**Text Classification**

**MLND Project Report**

**Prasann Pandya**

**Definition**

**Project Overview**

Understanding context from a given document by a computer is an age old problem under the field of natural language processing. Since my interests lie in the field of natural language processing, I decided to choose one of the fundamental tasks of natural language processing: **Document classification** or **document categorization**. **Document classification is an age-old problem in information retrieval, and it plays an important role in a variety of applications** for effectively managing text and large volumes of unstructured information. Automatic document classification can be defined as content-based assignment of one or more predefined categories (topics) to documents. This makes it easier to find the relevant information at the right time and for filtering and routing documents directly to users. The task is to assign a document to one or more classes or categories.

Every news website, scientific journals, online digital library, etc. needs to categorize document so that it can be easily found by users. Usually, this is done by manually screening and tagging of documents by humans. This task is not only time consuming, it can also be error prone and subject to bias as one person’s categorization can be different from other person’s categorization. Also, the screener may not be familiar with a particular topic. Since this task is redundant, manual and labour intensive (requires lot of reading), I decided to use Machine Learning techniques to automate this task. Machine Learning for classifying and categorizing documents can save time, improve accuracy and also remove bias.

There are many other applications for a document classification software namely: spam email filtering, genre classification (automatically determine genre of text), sentiment analysis, help librarians to categorize scientific papers by providing additional information beyond authors' keywords, assigning disease code in medical documents, etc.

There are many different approaches that have been used previously to tackle this problem previously. However, there is not standardized approach as far as I have found to classify documents. Thus, my purpose for this project is to try different supervised and unsupervised learning approaches to find the best model to classify documents.

**Problem Statement**

The problem I will be working on is automatic document classification. Document classification is a problem faced by all scientific journals, news organizations and digital libraries. The software will use documents as input and the software will automatically recognize which category it is related to from a list of categories. The performance of the software will be measured by how many documents it classifies correctly (accuracy) and also F1 score (precision and recall).

**Datasets and Inputs**

The dataset I will be using to solve this is popular 20 Newsgroups dataset. It is one of the datasets available in sklearn. The 20 newsgroups dataset comprises around 18000 newsgroups posts on 20 topics. Each document is labelled to be in one of the 20 categories.

Information on how to fetch 20 Newsgroups dataset from sklearn: <http://scikit-learn.org/stable/datasets/index.html#the-20-newsgroups-text-dataset>

This is a list of the 20 newsgroups:

1. comp.graphics
2. comp.os.ms-windows.misc
3. comp.sys.ibm.pc.hardware
4. comp.sys.mac.hardware
5. comp.windows.x rec.autos
6. rec.motorcycles
7. rec.sport.baseball
8. rec.sport.hockey sci.crypt
9. sci.electronics
10. sci.med
11. sci.space
12. misc.forsale talk.politics.misc
13. talk.politics.guns
14. talk.politics.mideast talk.religion.misc
15. alt.atheism
16. soc.religion.christian

I will be training and testing my models using this labelled data.

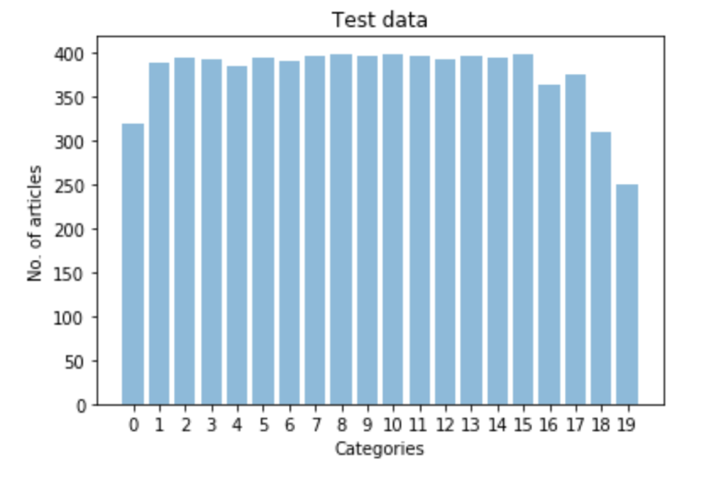
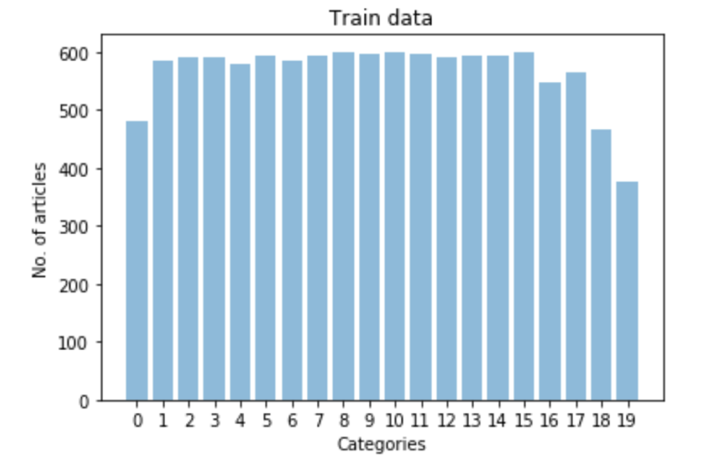
**Metrics**

Most text classification models are measured on accuracy. However, I want to use both Accuracy and F1 score as evaluation for the model since F1 score can provide better understanding of overall performance while taking into account the false positives and false negatives.

