Relevant Features and Models in the Detection of Malicious COVID-19 Tweets

CM3203: One Semester Individual Project – 40 Credits
School of Computer Science and Informatics - Cardiff University

Author: Izabele Bauzyte
Supervisor: Amir Javed
Moderator: Bailin Deng

14 May 2021

Abstract

With the explosion of technology usage spurred by the COVID-19 pandemic, malicious actors worldwide have taken this opportunity to create and spread new types of coronavirus-related malware and scams, relying on social media networks such as Twitter to quicken the spread. This paper investigated a set of processed tweet features, including: named entity labels, parts-of-speech, emotion and sentiment analysis, textual attributes, and tweet account features, to determine which features are most helpful in discovering tweets containing malicious URLs. It was discovered that the most telling features were text-attribute and account features, while parts-of-speech, entity labels, and sentiment analysis proved to be less helpful. This paper also tested a small number of different models to determine which models were able to classify the malicious/benign tweets most accurately, of which the random forest classifier and a stacked meta-classifier encompassing about half a dozen other models performed best, while the SVM and multi-layer perceptron models performed the worst.

Acknowledgements

I would like to thank my supervisor Amir Javed for pointing me in the right direction throughout this project and being an invaluable source of information, as well as providing helpful feedback on my report drafts.

Table of Contents

Table of Figures

1 Introduction

1.1 Motivation

In the 7 week span between March 9th, 2020 and April 26th, 2020, over 86,000 newly observed hostnames relating to COVID-19 were classified as malicious or high risk out of a total of 1.2 million hostnames analyzed [1]. These are sites that contained command and control malware distribution, phishing, had been previously classed as malicious websites, were hosted on bulletproof ISPs, or shared domains with malicious sites. Percentage-wise, this would be about 7.17% of analyzed sites classed as malicious. Additionally, researchers analyzing the credibility of recently created coronavirus-related domains could not establish the authenticity of around 80% of those sites which had presented themselves as official government websites [2]. Links to "Corona Antivirus" infect users with a BlackNET RAT malware which can execute harmful scripts and even launch DDoS attacks. Links to pages masquerading as official government websites collect users' personal information and bank details [3]. The list goes on.

Malicious information and links have a way of appearing on social media networks, through the untiring efforts of bad actors worldwide. As the pandemic progresses, and malicious actors continue to take advantage of individuals looking for COVID-19 information or assistance on the web, it is important to look for trends that indicate malicious activity and help mitigate the damage done to victims.

1.2 Project Aim and Scope

The aim of this project is to gather evidence of malicious activity on an online social media platform (Twitter), and to identify key features that could be used to distinguish between malicious and benign posts via supervised machine learning. Tweets will first be collected using the Tweepy Python package, and any URLs contained in the tweet will be analyzed using the VirusTotal API, which will reveal whether links contained in the posts were malicious or benign.

This data collected from Twitter can then be processed into a variety of features, such as parts-of-speech, named entity labels, text attributes, and tweet account data, among others. This processed data can be used to train a classification model which is capable of labelling tweets as either 'malicious' or 'benign', based on features of the tweet or account. The primary goals throughout will be to discover the most important features used by the classification model, as well as to discover which supervised machine learning models are best suited to malicious tweet classification. Although there has been large scale analysis of malicious tweets done in the past, this project specifically focuses on COVID-related tweets and data.

Additionally, if time permits, the creation of a similar classification model will be attempted for the purpose of classifying tweets containing misinformation, as well as the discovery of important features in the classification of misinformation.

1.3 Intended Audience

Although the final deliverable of this project includes a model trained to classify Twitter posts as either malicious or benign, as well as a script that takes an input files and makes predictions for all samples in the file, this model does not come anywhere close to the malware detection services Twitter has already put in place for URL monitoring.

This project is primarily intended as an aid and information piece for other researchers and students, particularly for feature and model selection in similar projects. The data collection and processing scripts used throughout have been written with flexibility, allowing their continued use through the addition of API keys for the various services used throughout the scripts. These scripts may be made available on a public repository for viewing and further use.

2 Background

2.1 URL Classification

The recent prevalence of COVID-19 malicious attacks and domain registrations has been subject to investigation by academics and researchers. Particularly in the realm of malicious URL classification, models have been built which use URL features exclusively to determine maliciousness. These features may include: length of domain names, count of hyphens, and count of numerical characters [4].

In 2013 a paper was published on real-time malicious URL classification, a system called WARNINGBIRD. This system also primarily used URL features and looked at URL redirection to determine if suspicious activities were taking place. The researchers compared their detection system to that of Twitter's and found that, on average, their system was able to detect malicious accounts 13.5 minutes before Twitter did [5].

2.2 Feature Selection

A lot of research has been put into what features should be used as input for machine learning models. The value of certain features often seem to be linked to the type of data being collected, and feature importance seems to differ from model to model, often depending on the type of dataset it was trained on. In Barbosa and Feng's analysis of twitter sentiment detection using biased/noisy data [6], text meta-features (such as parts of speech) and text syntax features were found to be good indicators of intent when building a subjectivity classifier, better even than unigram features. Additionally, POS (parts-of-speech) features are also shown to be useful in customer feedback data, in terms of sentiment analysis [7], although POS features were simultaneously found to be unhelpful in another analysis of Twitter Sentiment conducted by Go et al [8]. The usefulness of POS features is analyzed in this project. Additionally, polarity of parts of text is known to be used effectively in twitter sentiment analysis [9].

