University of Southampton

MediaEval Classifier

Report of an approach to mediaeval challenge

Kritagya Gurung – kg4g18
[Date]

Introduction and Problem characterization

Our task is to train a Machine Learning (ML) algorithm to correctly classify which tweets are fake and which tweets are real using the tweet's content as a basis for classification. This task essentially is a binary classification problem, the two classes being fake (humour label is classified as fake) and real.

Additionally, I will not be using any data relating to the author of the tweet, like how many followers or their biography description. I am only going to be using the following information to train the machine learning algorithm: tweetId, tweetText, userId, username, timestamp and the label (ground truth). In total, we have 14483 tweets for the training dataset and we do have a separate testing set to do our final evaluation on. I do have image id's, but I will not be using them.

We must use F1 score as a success metric for our implementation, but I am also including recall and precision scores as our dataset is biased.

Literature review

There have been multiple approaches before attempting to classify tweets into fake tweets and real tweets. A lot of these approaches use additional data such as how long the author of the tweet has been with twitter (1) (2) or how many friends the author has (1) (2) (3) (4).

Fortunately, these methodologies included what features were extracted and used from the tweet's content; of which we have access to. These include: length of tweet (3) (2) (4), does tweet have URL? (1) (3) (2) (4), does tweet have ! or ? (3) (2) (4), does tweet have #words? (1) (3) (2) (4) etc . This is key as it shows what features provide useful descriptive power to be able to classify fake and real tweets since these same features were also used in classification.

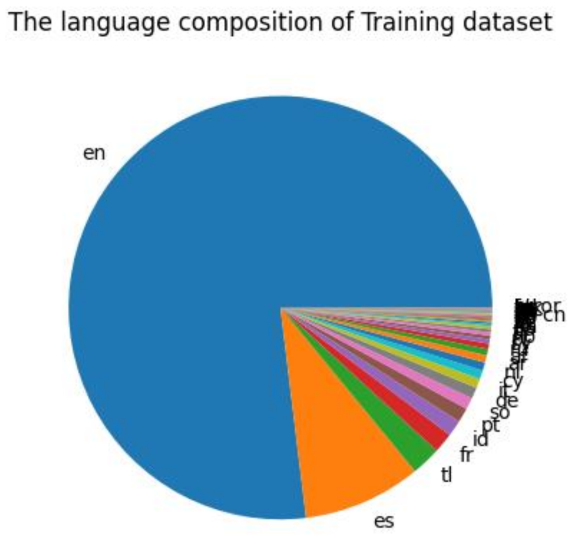
Previous attempts have used a wide range of ML algorithms and it seems these ML algorithms have been used a lot: support vector machines (5) (3) (2) (6) (4) (7), logistic regression (5) (2) (6) (4), random forest (1) (6) (4) (7), naïve bayes (1) (2) (4) (7), k-nearest neighbours (8) (4), etc. This is vital information since it shows the effectiveness of each technique and thus must be a reason to use them in this classification task.

The methodologies also mentioned important methods in feature extraction on the tweet's content via Parts of Speech tagging (POS) (8), n-grams (1), term frequency inverse document frequency (tf-idf) (5) (2) (6) (7) also Bag of Words approach (8) (2) (7).

Data Visualisation & Analysis

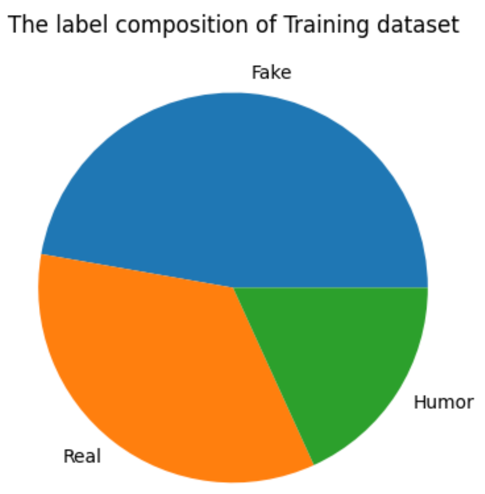
Firstly, I had a quick look at the training set.txt file and saw that there were 14483 records. There was not any data regarding the user that we could use e.g number of followers. The training set.txt file had the following columns tweetId, tweetText, userId, username, timestamp and the label. However, the tweetText data had some inconsistencies and interesting traits. Not all the tweets were English, tweet id 263046056240115712 was identified to be in Spanish. The tweetText had some inconsistencies regarding the URL as some tweets had the symbols escaped such as tweet id 443365433484521472. The tweetTexts also had a lot of recurring traits like the use of hashtag, mentions and emojis which seemed like a suitable feature to extract. I believe the timestamp was all converted to UTC. A lot of the tweetTexts had the exact same content such as tweetIDs 264054356855365633 and 264055584641400832 although their author was not the same. Some authors have also posted multiple tweets in the same dataset such as username CarlosVerareal.