**II. Analysis**

**Data Exploration**

The dataset contains articles from 20 newsgroups. The data is fetched from sklearn datasets. The number of articles for each category in train and test data are shown as follows:



The categories are numbered as in the following list:

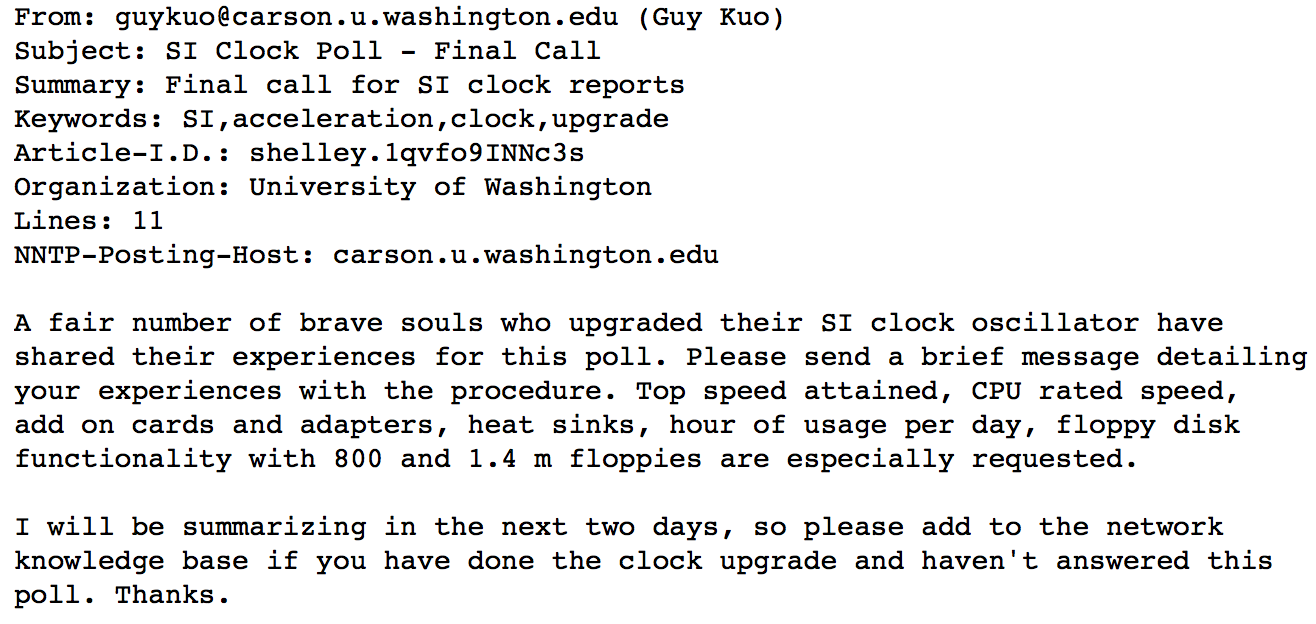
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast', 'talk.politics.misc', 'talk.religion.misc']

As it can be noticed in the above plots, the number of articles for each category in test and train data are proportionally the same. The category ‘talk.religion.misc’ has less number of articles in both train and test data. Thus, there is little to no chance of bias while training data.



The train-test split in the data is 60-40. The number of articles in the training data is 11,314 and the number of articles in the test data is 7,532.

