 153, fujitsu, ltd
Top 10 terms for topic #8: 7, el, tank, du, shipping, pont, byrd, monsanto, tonight, distributor
Top 10 terms for topic #9: notes, francs, pct, repurchase, swiss, add, bills, issue, mln, yield
Top 10 terms for topic #10: repurchase, authorized, yen, employee, buyback, miyazawa, irans, manufacturers, bills, stg
.....
Topic probability mixture: [(20, 0.31350909529685539), (25, 0.049917448031508008), (50, 0.61221448231266196)]
Maximally probable topic: topic #50
Topic modelling with LDA successfully completed

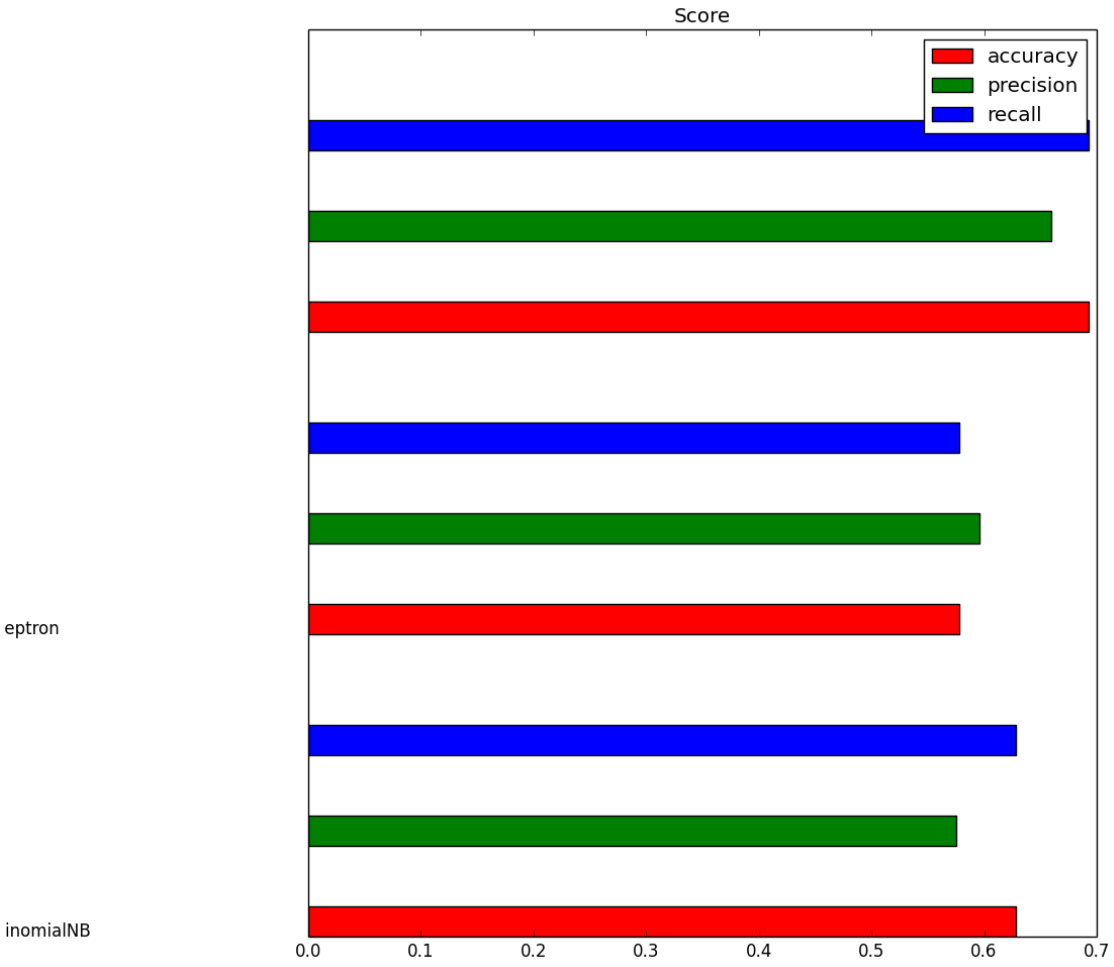
Example of LDA output

Figure 2:



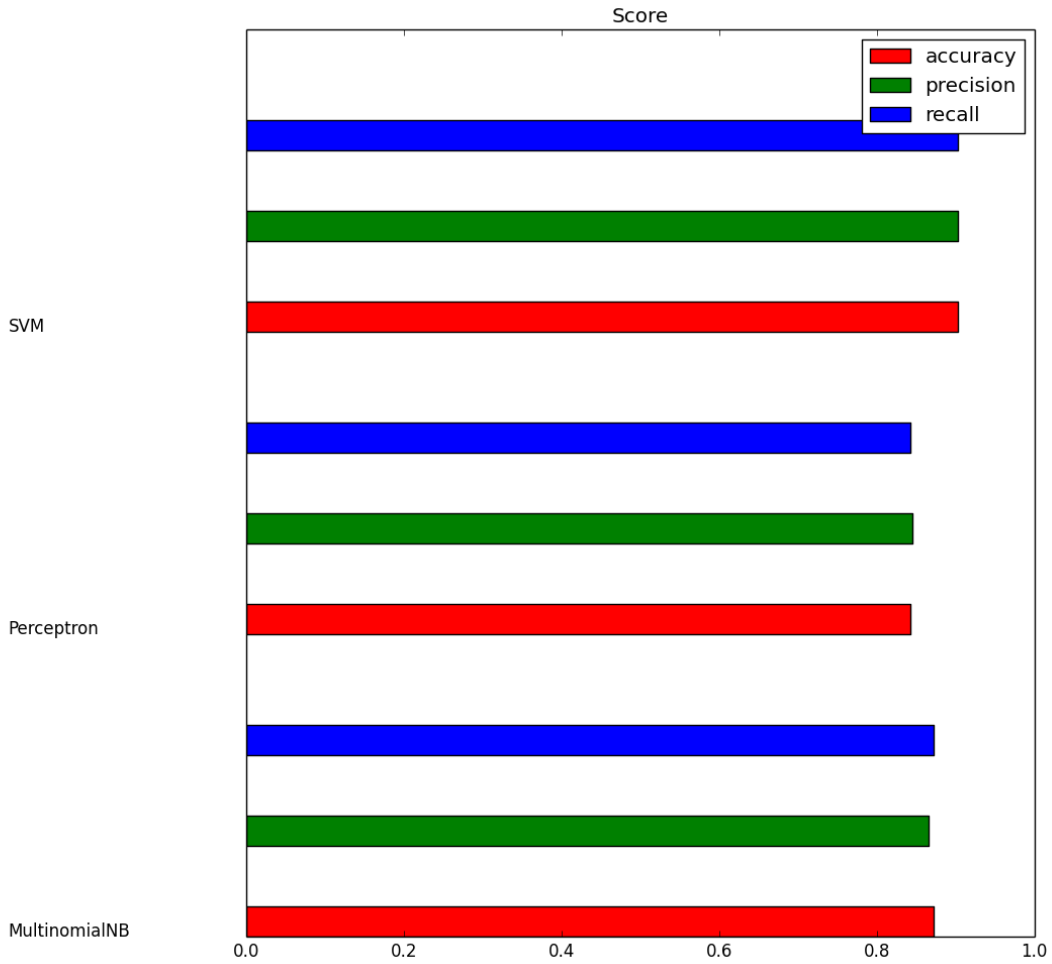
Performance evaluation with top 1000 most frequent terms

Figure 3:



Performance evaluation with top 1000 features from LDA

Figure 4:



Performance evaluation with top 1000 most frequent terms combined with the top 1000 features from LDA

Figure 5:

MODEL MultinomialNB
train time: 0.022s
test time: 0.003s
The precision for this classifier is 0.869642054835
The micro averaged precision for this classifier is 0.871033360456
The macro averaged precision for this classifier is 0.718593181592
The recall for this classifier is 0.871033360456
The micro averaged recall for this classifier is 0.871033360456
The macro averaged recall for this classifier is 0.692765796439
The f1 for this classifier is 0.868887973161
The accuracy for this classifier is 0.871033360456

MODEL Perceptron
train time: 2.844s
test time: 0.003s
The precision for this classifier is 0.855606435694
The micro averaged precision for this classifier is 0.853132628153
The macro averaged precision for this classifier is 0.712656024414
The recall for this classifier is 0.853132628153
The micro averaged recall for this classifier is 0.853132628153
The macro averaged recall for this classifier is 0.694064941107
The f1 for this classifier is 0.852241936569
The accuracy for this classifier is 0.853132628153

MODEL SVM
train time: 0.207s
test time: 0.002s
The precision for this classifier is 0.911749962385
The micro averaged precision for this classifier is 0.912123677787
The macro averaged precision for this classifier is 0.790495603732
The recall for this classifier is 0.912123677787
The micro averaged recall for this classifier is 0.912123677787
The macro averaged recall for this classifier is 0.776825592485
The f1 for this classifier is 0.911556874098
The accuracy for this classifier is 0.912123677787

Details of the performance evaluation

Figure 6:

(0, 986)	0.18601478263
(0, 941)	0.321789507229
(0, 939)	0.105816547169
(0, 861)	0.134744869905
(0, 808)	0.0294954478308
(0, 732)	0.101429884577
(0, 648)	0.145631451521
(0, 630)	0.372164937145
(0, 611)	0.0985135287352
(0, 587)	0.130378096987
(0, 461)	0.115538398265
(0, 445)	0.116370055103
(0, 417)	0.10840340459
(0, 412)	0.165569300331
(0, 411)	0.403690738807
(0, 399)	0.181701105499
(0, 299)	0.136969500776
(0, 205)	0.154221775203
(0, 116)	0.132989669323
(0, 101)	0.126590703518
(0, 99)	0.139793588747
(0, 94)	0.135990710384
(0, 66)	0.125741870273
(0, 61)	0.119531478268
(0, 46)	0.271421763327
: :	
(8012, 147)	0.0838248715605
. .	
. .	

Example of sparse matrix of the tf-idf method

Figure 7:

```
Clustering with KMeans
done in 13.736s
Homogeneity: 0.436
Completeness: 0.246
Silhouette Coefficient: -0.058

Clustering with DBSCAN
done in 426.872s
Homogeneity: 0.014
Completeness: 0.094

Clustering with Ward
done in 5438.183s
Homogeneity: 0.234
Completeness: 0.299
```

Ouput of the three clustering algorithms