I used the langdetect library to determine what language the tweet was written in to find out.



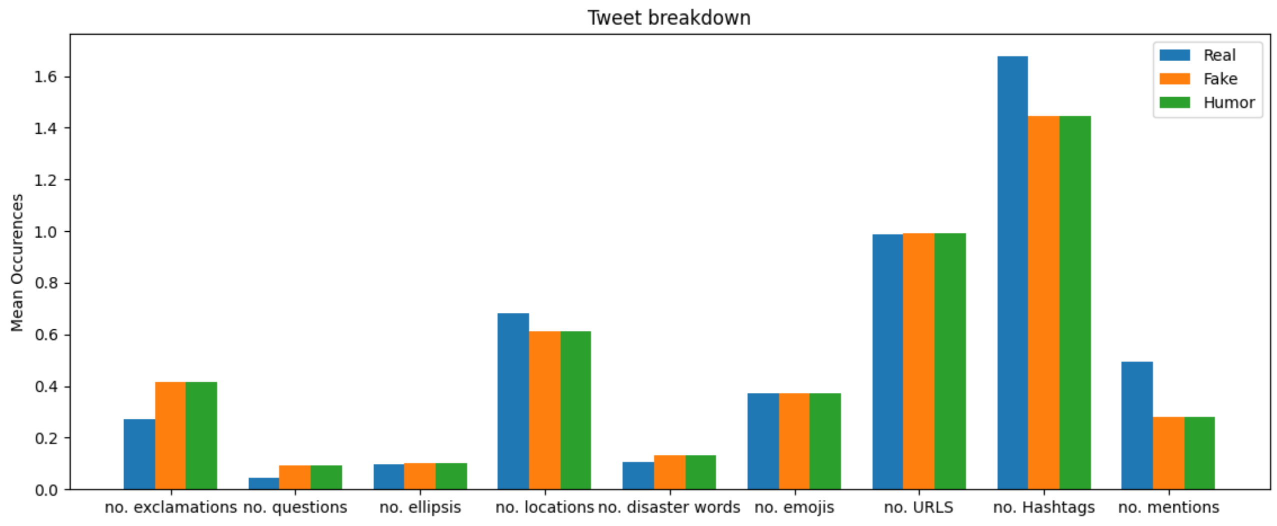
As you can see, English, Spanish, Tagalog, French and Indonesian are the most common languages in the training dataset in that order. English tweets make the majority of the training dataset (76.93% or 11142 tweets) which I deemed to be large enough for a training dataset. I believed that translating each tweet into English and performing feature extraction on (probably) broken English would yield very little, especially when all the other languages combined only make up 23.07% (or 3341 tweets). Also, most of the nouns such as location names would generally be still be English. Interestingly, langdetect library had trouble identifying tweet id "262974742716370944" due to poor spelling and very little tweet content to analyse. This tweet was counted as error as manually checking what language every tweet is assigned is far too exhaustive.

In this training dataset, I looked at the ground truth labels to check for any dataset bias.