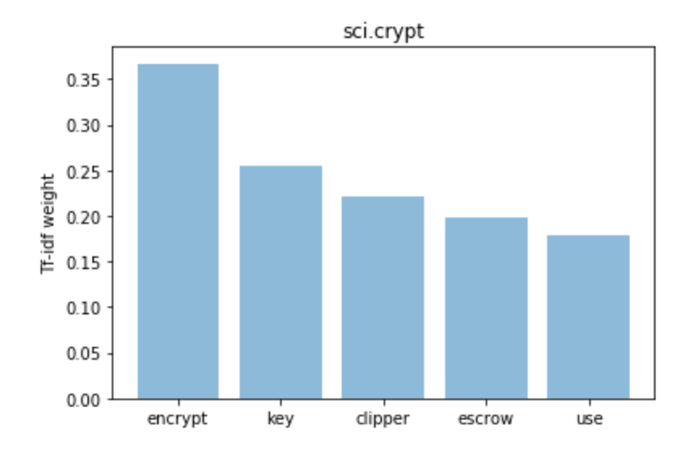
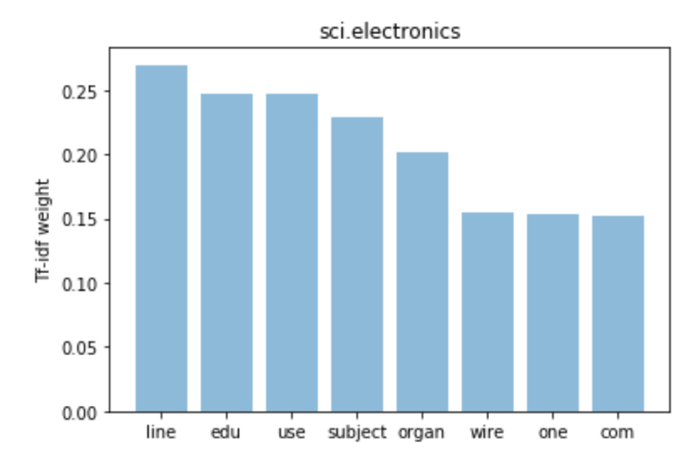
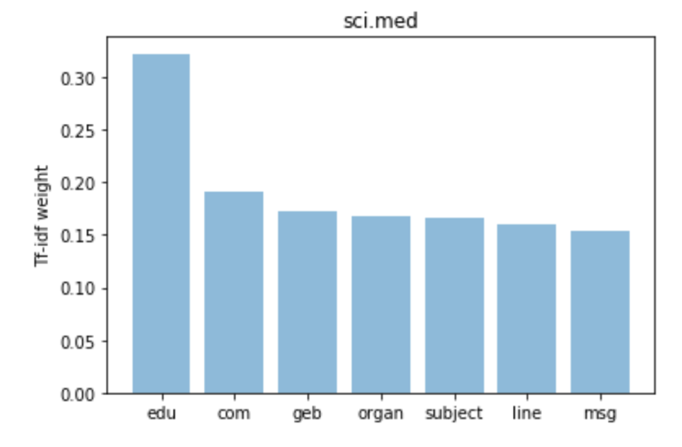
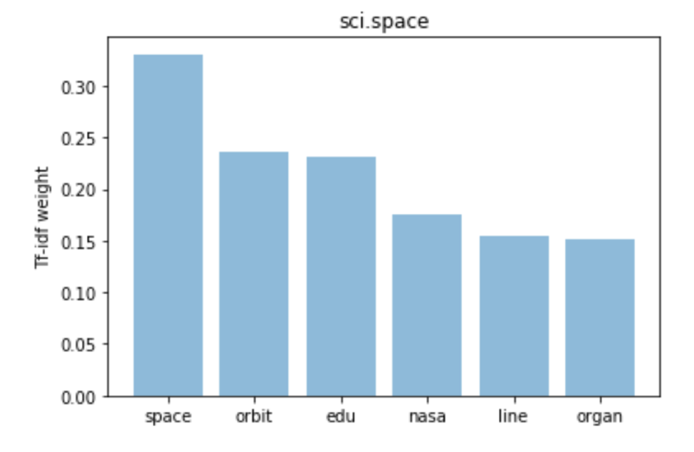
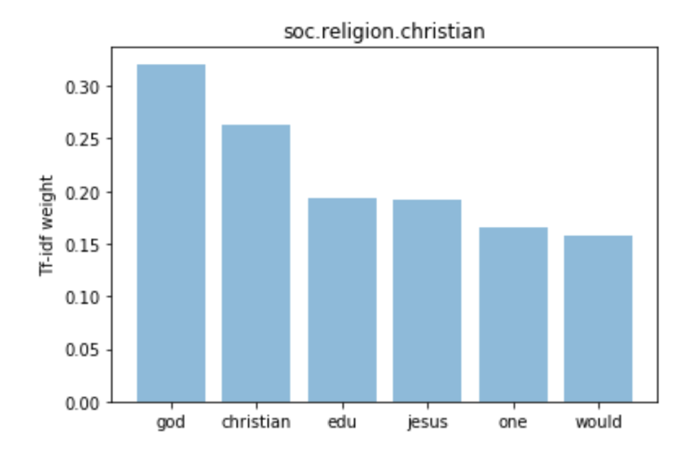
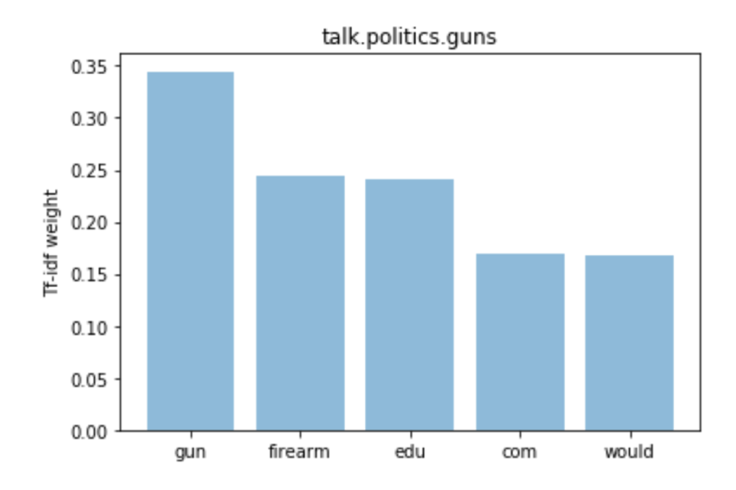
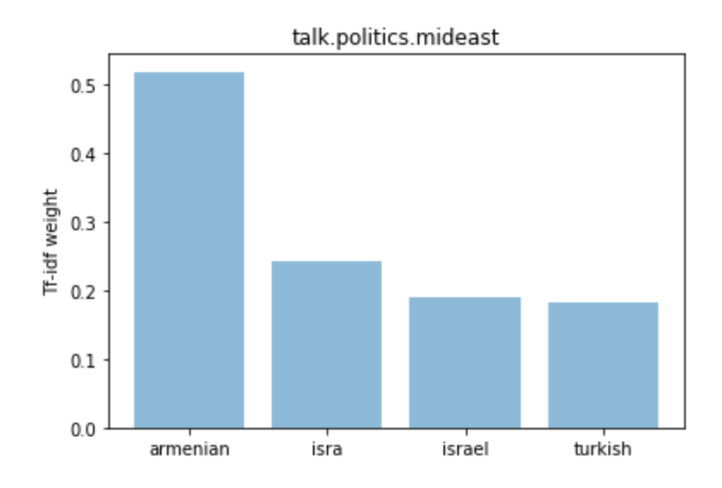