In addition to text syntax features and POS features, text labels will also be used to determine tweet maliciousness, using named entity recognition. Labels are used to identify different types of named entities and differentiate what type of named entity it is. Little research could be found on the topic of using named entity labels in machine learning classification, but there is some reasoning behind the inclusion of label features: the names of certain people, organizations, or even events may trigger different types of emotional responses from users, whether the response is fear, curiosity, anger, or any other type of emotion. In fact, the World Health Organization has published guidelines on naming diseases [10], and has cautioned exluding names, places, animals, and cultures from the naming scheme, likely due to the negative associations and stigmatization that occur from such naming. For this reason, some research was put into which hashtags were currently being used to describe the virus, particularly focusing on tags that might contain location-specific names. Label features such as these will also be included in the list of features to be analyzed during model creation, in the hopes that named entity labels could be useful in classifying malicious/benign tweets. Namely, could individuals be more likely to click on a link if they experience a strong reponse to a named entity that is contained in the tweet text? Named entity unigrams might have also worked well here, but implementation of unigrams could go outside the limited time schedule of the project. Unigrams refer to a 1-word sequence of text, and could be implemented as a binary flag of whether that unigram is discovered in the text. The results of this research will be referred to when selecting features as input for training a model.

2.3 Wider Context

This project expands upon this idea of using URL features, and relies on data in the broader context of a social media platform such as Twitter, using features available on social media posts, in tandem with URL features. The main question is: can we successfully identify malicious COVID URLs using expanded features such as POS, named entity labels, textual attributes, and account features?

In terms of feature selection, there is much research done on the selection of suitable features to be used as input for machine learning models. Many proposed models are known to use text syntax features, in addition to text polarity features such as sentiment. POS features have high popularity as well, but suffer from disagreeing conclusions as to their usefulness. These features will be looked into, as well as features such as URL length, discussed in section 2.1. Named entity labels will also be proposed as machine learning features in this project, and results will be analyzed to determine if labels are seen as important features by the models in terms of classification.

This project differs from similar research done in the field. The WARNINGBIRD research addressed in section 2.1 writes off account features as unreliable, particularly since hacked accounts spreading malicious links might primarily display features of a benign account, as well as the fact that bad actors are becoming better about creating accounts that blend in more with normal user accounts. But account features would not be the only features used in this paper. One of the main ideas of the WARNINGBIRD research was that bad actors continuously evolve as their techniques are discovered and made public. Features must be continuously anaylyzed to see if changes occur. Additionally, the WARNINGBIRD research was conducted around 2013, and with the advent of COVID-19 it is not unlikely to assume that with the explosion of technology use, many bad actors have used this as a chance to spread malware for the first time. Account features will be included to see if bad actors continue to sufficiently blend in with other accounts, or if new players in the COVID-19 malware game can be determined from account features.

3 Methodology

Before the project was started, a rough timeline was created which dictated weekly activities from weeks 1 through 12.

Data Collection and Processing	
Week 1	Initial Plan, background research on pulling data from twitter and determining malicious links [create necessary dev. account]
Week 2	Start pulling data, background research on similar projects and also deciding between using Virus total or Cuckoo Honeypot for URL analysis
Week 3 - 5	Set up data store, import twitter data, pre-processing of data set by connecting with Virus total API/Cuckoo Honeypot and segregating malicious and benign URLs Create a python/Java based program that will preprocess data and create a machine learning input file containing a set of features (features will be identified in week 2)
Building a Supervised Machine Learning Model	
Week 6 - 8	Building supervised machine learning model in python/java to categorize tweets into either “malicious” or “benign” based on user features
Easter weeks 1-3	[CATCH UP] on previous weeks work if falling behind, otherwise maybe try to push ahead
Week 9	[IF EXTRA TIME] Try to reuse code to now categorize malicious tweets into those that contain misinformation or not.
Week 10	[IF EXTRA TIME] Try to reuse code to now categorize malicious tweets into those that contain misinformation or not.
Final Report Work	
Week 11	Finalize documentation based on project notes, analysis for final report
Week 12	Finalizing final report

This timeline was followed pretty closely. When it became clear that no automated fact checking tools would be available for the misinformation categorization in weeks 9 and 10, more time was spent tuning classification models. Consequently, only 3-4 full days were spent doing manual fact checking, but not enough disputed claims could be found through manual search, and no misinformation classification model could be trained.

3.1 Data Collection

To start off the process, tweets and data from Twitter were collected by using the Twitter developer API. An application is required to gain access to the API. This process was started early to ensure that the application was approved by the time tweet collection was started. An application was completed and sent on Febrary 8th, and an approval was received by the 9th, a very quick processing time.

Next, a way was needed to determine if the URLs included in each collected tweet were actually malicious, so that these malicious/benign labels could be used to train a model. Several options were available for this task, two chief contenders being Cuckoo Honeypot [11] and VirusTotal [12]. In the end, VirusTotal was chosen for URL processing due to its extensive documentation and examples, which Cuckoo Honeypot was missing. VirusTotal has both web and API interfaces, and allows users to submit and analyze files and URLs for malicious content. It claims to use 70+ different antivirus scanners and URL/domain blacklisting services.