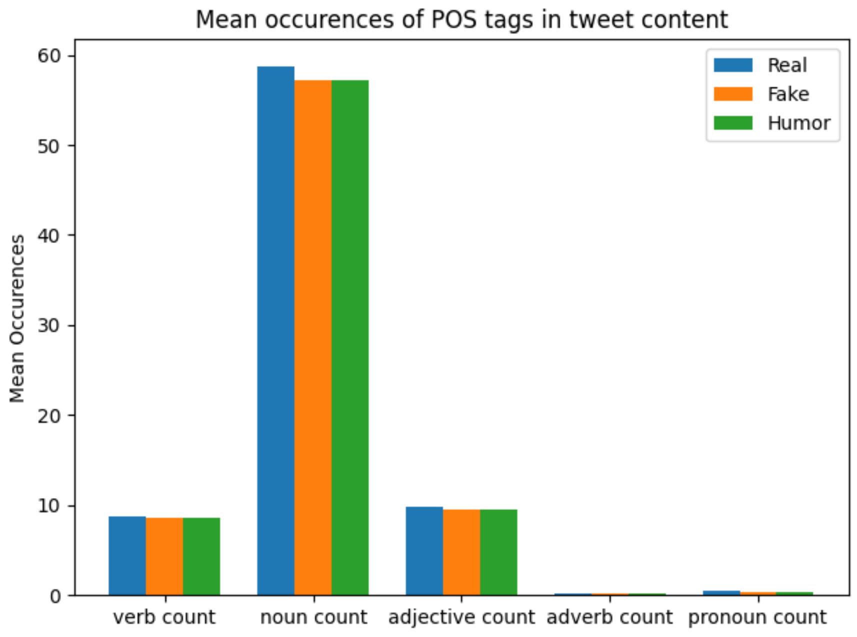
There are 5009 real labels (34.59%), 6841 fake labels and 2633 humour labels. In total, there are 9474 fake labels (65.41%) which is includes the Humour label. This is very important because now we have a large dataset bias i.e. a dumb algorithm that would guess fake for every post would be 65.41% accurate, for this dataset. This means for our training cycles; we would need to even out the dataset labels to avoid this bias.

I then analysed the tweet contents for punctuation, emojis, URLs, hashtags, mentions, number of locations mentioned, and number of disaster words mentioned for each label in the training dataset.



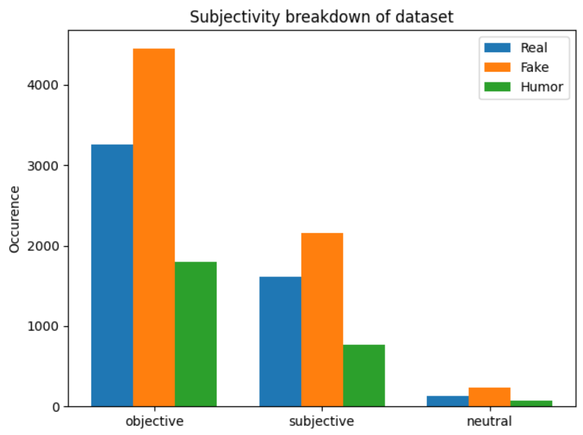
It seems that real tweets use less exclamation marks and question marks however uses more hashtags, mentions and references to locations when compared to fake and humorous tweets which implies that these are suitable descriptive features. Also, Emoji, URL and ellipsis usage do not seem to have much descriptive power as they remain consistent with other classes.

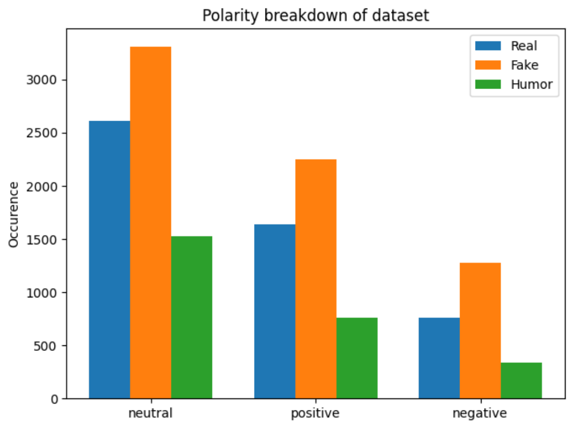
Next, I looked at the Parts of Speech tags composition of the whole training dataset. This was done via nltk's pos_tag() function which provides tags to tokens (using nltk tokenize function).



Unfortunately, these features are not too useful for us as there does not seem to be a significant difference between usage in fake and real tweets. There also seems to be an absurdly high noun usage in both real and fake tweets. This is most likely to have been caused by different languages or misspelt words being identified as nouns.