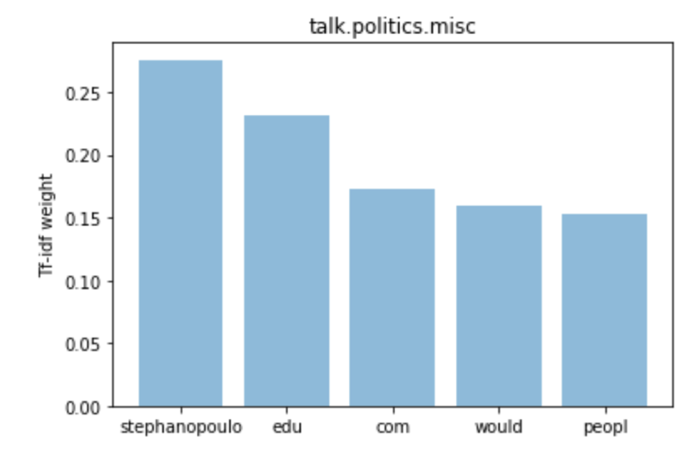
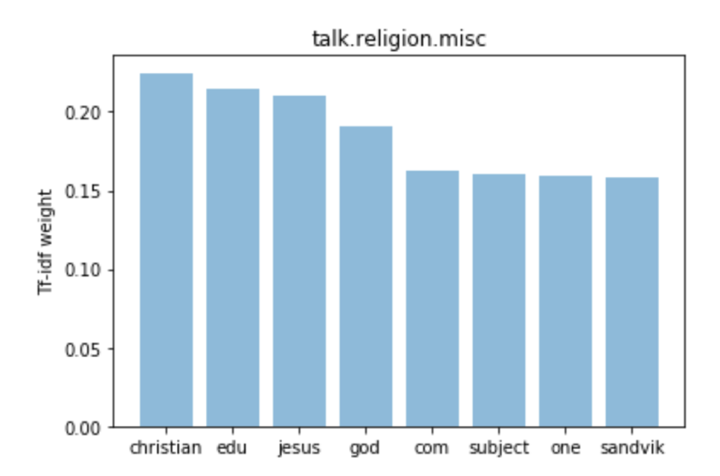
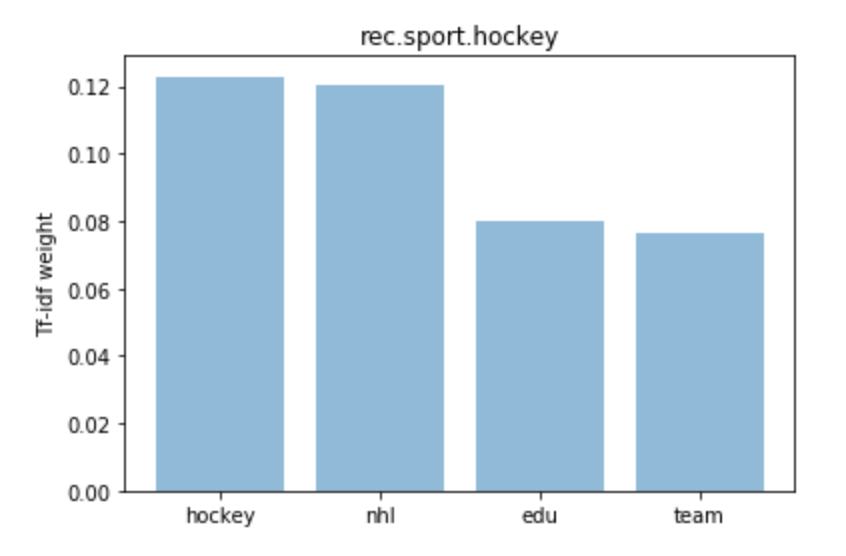
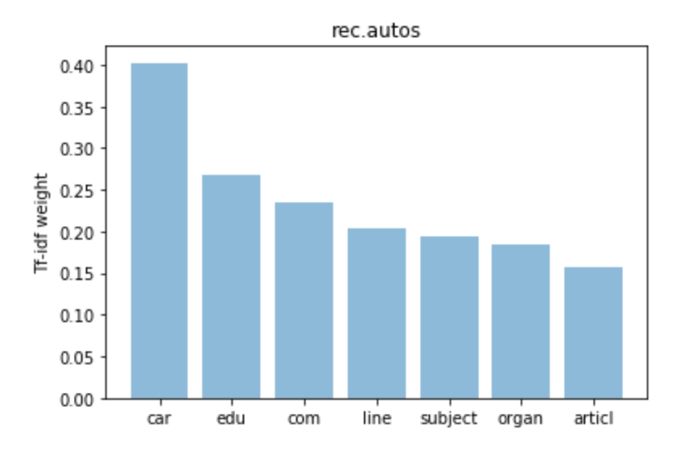
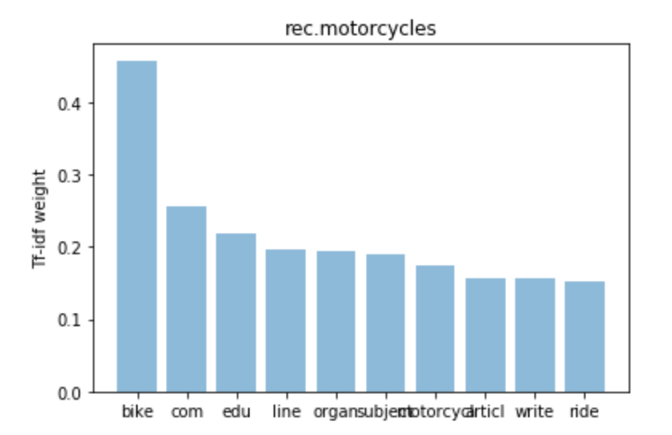
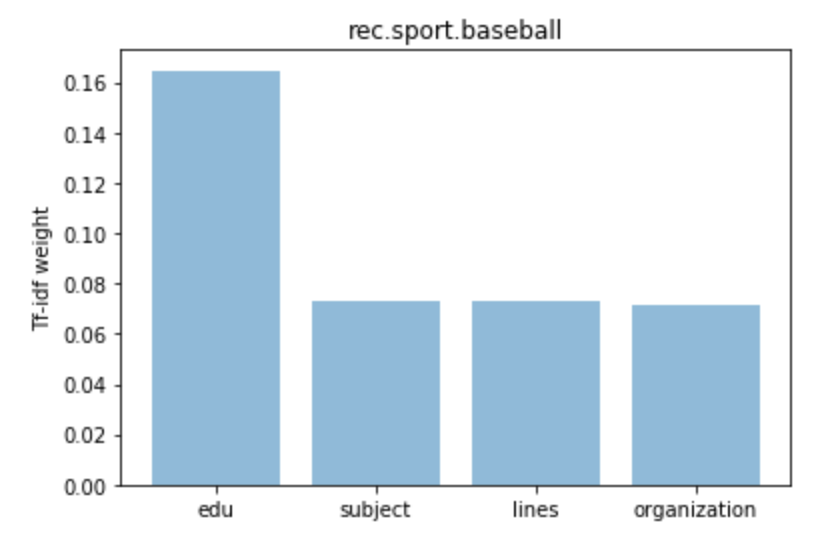
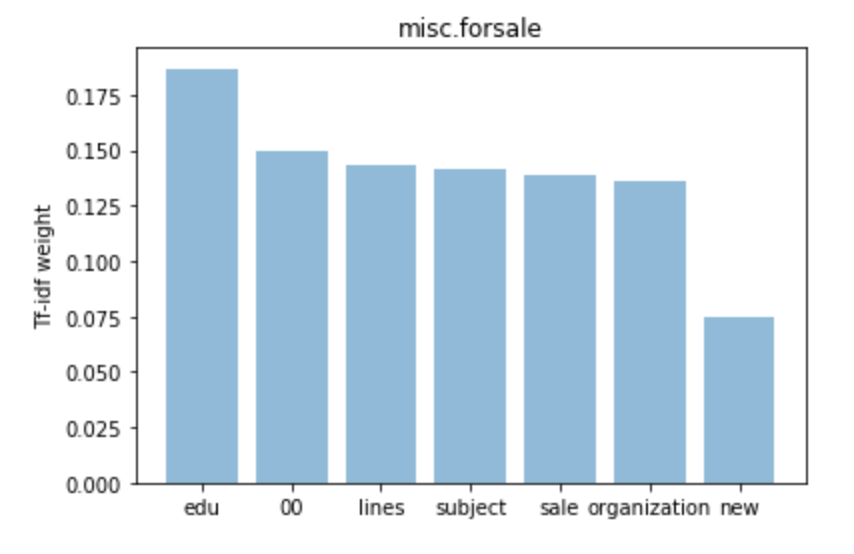
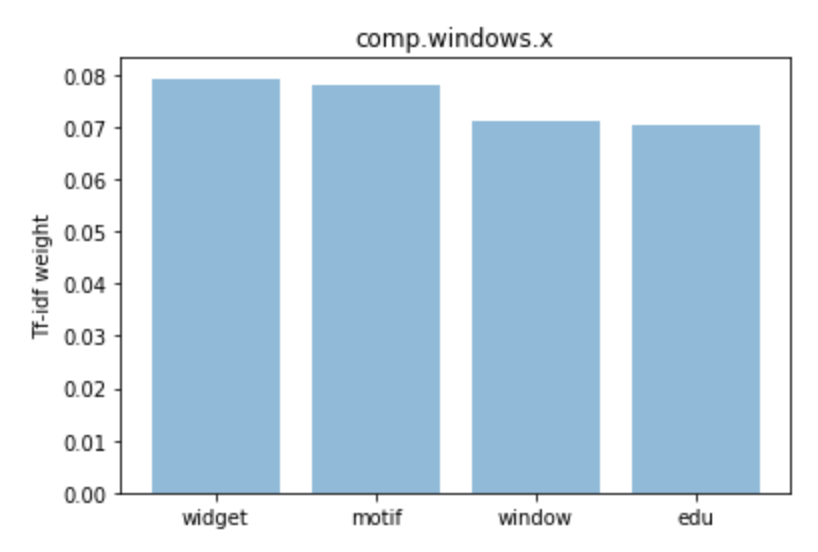