Access to VirusTotal API was required to determine if a certain sample should be labelled as ‘malicious’ or ‘benign’ during the training and testing phases. The VirusTotal public API is limited to 500 requests per day at a rate of 4 requests per minute. Initially, several accounts were created using different email accounts to obtain a limit higher than 500, but unfortunately, only the initial account was able to successfully call API requests. Access to the VirusTotal academic API was later granted after a successful application using an account with a Cardiff University email. The quota was increased to 1000 requests per minute (20,000 per day, 600,000 per month).

For this reason, the initial collection class, called TweetProcessor, was designed for use with multiple API keys, which could be rotated after the previous API key had reached either the specified minute-limit or the daily limit. VirusTotal returned a status code response of 200 to specify the request had either been retrieved or queued, and 204 to specify that some quota limit had been reached. TweetProccesor stored a parallel array of the same size as the number of API keys to be used, with each position having the following 3 values: ‘nolimit’, ‘minlimit’, ‘dailylimit’. A value of ‘nolimit’ indicated that the current VirusTotal key being used had not yet reached either it’s minute or daily limits and could be used again for another query. ‘minlimit’ indicated that the current VirusTotal API key had returned a status value of 204 once in a timespan of a minute, and could not be used successfully in the next minute. A value of ‘dailylimit’ indicated VirusTotal had returned a status code of 204 two or more times in a row, spanning several minutes, and that the daily limit for this API key had likely been reached. Thus requests would be sent off to VirusTotal until all URLs collected had been processed, or all VirusTotal keys had been used up to their daily limit, more preferably the former. This process is demonstrated in Figure 1.

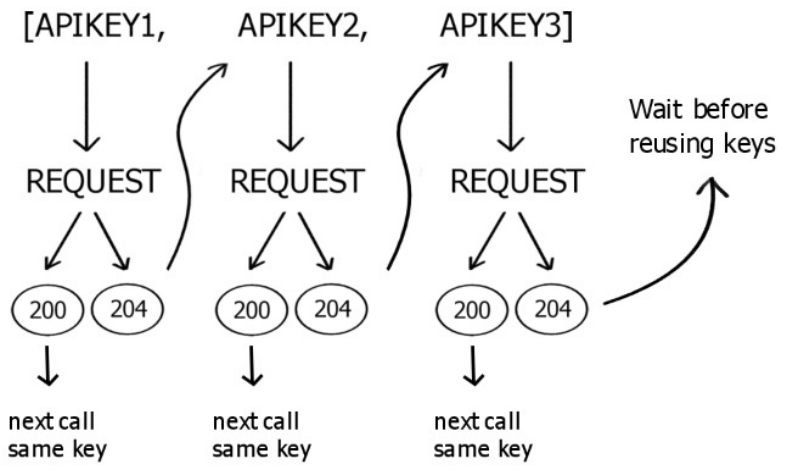

Figure 1: API Request Key Architecture

However, after being granted access to VirusTotal's Academic API, this system became somewhat redundant with the 1000 request per minute quota. As soon as the results for a specific URL request were received, the Tweet object data and VirusTotal results were written to file, so that if the script were to fail later, all objects processed thus far would be retained on the disk.

The data collection script would call Tweepy to collect several thousand Tweet objects, depending on the filter. The URLs in these objects would then be sent off to VirusTotal, after which the results would be received or the URL would be queued for processing. After all requests were sent to VirusTotal, the script would wait for 100 seconds before beginning to collect the requests that had been previously queued. The rest of the results would be written to a file, and the process would terminate.

The tweet objects collected by Tweepy include the text data of the tweet, which can have many different components. Hashtags are used to label a tweet as pertaining to a certain topic and begin with the '#' symbol. Data collected in this project all related to some type of COVID-19 hashtag, such as '#covid', '#coronavirus', and so on. User mentions begin with the '@' symbol and notify users who are included in the body of the tweet. In addition to these, users can include images, videos, polls, URLs, and symbols. Besides the textual and media data contained in the body of the tweet, Tweepy also includes account data such as number of friends, followers, favourites, etc., as well as tweet metadata, such as creation date and source, among others, which are returned with the rest of the Tweet object.

Tweepy also allows filtering by various tweet characteristics. The base filter specified was "Filter:links -Filter:retweets", which would collect all tweets containing links, and exclude any tweets which were retweets, which is essentially a repost or share of the original tweet by another user. Along with this base filter, several hashtags were appended to ensure the tweets were COVID specific. Unique tweets were collected by checking the tweet IDs against a dictionary of all tweet IDs that have been collected. Since the aim was only to collect unique tweets, varied hashtags were used to collect as many covid related tweets as possible per day.

Initially, 4 main hashtags were chosen to perform the data filtering with, with others added along the way. These were: covid, covid19, coronavirus, and corona. This was due to their popularity on the hashtag aggregator site Hashtagify [13] in the beginning of March 2021.