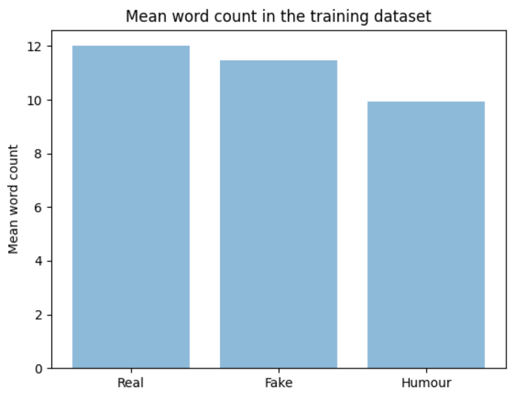
I then analysed the subjectivity and polarity of each tweet via TextBlobs .subjectivity and .polarity attributes. I then gave them a custom label based on their scores. For polarity, a score of less than 0 was negative, same as 0 was neutral and a score above 0 was positive. For subjectivity, a score of less than 0.5 was objective, a score of 0.5 was neutral, and a score above 0.5 was subjective.

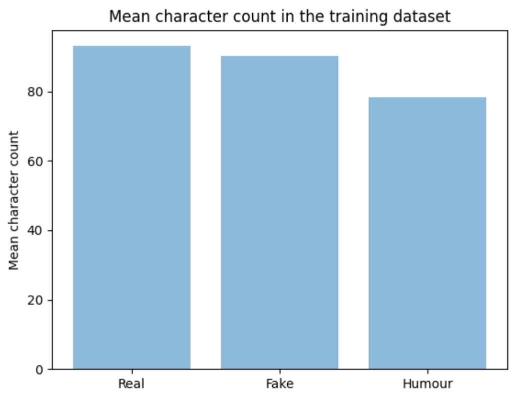




It seems that these features seem descriptive however the large differences between classes could have been caused by the fact that there is not an equal amount of these labels. Also, the subjectivity and polarity attributes for tweets of other languages could have defaulted to neutral in polarity and objective in subjectivity due to them being in a different language and 0 being the default value.

Further analysis on the length of tweets.





It seems to be that real tweets tend to be longer in both word and character count in comparison to fake and humorous tweets. These features seem to have adequate descriptive power in classifying real from fake tweets.

Algorithm design

I first conducted research into text classification and what were the general approaches to this type of problem. Academic papers have shown that traditional supervised ML algorithms are viable for these sorts of problems, and I have more experience in traditional ML than Neural Networks. I ultimately went for the traditional approach. For pre-processing, I simply converted to the ground truth label into a binary value (0 for real, 1 for fake and humour) so that the ML algorithms can interpret the target values. Additionally, I used the stopwords list from nltk library to remove words that don't add meaning to a sentence like "I".

My first approach was to use tokenization of the tweetText via word_tokenize() from the nltk library .This was not very successful. The features that I came up with were as follows: language, polarity, subjectivity, polarity score, subjectivity score, character count, punctuation count, number of exclamations, number of questions, number of ellipsis, word count, noun count, verb count, adjective count, adverb count, pronoun count, number of locations, number of disaster words, number of emojis, number of URLS, number of hashtags, number of mentions, word density and target.

Language identification was via langdetect library using the .detect() function which gave a 2 letter abbreviation of the most likely language.

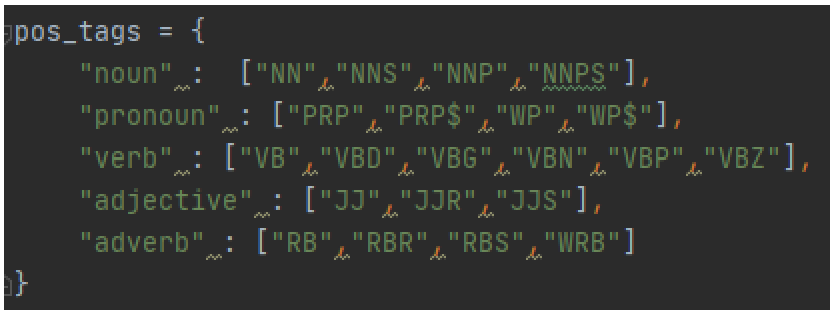
Polarity, polarity score, subjectivity and subjectivity score were all from the TextBlob library using the .polarity and .subjectivity attributes. Polarity and subjectivity were just used as labels for data visualisation and not used as a feature while the actual scores were used for training.