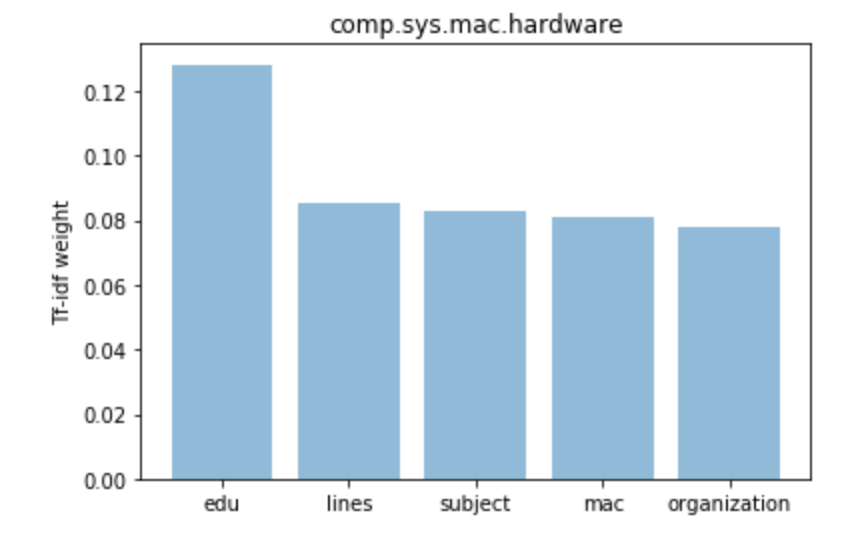
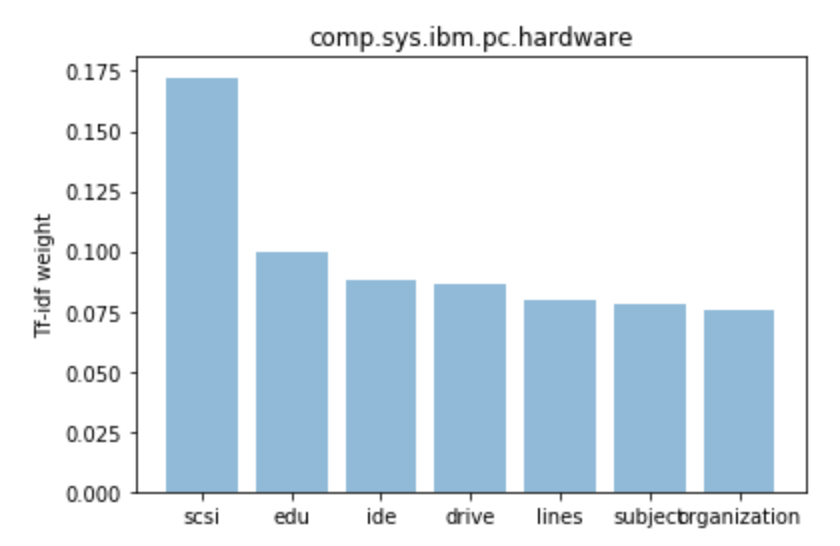
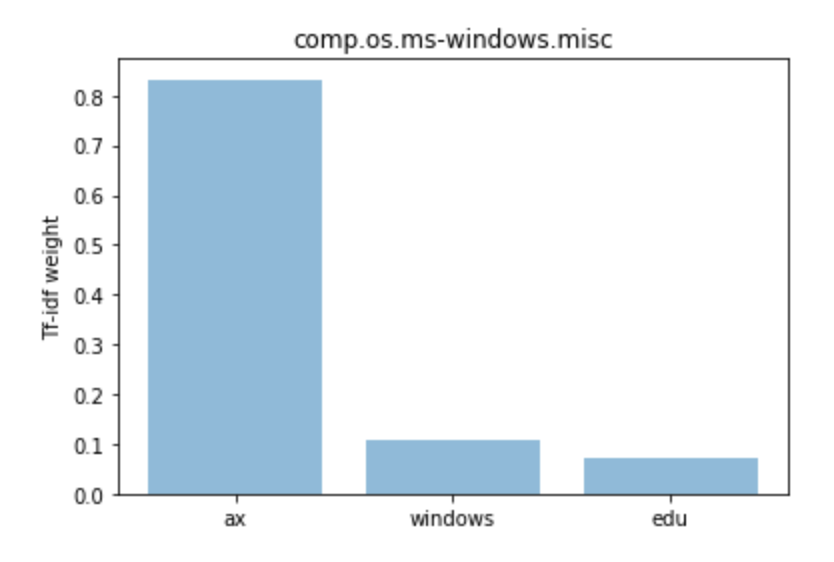
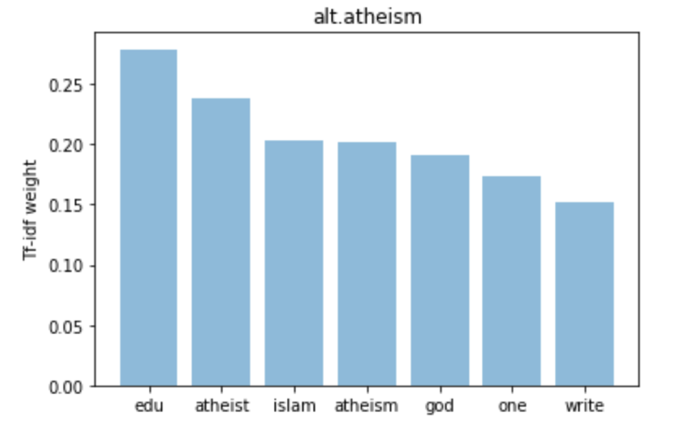
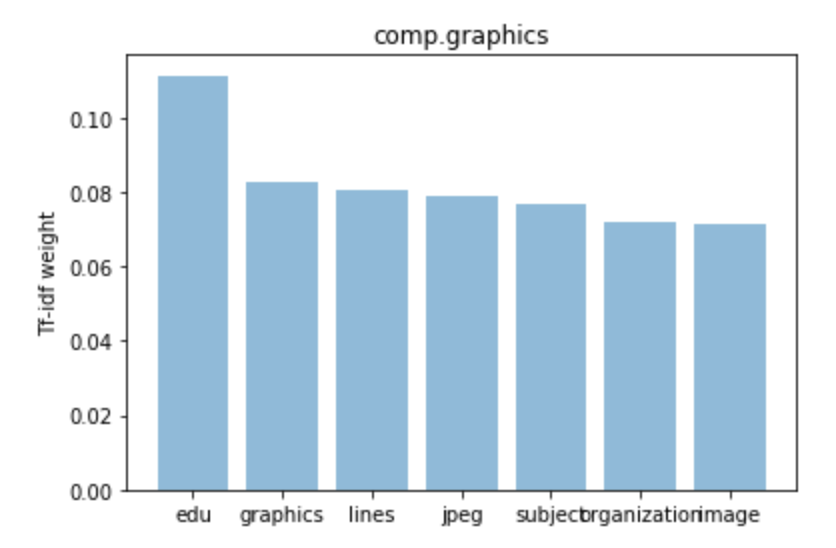
One example of news article from data related to category 'comp.sys.mac.hardware' as shown below:



As you can notice, each article contains header such as this and content as shown above.

**Exploratory Visualization**

To visualize the data, I applied tf-Idf (Term frequency – Inverse document frequency) weightings to words in each category to find most important words of each category. The tf-idf is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The tf-idf value increases proportionally to the number of times a word appears in the document and is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general [6]. After applying tf-idf, the most important words in each category were visualized as below:

Analyzing the plots above, we can notice that there are certain words that are repeated in many categories and are picked up by tf-idf as important words. There words include “edu”, “organ”, “line”, “subject”, that are picked up in multiple categories. However, there are other words in those categories that are specific to the particular categories. The words “edu” and “organ” occurs in pretty much every category and it does not add any meaning to any category. However, in order to create a model that can be applied to any new data, it’s best not to remove those specific words. Word “god” is repeated in both “Atheism” and “Christian” categories which might lead to misclassification. There are also similar words between “Windows” and “Hardware” categories.

Thus, these plots provide good insight into steps needed from preprocessing. Also, looking at the similarities between certain categories, there can be some misclassification.

**Algorithms and Techniques**

There are two techniques for converting documents into vectors that I decided to use to solve this problem:

1. **Tf-Idf (Term frequency – Inverse document frequency)**: The tf-idf is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The tf-idf value increases proportionally to the number of times a word appears in the document and is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general [6].
2. **Doc2Vec**: Tf-Idf has many disadvantages. The word order is lost, and thus different sentences can have exactly the same representation, as long as the same words are used [3]. To combat this, a new approach called Doc2Vec (also called paragraph vector) was introduced. In this approach, word order is taken into consideration. The vectors of words with similar context (surrounding words) are similar and thus the word meanings are taken into account in this approach. This Doc2Vec provides numerical representation of a document which represents concept of the document. More details on Doc2Vec approach are in [3].

Once the words were converted to vectors, many different classifiers can be used for document classification. However, I trained the model using the following two classifiers which are known to word well with text data:

1. **Naïve Bayes**: Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. It is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features [7]. Naïve Bayes has been used previously for text classification tasks such as spam filtering and sentiment classification and hence I decided to use this classifier.
2. **Linear Support Vector Machines (Linear SVM)**: A linear support vector machine constructs a linear hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection. Linear [8]. After much research, I found that text is often linearly separable and SVMs are most widely used for text classification.

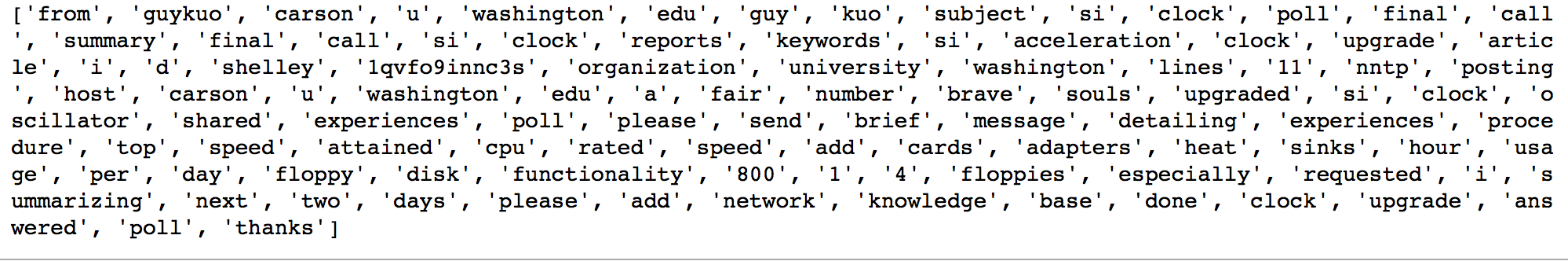
**Benchmark**

The traditional way of performing text classification is to use a bag of words model, convert the words to vectors using TF-IDF and using a classifier mainly SVM. This usually gives an accuracy of around 90% in document classification of 2-4 categories and around 80-85% when there are more that 4 categories.