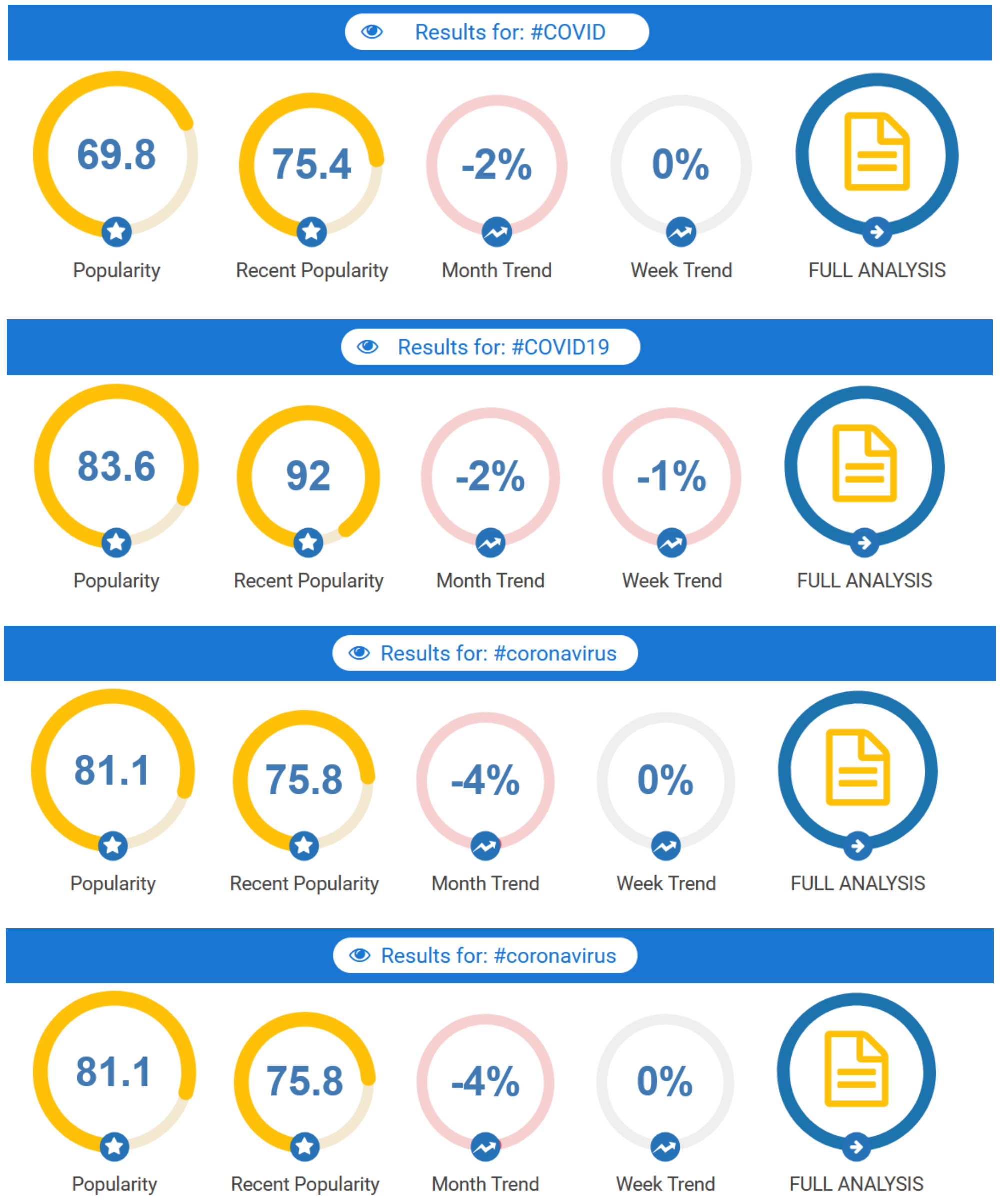

Figure 2: Popular COVID-related Hashtags in March 2021

There exist many possible names the coronavirus is known by, with some being used in an offensive or derogatory manner, these are important to look into as well, as they could possibly reveal data patterns about malicious links. The following hashtag filters were used at least once to determine the hashtag's popularity.

1. coronavirus
2. covid
3. covid19
4. covid_19
5. covid-19
6. corona
7. 2019-nCoV
8. chinavirus
9. ccpvirus
10. wuhanflu
11. kungflu
12. rona
13. wuhanvirus

More accurate data logging and breakdowns were put in place around 2 days after data collection began, meaning samples collected on days 1 and 2 may be missing from some of the breakdowns by hashtag type. There were several key trends discovered throughout the data collection phase which will be covered in more detail in section 4.1.

A script was created which collected and analyzed tweet links using virus total. The processing script was written to check whether a URL had already been checked by VirusTotal, otherwise it queued the url to be checked by VirusTotal later and continued to process the next URL in line. At the very end of processing, the script would retrieve all queued URLs from VirusTotal. It would attempt retrieval for a specified number of times before finishing the processing step.

Language information was not initially collected during the tweet processing step, but was later analyzed after all data was collected; the language of each tweet was detected using a package from Python called langdetect. Language info was looked into, to determine if some regions had a higher ratio of malicious/benign tweets.

3.2 Feature Selection

From the background research conducted in section 2.2, these features will be implemented: parts of speech such as noun, verb, adverb, etc., as well as textual attributes such as number of hashtags, number of emojis, average sentence length, etc.

Emotion and sentiment features will also be used, to determine if a certain emotion or sentiment is commonly seen in malicious tweets. For the final set of features, generic tweet and user features will also be used; this includes fields such as days since account creation, url length, friends count, etc.

Some more advanced data processing methods were not used, in order to keep with the schedule of the project. There is some research on the success of emoticon processing, finding and counting mispellings in text, and the analysis of acronyms used in text [9]. These were not included due to limited time constraints when writing the feature extraction scripts.