Character count, word count and punctuation count were all extracted using len() and some other string operations such as split(). For punctuation, I imported the punctuation list from String to be used as a checklist however, this only applies to English and not characters like the upside question mark. Word density was achieved by diving character count by word count.

Exclamations marks, question marks, ellipsis, emojis, URLS, hashtags, mentions were all found by regexes from re library (native to python). The following are the regexes used: "!", "?", "\.{3}", r"(:[^:]*:)", "http.*", #([0-9]*[a-zA-Z]*)+", @([0-9]*[a-zA-Z]*)+" respectively. To extract emojis, I would use the demojize() function on the tweet from the emoji library, then use the regex to find the emojis.

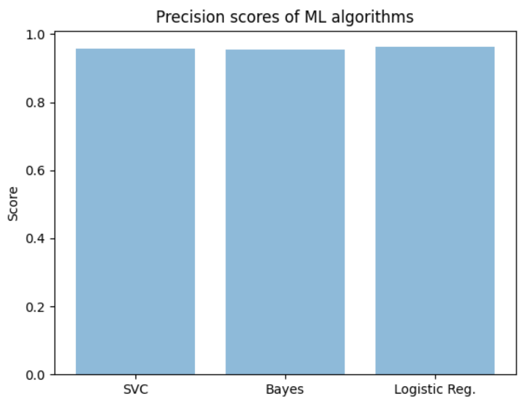
To extract the number of locations mentioned in the tweet, I downloaded a database from simplemaps (9) which provides free csv file of most large cities in the world and includes the country they are in and country abbreviations. I used this as a corpus and checked each token if they refer to a location and incremented the count appropriately. I used teachstarter as a source for disaster words (10) for my corpus and was used as the same way locations extractor.

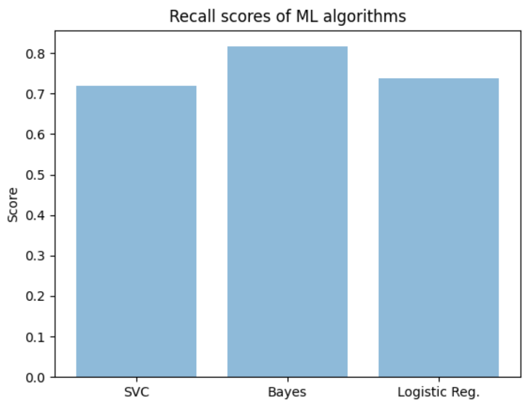
I extracted noun count, verb count, adjective count, adverb count and pronoun count via pos_tag() function from the nltk library. I also made a dictionary mapping string to a list of tags; the list of tags was from penn Treebank project (11).

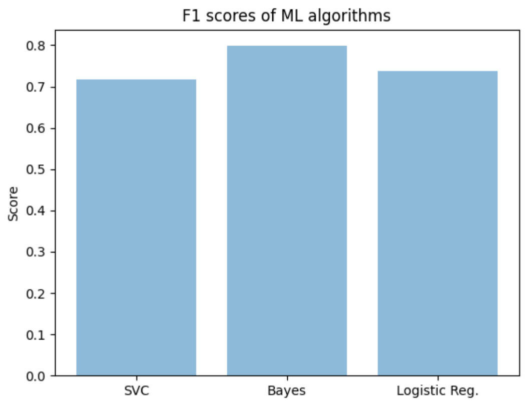

```
pos_tags = {
    "noun":  ["NN","NNS","NNP","NNPS"],
    "pronoun": ["PRP","PRP$","WP","WP$"],
    "verb": ["VB","VBD","VBG","VBN","VBP","VBZ"],
    "adjective": ["JJ","JJR","JJS"],
    "adverb": ["RB","RBR","RBS","WRB"]
}
```

With these new features, I fed them into a SVC (a support vector classifier from svm, sklearn) however I only managed to get up to 0.560 for my F1 score on the test split on the training set. This is barely better than just randomly guessing and I could not think of any more features to extract from the tweetText. I decided to leave this approach instead of optimising it since the F1 gain would be so low.