**III. Methodology**

**Data Preprocessing**

The data was preprocessed using a tokenizer from nltk library which breaks down every word inside the article into a separate entry in a list. All the words were further converted into lower case. The example of words of an article after tokenization is shown below:



**Implementation**

**First Approach**

**Converting Words to Vectors:**

After the data was tokenized, words were converted to vectors using tf-idf. Tf-idf creates a vector with higher values for important words in a particular article. These vectors are created for each article. The vectors from each article are used as input for Naïve Bayes and SVM classifiers.

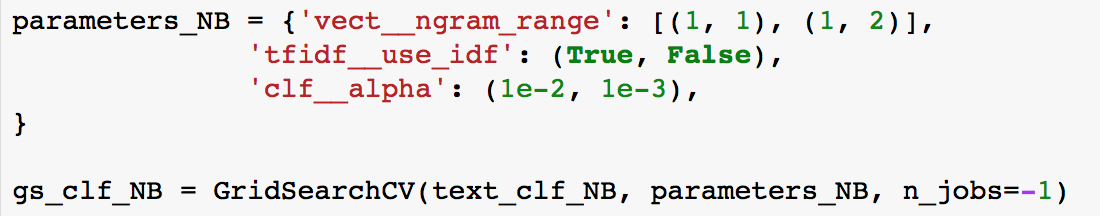
**Classification:**

After the data was converted to vectors using tf-idf, the data was then used as input in the following classifiers:

1. Multinomial Naïve Bayes
2. Linear Support Vector Machines
3. Linear Regression

I used MultinomialNB() classifier from sklearn.naivebayes library. Multinomial Naïve Bayes classifier is used specifically for text classification tasks as it is suitable for classification with discrete features [9].

Naïve Bayes has a parameter called alpha that can be changed to different values. The values for the parameter are decided using Grid Search functionality which checks for various combinations of parameter values to find the combination with best accuracy. A pipeline was developed for Naïve Bayes with CountVectorizer(), TfidfTransformer() and MultinomialNB(). The parameters for each of these steps are decided using GridSearchCV functionality from scikit-learn shown as below:



The best parameters after performing grid search for Naïve Bayes pipeline were found to be ngram\_range=(1,2), use\_idf=True, alpha=0.01. The classifier model was then fit to the training data and tested on test data. The model accuracy found using the default values was 81%. Using the grid search parameters, the accuracy of the model on test data increased by more than 2 percent to 83%.

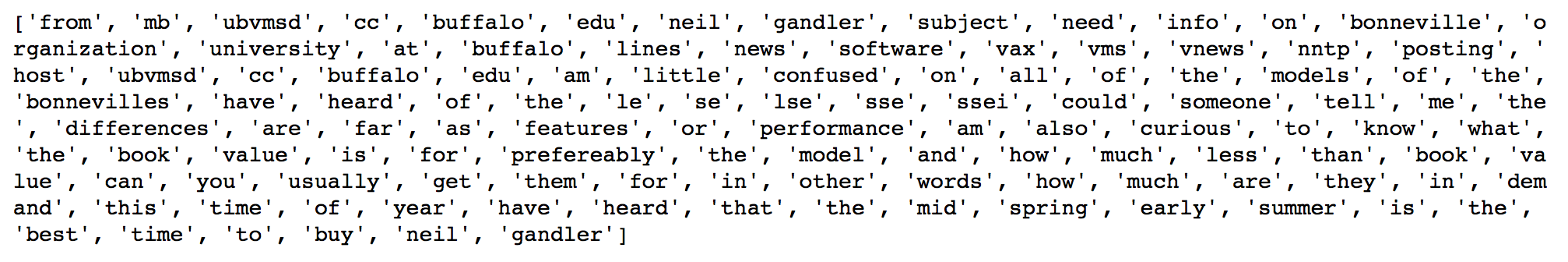
For the LinearSVC() classifier, a similar pipeline was developer containing CountVectorizer(), TfidfTransformer() and LinearSVC(). The parameters for these were again found using grid search to be ngram\_range=(1,2), use\_idf=True, C=10. The model was fit to the train data and tested on test data. After testing the model accuracy on test data using grid search found parameters, the model accuracy was found to be 86%.

Similar approach was used for linear regression classifier. The accuracy was found to be around 83%.

**Second Approach**

Since the tf-idf approach does not take into account order of words in an article and their semantic relationships, I decided to try out an approach called Doc2Vec which takes them into account. More details of this model are in [3].

In terms of preprocessing, the words are tokenized in a similar way. The stop words are not removed as they account to provide relationships between words which is the main purpose of using doc2vec. Sample of tokenization done using Doc2Vec is shown below:



Library called genism was used for implementing Doc2Vec [4]. It is one of the popular libraries for this purpose.

To convert all documents to vectors, the documents were assigned an id relating to their index. The vectors were then generated using doc2vec model. Each vector size was kept to 50. The vectors were trained for 25 epochs. The size of vectors and number of epochs was decided after much trial and error.

The time taken to convert documents to vectors using doc2vec was around 10-15 minutes. So, this was significantly high.

The new document vectors were then used as input into SVM and tested on test data. The accuracy of this model was around 55% which is much less than approach using tf-idf. Since doc2vec is a deep learning technique, it needs large amounts of data for each category to work well. The data in this case is not large enough for doc2vec to understand the ordering of words. Each category only has 400-500 articles and doc2vec generally needs thousands of input articles to understand the semantic relationships better. Also, the content in each article is very little which again leads to less training data. Thus, tf-idf is a better approach for this corpus.

**Refinement**

In this section, you will need to discuss the process of improvement you made upon the algorithms and techniques you used in your implementation. For example, adjusting parameters for certain models to acquire improved solutions would fall under the refinement category. Your initial and final solutions should be reported, as well as any significant intermediate results as necessary. Questions to ask yourself when writing this section:

* *Has an initial solution been found and clearly reported?*
* *Is the process of improvement clearly documented, such as what techniques were used?*
* *Are intermediate and final solutions clearly reported as the process is improved?*

There were a lot of refinements made to the model over time. There were many things tried through trial and error and the evaluation criteria was model accuracy and F1 scores.

