# COMP 90042 PROJECT 2020: CLIMATE CHANGE MISINFORMATION DETECTION

**SUBMITTED BY :**

NAME : RIBHAV SHRIDHAR

STUDENT ID : 1037144

## INTRODUCTION

In January 2020, United States congressmembers made a statement to Google, asking them to act against "Climate Misinformation". The world wide web is open for all and everyone's opinion, but which opinion is true, and which is not remains to be seen.

In this report we are assessing the effectiveness of different machine learning methods and algorithms that can be used for predicting whether a piece of text is "Climate Misinformation" or not. While also gaining insights about how certain characteristics of a text can strengthen or weaken our results.

## DATA SET

We were provided with 3 data sets. All in .json format. Two of the data sets were labelled.

The train data set is a labelled data set, having 1168 text documents all which are labelled positive, i.e. classified as climate misinformation.

The dev data set having 100 text documents, which is to be used for evaluating our model, consisted of texts with both positive and negative labels.

The test data set containing 1410 unlabelled text documents, to be used for testing or model.

For any robust binary classification problem, we require two classes to train our model. As we were given only positive label text documents, we used a "BBC News articles data set" to put as negative labels for our training data set.

## PREPROCESSING

Before running any machine learning or feature engineering on our data, we first need to clean the data set, to bring it in a constant and uniform format.

We have used nltk's Regular Expression Tokenizer and Porter Stemmer. Also used the Stopwords corpus from the nltk package.

RegexpTokenizer : This tokenizer is used to split the string into substrings using regular expression. We use the "*nltk. RegexpTokenizer(r"\w+")*" regular expression

to remove any punctuation marks or any special characters from the text. [2]

Stopwords : We use the English Stopwords corpus to remove any stopwords from the text. {'ourselves', 'hers', 'between', 'yourself', 'but', 'again', 'there', 'about', 'once', 'during', 'out', 'very', 'having', 'with', 'they', 'own', 'an', 'be', 'some', 'for', 'do', 'its' } are some examples of English stopwords in the nltk package. [2]

Porter Stemmer : This is used to normalize the word to its root form. We do this to shorten our text and be more precise. [2]

For example : words : *{"Python", "Pythoner", "Pythoning", "Pythoned"}* all when run through the porter stemmer return the word "python"

After the above 3 steps we join each word to obtain a sentence like structure.

## FEATURE ENGINEERING

This step is used to convert our raw text data to machine friendly feature vectors.

We have used 3 different vectorizers to find out which one suits our data set the best.

- DictVectorizer
- HashingVectorizer
- CountVectorizer

**DictVectorizer** : DictVectorizer is used to convert feature arrays represented as lists of standard Python dictionaries to the NumPy/SciPy representation used by scikit-learn estimators.

We have used DictVectorizer for Naïve Bayes.

**HashingVectorizer** : HashingVectorizer is the most memory efficient of the above two. It is best used when dealing with a large data set. In this, instead of storing the tokens as strings, the vectorizer applies hashing to convert them as numerical indexes.

We have used HashingVectorizer for Decision Trees

**CountVectorizer** : The CountVectorizer provides a straightforward way to both tokenize a collection of texts and generate a vocabulary of known tokens, but also to encode new texts using that vocab.

We have used CountVectorizer for LinearSVC

All the above 3 vectorizers are present in the Python Scikit-Learn package. [1]

## MACHINE LEARNING ALGORITHMS

This section defines and explains the algorithms used to generate predictions after the pre-processing and feature engineering
And also, the methods used to improve on those predictions. We have used three different Machine Learning models to train our data set.

All of the below mentioned classifiers are used from the Python Scikit-Learn library. [1]

- **Multinomial Naïve Bayes :**

We have used Naïve Bayes as the base line classifier.
Naïve Bayes is a classifier based on Bayes' theorem. This classifier has been used very

often for sentiment analysis, news categorization, spam filtering, etc.
Naïve Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. This is called class conditional independence.

- **Linear Support Vector Classification**

The aim of a Linear SVC is to the train data and return a "best fit" hyperplane that divides, or categorizes, our data.

Comparable to SVC with hyperparameter kernel='linear', but applied in terms of liblinear rather than libsvm, so it has more elasticity in the choice of penalties and loss functions and should scale well to large numbers of inputs.

This class supports both dense and sparse input and the multiclass support is handled according to a one-vs-the-rest scheme.

- **Decision Trees**

Decision trees builds a classification model in form of a tree structure.

The Gain Ratio measure is used to breakdown the data set. The selected attribute is made a decision node, and keeps breaking down the data set until one of the conditions matches:

- All instances belong to the same attribute value.
- There are no remaining attributes/instances.

The working of these algorithms has been further explored and explained in the Experimental Section.

# EXPERIMENTAL

In this section we will explain the working of the Machine Learning Models and the motivation behind each one of them.