Instead, some of the approaches mentioned using different feature extractors such as term frequency inverse document frequency (tf-idf) which I ultimately implemented in conjunction with n-grams. The Tf-idf used in this approach was imported from sklearn. Tf-idf shows how often each term appears in the corpus (our tweetText is the corpus) multiplied with the inverse of the amount that the same term appears in other documents. This means if a word is used in multiple documents and is common, e.g. a stop word like "I", then its weighting will be very low and holds very little descriptive power. If a word has high occurrence in some specific tweets, then its weighting will be very high as it is very descriptive since only a few documents use that word and thus associations start to occur between class and word. In addition to Tf-idf, I use the n-gram parameter (1,2) to allow bigrams and unigrams to form from the tweetText. This creates both single- and 2-word phrases in "dictionary" (list of most popular words) to be used when analysing tweetText. I added bigrams since they preserve some information regarding context as it is a phrase and not a just a single word. This is the main reason I chose to add n-grams is to preserve contextual information which tf-idf and bag of words naturally do not contain. I then plugged tf-idf feature vector into 3 different ML algorithms (Logistic Regression, SVM and Multinomial Naïve Bayes) to see what kind of baseline performance each algorithm provides. The following are the initial results.







From this, I went with the Multinomial Naïve Bayes ML algorithm.

With this approach, I initially had an f1 score of 0.798 with default configuration of the Tf-idf constructor and Bayes.

Evaluation setup, Iteration & analysis

Noticing that the training dataset was not balanced, I used a total 9000 records for training which was split into 4500 real tweets, 2250 fake tweets and 2250 humorous tweets. This would ensure that we do not have any bias in the dataset when training the ML algorithm. The rest of the tweets formed the testing set. I decided not to use accuracy as a metric as the test split has only 509 real tweets and the rest were fake so a dumb algorithm could easily have a high accuracy. I decided to use instead the F1 score in combination with the recall and the precision.

Recall is all the correct positive results / all samples that should have been positive i.e. how many positive predictions were correct out of all samples with the positive ground truth. To get the recall result I used metric.recall_score() from sklearn . Precision is all the correct positive results / number of positive results predicted by classifier i.e. how many of the classifiers positive predictions are actually positive? To get the precision result I used metric.precision_score() from sklearn . F1 is the harmonic mean between recall and precision i.e average between precision and recall. To get the recall result I used metric.f1 _score(average="micro") from sklearn .

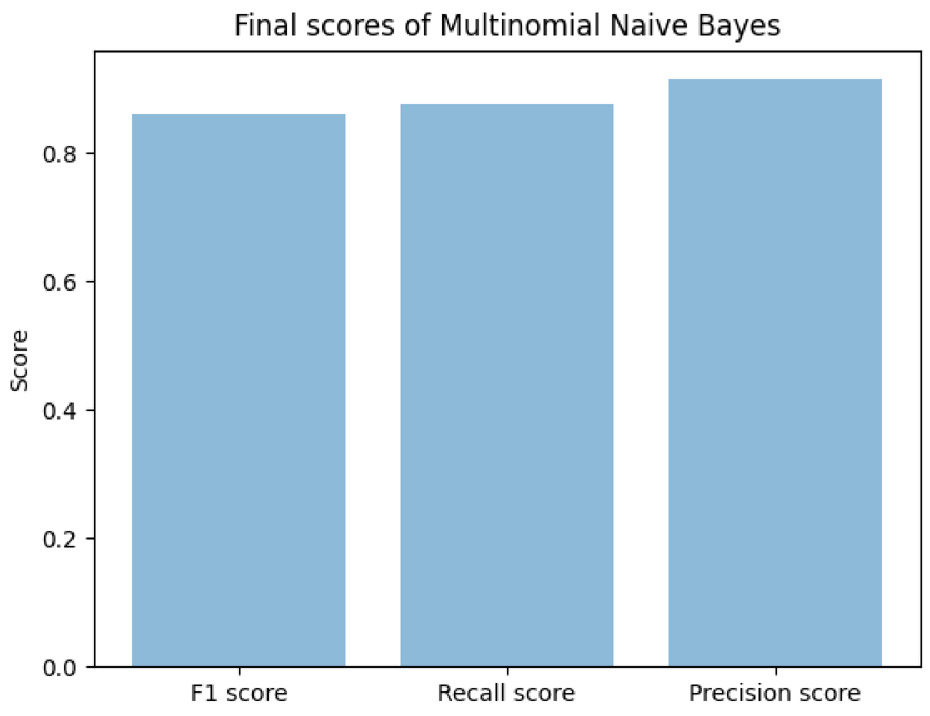
I then used grid search (from sklearn) with the following parameters: {"alpha":[0.0,0.1,0.2,…,1.0]} I also adjusted the number of max features used by Tf-idf simultaneously starting from 5000 to 50,000 in increments of 5000 features. I chose a maximum limit of 50000 features as the general trend was F1 scores were dropping as number of features was increasing.