**Refinement 1: Stemming**

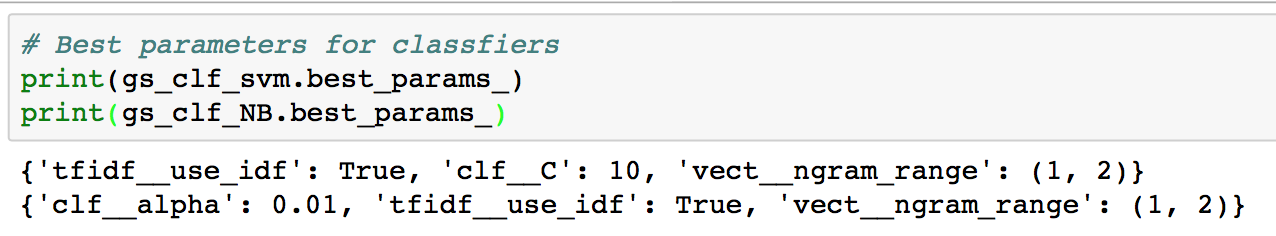
During tokenization of words from articles, stemming was tried to reduce each word to its root form. This can help with considering words such as “babies” and “baby” as same thing. However, the result before and after stemming made little to no difference (less than 1%). So, stemming was not performed during tokenization.

**Refinement 2: Removal of stop words**

The tf-idf model was tried after with and without the removal of stop words. This also made no difference in accuracy and f1 scores (less than 1%). This can be due to the fact that tf-idf on its own is pretty good at removing common words from all categories which are generally stop words. However, for purposes of making a good model, stop words were removed in the final solution as they add no value in this approach.

**Refinement 3: Parameter tuning using Grid Search**

The parameters of classifiers were tuned using Grid Search functionality in scikit-learn. Grid search goes through all the parameter combinations to find the combination with best accuracy. The best parameters for SVM and Naïve Bayes classifiers were found to be:



The ngram\_range variable for both was (1,2). This means that taking pairs of words (bigrams) were also found to increase accuracy than just unigrams while performing tf-idf. To further test this hypothesis, I tried using just unigrams and then both unigrams and bigrams to train the models. The increase in accuracy while using bigrams was 2-3%.

**Refinement 4: Finding appropriate size of vector and number of epochs while training Doc2Vec**

Doc2Vec

**IV. Results**

*(approx. 2-3 pages)*

**Model Evaluation and Validation**

In this section, the final model and any supporting qualities should be evaluated in detail. It should be clear how the final model was derived and why this model was chosen. In addition, some type of analysis should be used to validate the robustness of this model and its solution, such as manipulating the input data or environment to see how the model’s solution is affected (this is called sensitivity analysis). Questions to ask yourself when writing this section:

* *Is the final model reasonable and aligning with solution expectations? Are the final parameters of the model appropriate?*
* *Has the final model been tested with various inputs to evaluate whether the model generalizes well to unseen data?*
* *Is the model robust enough for the problem? Do small perturbations (changes) in training data or the input space greatly affect the results?*
* *Can results found from the model be trusted?*

**Justification**

In this section, your model’s final solution and its results should be compared to the benchmark you established earlier in the project using some type of statistical analysis. You should also justify whether these results and the solution are significant enough to have solved the problem posed in the project. Questions to ask yourself when writing this section:

* *Are the final results found stronger than the benchmark result reported earlier?*
* *Have you thoroughly analyzed and discussed the final solution?*
* *Is the final solution significant enough to have solved the problem?*

**V. Conclusion**

*(approx. 1-2 pages)*

**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**

In this section, you will summarize the entire end-to-end problem solution and discuss one or two particular aspects of the project you found interesting or difficult. You are expected to reflect on the project as a whole to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section:

* *Have you thoroughly summarized the entire process you used for this project?*
* *Were there any interesting aspects of the project?*
* *Were there any difficult aspects of the project?*
* *Does the final model and solution fit your expectations for the problem, and should it be used in a general setting to solve these types of problems?*

**Improvement**

In this section, you will need to provide discussion as to how one aspect of the implementation you designed could be improved. As an example, consider ways your implementation can be made more general, and what would need to be modified. You do not need to make this improvement, but the potential solutions resulting from these changes are considered and compared/contrasted to your current solution. Questions to ask yourself when writing this section:

* *Are there further improvements that could be made on the algorithms or techniques you used in this project?*
* *Were there algorithms or techniques you researched that you did not know how to implement, but would consider using if you knew how?*
* *If you used your final solution as the new benchmark, do you think an even better solution exists?*

**Before submitting, ask yourself. . .**

* Does the project report you’ve written follow a well-organized structure similar to that of the project template?
* Is each section (particularly **Analysis** and **Methodology**) written in a clear, concise and specific fashion? Are there any ambiguous terms or phrases that need clarification?
* Would the intended audience of your project be able to understand your analysis, methods, and results?
* Have you properly proof-read your project report to assure there are minimal grammatical and spelling mistakes?
* Are all the resources used for this project correctly cited and referenced?
* Is the code that implements your solution easily readable and properly commented?
* Does the code execute without error and produce results similar to those reported?

**References**

1. Evaluation of Text Classification (Stanford NLP Group): <https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-text-classification-1.html>
2. Working with text data: <http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html>
3. Distributed Representations of Sentences and Documents: <https://cs.stanford.edu/~quocle/paragraph_vector.pdf>
4. Gensim Doc2Vec tutorial: <https://github.com/RaRe-Technologies/gensim/blob/develop/docs/notebooks/doc2vec-IMDB.ipynb>
5. Document Classification Wikipedia: <https://en.wikipedia.org/wiki/Document_classification>
6. Tf-Idf Wiki: <https://en.wikipedia.org/wiki/Tf–idf>
7. Naïve Bayes Wiki: <https://en.wikipedia.org/wiki/Naive_Bayes_classifier>
8. Linear SVM: <https://en.wikipedia.org/wiki/Support_vector_machine>
9. Multinomial Naïve Bayes: <http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html>