3.3 Feature Extraction and Processing

All tweets were first translated into english using google's translation API in Python, and feature extraction was performed on translated texts. The following features were extracted from the data collected from tweets, split into 5 different categories: LABEL, POS, EMOTION & SENTIMENT, ATTRIB, TWEET.

3.3.1 Labels

For the text of each tweet, the instances of each type of named entity label were counted, these were:

PERSON: People, including fictional
GPE: Countries, cities, states
NORP: Nationalities or religious or political groups
ORG: Companies, agencies, institutions, etc.
EVENT: Named hurricanes, battles, wars, sports events, etc.

Natural language processing was accomplished using SpaCy's en_core_web_sm model. (Definitions come from the OntoNotes Release 5.0 Documentation [14]), which is used as a source of the en_core_web_sm model [15].

3.3.2 POS

The following parts of speech were used as features, with instances of each POS being counted throughout:

POS_ADJ	POS_ADV	POS_INTJ
POS_NOUN	POS_PROPN	POS_VERB
POS_ADP	POS_AUX	POS_CONJ

POS_DET	POS_NUM	POS_PART
POS_PRON	POS_SCONJ	

3.3.3 Emotion & Sentiment

For sentiment analysis, the NRC Emotion Lexicon [16] was used, which maps word associations to 8 emotions (anger, anticipation, disgust, fear, joy, sadness, surprise) and 2 sentiments (positive, negative). For analyzing the text sentiment, the tweets were first cleaned. First tweets were converted to lowercase, URLs and usernames were removed while hashtags were retained on the assumption they might contain relevant information. Punctuation and stop words, which are words that do not add much meaning to a sentence, like 'it', 'is', 'the', were removed from the text. And finally, the remaining text was put through a lemmatization process, which will remove any word endings and only keep the base form of a word. For example, 'player' will be converted to 'play'.

EMOTION_anger	EMOTION_anticipation	EMOTION_disgust
EMOTION_fear	EMOTION_joy	EMOTION_sadness
EMOTION_surprise	EMOTION_trust	SENTIMENT_negative
SENTIMENT_positive		

3.3.4 Attributes (textual)

Attribute features consisted of sums of various characters, as well as word and sentence averages.

ATTRIB_avg_sentence_length	ATTRIB_avg_word_length
ATTRIB_length_of_tweet	ATTRIB_number_of_capital_letters
ATTRIB_number_of_emojis	ATTRIB_number_of_exclamation_marks
ATTRIB_number_of_hashtags	ATTRIB_number_of_periods
ATTRIB_number_of_question_marks	ATTRIB_number_of_sentences
ATTRIB_number_of_special_characters	ATTRIB_number_of_uppercase_words
ATTRIB_number_of_user_mentions	ATTRIB_number_of_words

3.3.5 Tweet Data

Finally, some of the original tweet data, as well as account features associated with that tweet data were used as features:

TWEET_days_since_tweet_creation	TWEET_retweet_count
TWEET_favorite_count	TWEET_verified

TWEET_followers_count	TWEET_friends_count
TWEET_listed_count	TWEET_favourites_count
TWEET_statuses_count	TWEET_days_since_account_creation
TWEET_url_length	

For training and testing purposes, the binary classification of malicious/benign was also used, where malicious reponses are flagged as having 1 or more positives from the antivirus scanners and URL/domain blacklisting services used by VirusTotal. This makes for a total of 54 features, not including the training class identifier of ‘malicious/benign.’

3.4 Balanced Dataset Reasoning

There is evidence that models trained on balanced datasets perform better in the classification [17][18]. Experiments have also shown that between the two methods of oversampling and the no compensation approach, there is no clear preference in terms of metrics [17]. For this reason, a balanced dataset was used instead of the naturally unbalanced dataset that is seen throughout the data collection process. About 10% of collected URLs were flagged by VirusTotal as malicious throughout data collection, and to generate a balanced dataset, random undersampling of the majority class was used.

3.5 Model Selection

After data collection and processing, the training phase could begin. Around 13,000 tweets labeled as ‘malicious’, and 13,000 randomly sampled tweets labeled as ‘benign’ were processed to keep a balanced dataset.

There are two possible methods here: to pick the best classifier out of tested classifiers based on models which are known to perform well, or to combine a group of tuned models together into a meta-classifier, if it performs better than any singular model.

Several techniques can be used to combine models; bagging, boosting, and specifically, stacking. Using a stacked classifier, all models could be combined into a classifier which could have a higher accuracy than any of the original models. The more varied the assumptions of each of the different models were, the better the stacked classifier could perform. It specifically works by taking the predictions of each model and creating another model from these using a meta-classifier.