- We have used Multinomial Naïve Bayes as a baseline model.
  After preprocessing both the train and the dev set (as explained in the Preprocessing section), we implement a bag of words approach using the DictVectorizer.
  We decided to use Naïve Bayes because it is easy and fast to implement and gives constant results regardless of the feature engineering. As we have used MNB as a baseline, so look to improve our feature engineering and use a more robust Machine Learning Method to improve our scores.

- Next we have used the Decision Trees model to train our data.
  Trees perform fairly well for Text classification problems, as with text documents the vocabulary can be high and sparse.
  We have used decision trees with both with DictVectorizer and HashingVectorizer
  It performs better with Hashing.
  We have tuned the hashing vectorizer to accommodate high number of features to avoid collisions as our vocab count is high.

As Decision Trees in Scikit-Learn is a non-parameterized method, no further hyperparameter tuning is required.

- For our final and best approach, we have used LinearSVC.
  For this we are using a pipeline approach, with CountVectorizer.
  We apply hyperparameter tuning to find the best tuning for our approach and data set.

## RESULTS

To assess our models we are using Precision, Recall, Accuracy and F1 score as the evaluation Metric.
Below are the scores for Multinomial Naïve Bayes (Baseline)

| Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|
| 59% | 0.51 | 1.0 | 0.67 |

As it can be seen that the model is working fairly randomly and needs improvement.

A limitation of Naïve Bayes is the assumption of independent predictors. In actuality it is next to impossible to have predictors which are completely independent of each other.

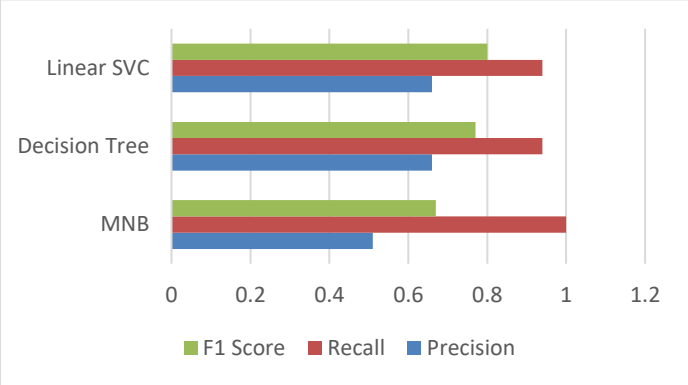
Results for Decision Trees

| Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|
| 73% | 0.66 | 0.94 | 0.77 |

Decision Trees tend to overfit to accommodate any outliers, which in turn affects its performance.

Results for Linear SVC

| Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|
| 77% | 0.66 | 0.94 | 0.80 |




Clearly the LinearSVC gives the best results out of the 3 models we have tried.
This is because, generally Support Vector based classifiers work comparatively well when there is a clear difference between the classes.
Although more noise can hinder its scores, so efficient pre-processing and feature engineering is very important.

## ERROR ANALYSIS

After studying and analyzing the predicted labels for the dev data set with the actual true labels, we can see a few patterns due to which some errors are arising in our approach.

- Naïve Bayes is a very probabilistic model, and a major issue/cause for error is its assumption of independent features, which is rarely true in the case of text tokens.
  Hence it produces high number of false positives.
- Words such as “PM”, “Government”, “Stated” are very commonly used in real true news texts/articles, as we are using extra data from “BBC news articles” to put as negative labels for our training data set. These words are actively present in that data set and has

been labelled as Negative in our Training data.
Now, any Positive labelled text in the dev data, which has these words (example “Scott”, “Morrison”, “uk”, ”British”), gets mapped as Non-Climate misinformation. (Following the one-vs-the-rest scheme)

- On a similar note, it can be seen (to some extent) that texts with high amount of “Numerical” data is classified as non-climate misinformation in our training, because of that similar texts in the dev set generate errors.

## CONCLUSION

Overall, analyzing our model and the different algorithms used for the problem in hand, it can be seen that Training data plays a very important role in the process of machine learning. As we did not have any text for negative labels, getting “makeshift” data from external sources worked fine with a small data set(dev), but underperformed with the test data set, as we achieved a final F1 score of 0.68.

This was due to the text distribution of both the labelled data sets (Train and Dev) was very different to that of Test data, also the test data was very vast to as compared to the dev. So, evaluating and building our data w.r.t to the dev set did not represent the true extent of our results and the errors faced.

Also, text cleaning is utmost important, as it reduces noise in the data. Having significant noise can make the models overfit.
And throwing a light on future work, we can use ensemble models, i.e. stacking different classifiers and mixing their predictions will further help improve out results.

## REFERENCES


1. https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
2. https://www.nltk.org/py-modindex.html
3. https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/
4. https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a
5. External Data - https://www.kaggle.com/yufengdev/bbc-fulltext-and-category