There was no use of PCA to reduce dimensionality. The only factor affecting the dimensionality of the feature vector was the number of features being included.

The final parameters used were alpha=0.1 for Multinomial Naïve Bayes and ngram_range = (1,2), max_features = 5000, min_df = 1, stop_words = "English" for tfidfVectorizer.

Final f1 score (average = micro), recall and precision on the test set are as follows: F1 score = 0.861, Recall = 0.876, Precision = 0.915.

Conclusion



I should have invested more time in feature selection, I could have used some of the features in my first attempt and combined with my final approach, this could have boosted my results even further.

Although I decided not to translate any other languages into English, it may have been a good idea to do so because I think this is what caused the sharp peak in my “noun count” feature and it would have meant Tf-idf could have been applied to different languages, other languages made up a quarter of the dataset. This would have also made the implementation work in more areas of the world if it were deployed.

Also, I did not scale any of my features in the first approach so that might have made a difference to the overall success metrics of that approach.

References

1. **Rajdev, M, Lee, K.** *Fake and Spam Messages: Detecting Misinformation During Natural Disasters on Social Media.* s.l. : IEEE, December 2015.

2. **Hirlekar, V.V., Kumar , A.** *Natural Language Processing based Online Fake News Detection Challenges – A Detailed Review.* s.l. : ICCES, June 2020.

3. **Krishnan. S, Chen, M.** *Identifying Tweets with Fake News.* s.l. : iEEE, July 2018.

4. **Hassan, N. Y., Gomaa, W. H., Khoriba, G. A., Haggag, M. H.** *Supervised Learning Approach for Twitter Credibility Detection.* s.l. : iEEE, December 2018.

5. *Fighting an Infodemic: COVID-19 Fake News Dataset.* **Patwa, Parth, Sharma, Shivam, PYKL, Srinivas, Guptha, Vineeth, Kumari, Gitanjali, Akhtar, md Shad, Ekbal, Asif, DAs, Amitava, Chakraborty, Tanmoy.** s.l. : arXiv, 2020-11-06, Vol. arXiv:2011.03327 [cs].

6. **Dyson, Lauren, Golab, Alden.** *Fake News Detection Exploring the Application of NLP Methods to Machine.* s.l. : University of Chicago, December 2017.

7. **Helmstetter, S, Paulheim, H.** *Weakly Supervised Learning for Fake News Detection on Twitter.* s.l. : iEEE, August 2018.

8. **Dey, A, Rafi, R Z., Parash, S. Hasan, Arko S K., Chakrabarty, A.** *Fake News Pattern Recognition using Linguistic Analysis.* s.l. : ICIEV and iclVPR, June 2018.

9. **World Cities Database. *simplemaps.* [Online] Pareto Software, 2010. [Cited: 13 1 2021.] https://simplemaps.com/data/world-cities.**

10. Jill. Natural Disaster Word Wall Vocabulary. *Teach Starter.* [Online] Teach starter, 26 10 2013. [Cited: 13 01 2021.] https://www.teachstarter.com/gb/teaching-resource/natural-disaster-word-wall-vocabulary-gb/.

11. Alphabetical list of part-of-speech tags used in the Penn Treebank Project:. [Online] [Cited: 12 1 2021.] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html.

12. Manzoor, S I., Singla, J. *Fake News Detection Using Machine Learning approaches: A systematic Review.* s.l. : ICOEI, April 2019.