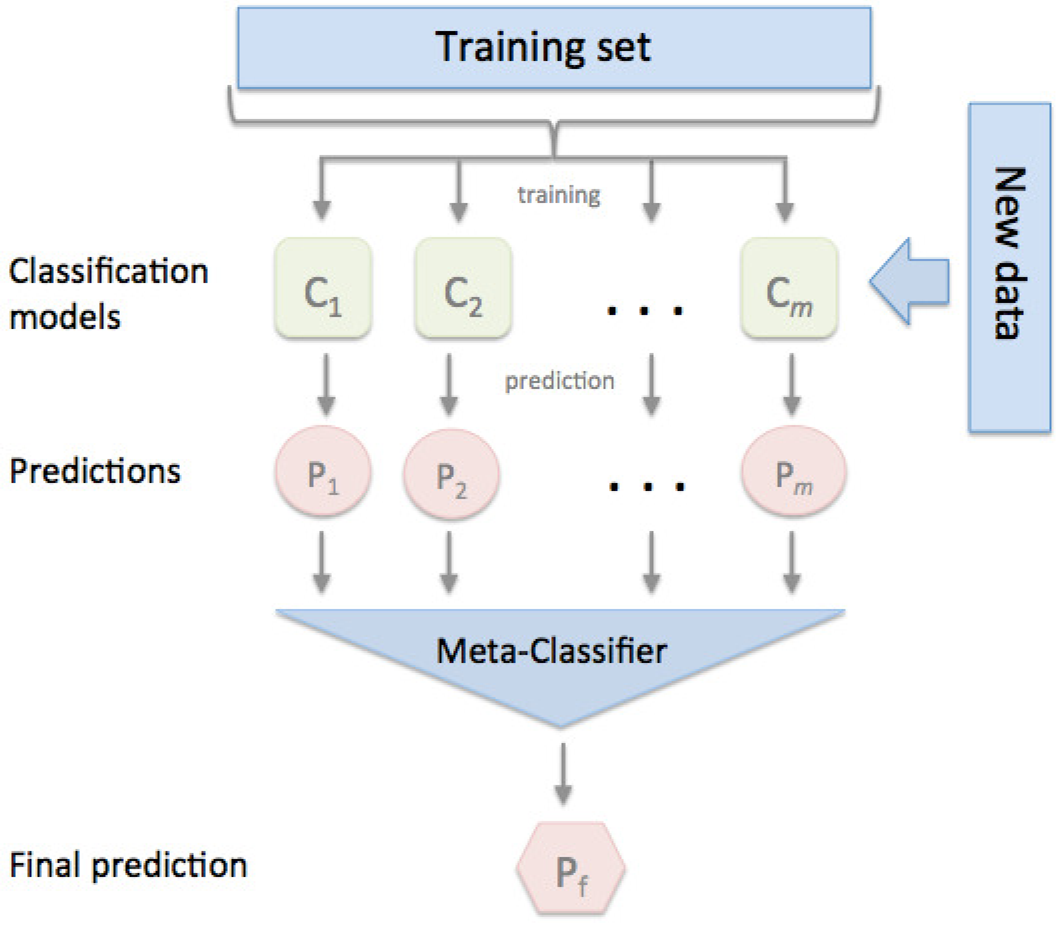

Figure 3: Stacked Classifier Architecture [19]

The following are classifier models available in the Python sklearn package [20], which is the main package used for classification in this project. sklearn was chosen because of it's simple model creation scheme and the author's familiarity with the software.

- DecisionTreeClassifier
- RandomForestClassifier
- MLPClassifier
- GaussianNB
- SVC
- KneighborsClassifier

Decision Trees

Decision trees work by splitting the dataset recursively using rule-based behaviour. At each split, the decision tree will calculate the best split that reduces the impurity of the dataset and try to move the greatest number of samples into their respective classes. Specifically, the sklearn implementation of decision trees uses an optimized CART algorithm [32]. Decision trees will be used in this project for their computational efficiency and easy interpretation. The internal structure can be visualized and feature importance can be easily extracted. Using the python graphviz package, visualizations of rule based decision making can be seen in the internal structure of the decision tree. Although they are fast and easy to use, decision trees are also prone to overfitting, especially if the max depth parameters of the tree isn't limited, in which case it will keep splitting nodes until all data points have been categorized.

Random Forests

Random forests use a number of decision trees and fit them on sub-samples of the dataset used. They use averaging to improve the accuracy of prediction and control overfitting [33]. There is evidence that bagging classifiers, such as the Random Forest Classifier, outperforms almost all cases of singular classifiers. Furthermore, both bagging and singular classifiers can be outperformed by boosting, but only if the dataset does not suffer from excessive noise [21]. For this reason, random forests were included in model selection, since they are likely to perform well. In this particular case, the dataset is bound to be very noisy; consisting of many different writing styles, languages, poor translation, etc., and hence, boosting was excluded, while bagging was not.

MLPClassifier

The multi-layer perceptron uses a number of hidden layers to discover relationships between the data. The sklearn implementation of MLPClassifier uses LBFGS or stochastic gradient descent to optimize the log-loss function [34]. Neural nets have been shown to perform well on complex relationships, while also not being prone to overfitting [22]. Since the dataset is assumed to be very noisy due to different writing styles, languages, etc., the MLP classifier will be added to the list of models to evaluate.

In terms of neural network training/testing, the dataset size of 26,000 is on the lower end in regards to size. Dataset size is one of the most influential factors of neural network performance [23], and has been shown to require at least 30d(d+1) training samples, or 60d(d+1) samples to achieve near optimal performance [24], where d is the number of input features. Going off this data, the MLP classifier will be a good fit because it is not prone to overfitting, but it will still not be close to performing optimally, due to small dataset size.

GaussianNB

The GaussianNB classifier is from a family of probabilistic classifiers using Bayes' Theorem. In NB, each feature is counted as being independent of every other feature. Gaussian NB is used when the input features have continuous values and follow the gaussian distribution [35]. NB is known to perform well when the inputs have high dimensionality. And according to sklearn, it may lead to better generalization in high-dimensional spaces [25].

SVM/SVC

SVMs are a classification algorithm that create planes in multi-dimensional space, and classify new samples by determining their position in this space. The sklearn SVC (C-Support Vector Classification) uses the libsvm implementation [36]. SVMs are also supported by sklearn and said to perform well in high-dimensional spaces, just as naive bayes.

KNearestNeighbors

K-nearest neighbors is a classification algorithm which utilizes and memorizes the entire training set to make predictions. It determines the similarity of new inputs using it's stored existing data, and outputs the classification result based on some distance metric. The sklearn implementation of K-nearest allows choice between the Minkowski (default), Manhattan , and Euclidean distance metrics. The KNearestNeighbors algorithm is said to perform well on non-textual data with less than 100k samples [26], according to sklearn.

StackingClassifier

The stacking classifier was also included in the list of classifiers to be evaluated. By looking at many different models that make different assumptions, ensemble classifiers, such as the stacking classifier, can determine where particular models make good predictions and where they fail, and therefore understand the spaces where each model performs best, combining them according to these metrics. By combining algorithms that look at data in different ways, i.e. rule based, statistical, neural network based thinking, different breakdowns of the data may be found.

3.6 Model Creation and Tuning

After the initial models were selected, their parameters were tuned to ensure optimal performance. Three primary metrics were used to determine the performance of models during tuning and prediction: Accuracy, precision, and recall. These three metrics make use of the confusion matrix, which is used to measure the performance of binary classifiers using the actual and predicted class values. Four values are used: true positives (actual true, predicted true), false positives (actual false, predicted true), false negatives (actual true, predicted false), and true negatives (actual false, predicted false).

Accuracy measures how many samples are correctly identified (true positives and true negatives) by the classifier, out of all the samples in the dataset. Precision measures how many samples were actually positives, out of all the positives that the classifier identified. Recall measures how many true positives the classifier was able to discover out of all samples that were actually positives.

Accuracy = (TP + TN) / (TP + FP + TN + FN)
Precision = (TP) / (TP + FP)
Recall = (TP) / (TP + FN)

DecisionTreeClassifier

One of the main parameters to tune in a decision tree is the max depth. Unbounded, the original models reached depths of 28 layers, their accuracy dropped, and they began to overfit the data. The internal structure begins to fit to individual data points, shown in Figure 4.

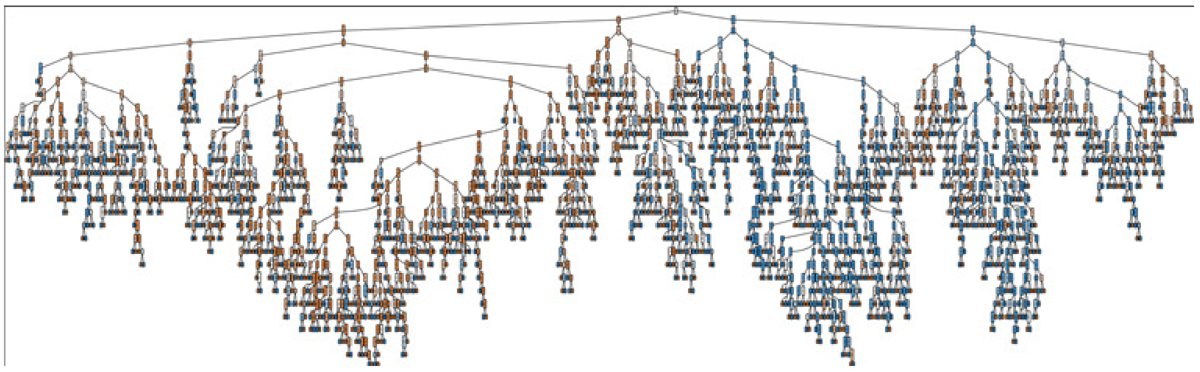

Figure 4: Decision Tree Structure

After varying the values of the maximum depth parameter, a final value of 10 was chosen. At a maximum depth of 10, the model seemed to reach optimum accuracy and precision, before dropping down at higher depth values.

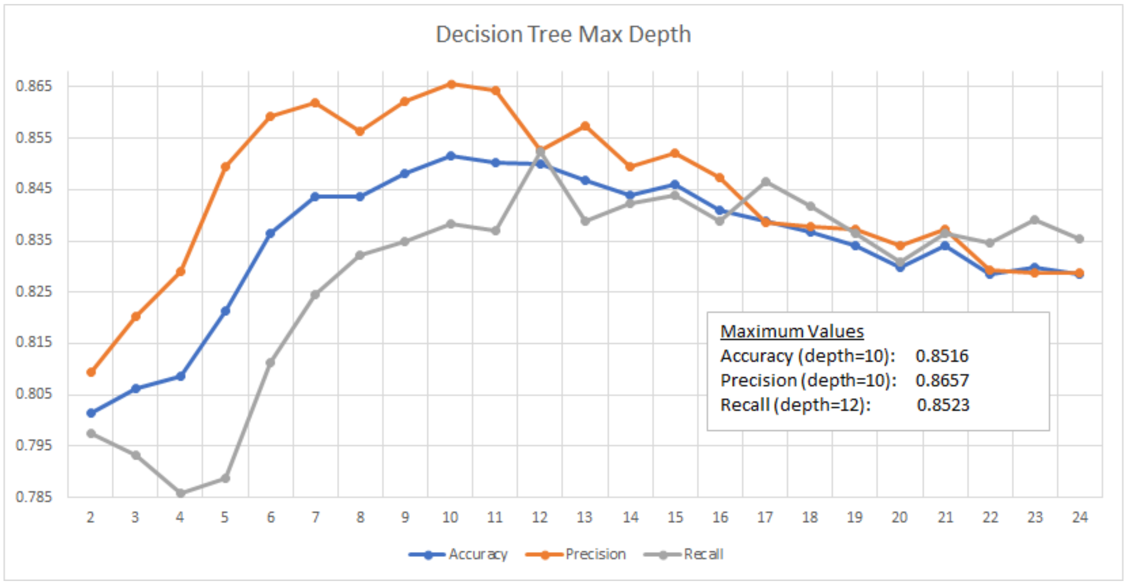

Figure 5: Decision Tree Performance at Variable Max Depth

Next, there is some argument for attempting to use the 'random' splitter instead of the 'best' splitter used by default, especially when the data doesn't have hundreds of features [27], as is the case here. The 'random' splitter also doesn't have the computational overhead of computing the best split.

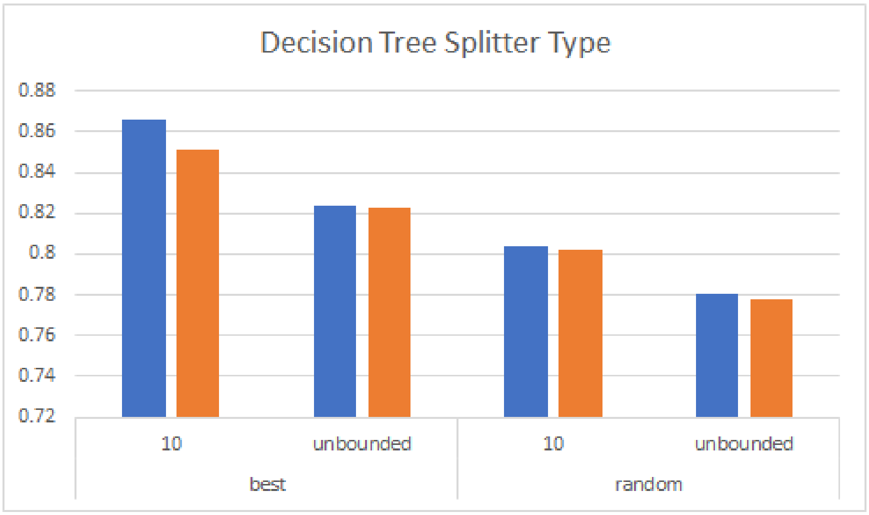

Figure 6: Decision Tree Performance at Variable Splitter Type

After 10 runs of each splitter type (5 with 10 max depth, 5 with unbounded depth), the 'best' splitter type performs better. Thus the final tuned decision tree will be run with max_depth = 10, and splitter = 'best'.

RandomForestClassifier

One main parameter to tune in RandomForest (which uses DecisionTrees by default) is the number of estimators used in the forest. After some tuning, a general positive trend is established between the number of estimators and evaluation metrics, with the growth rate of metrics beginning to slow around 15 estimators. In a time/accuracy tradeoff, a value of ~1 second is chosen as a cuttoff, which occurs at 17 iterators.

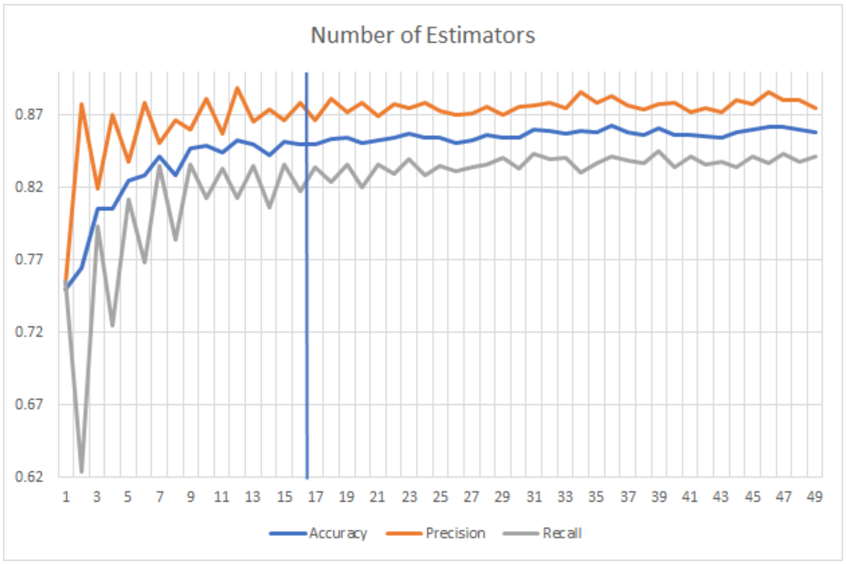

Figure 7: Random Forest Performance at Variable Number of Estimators

The maximum depth parameter is used to determine the maximum depth that all of the estimators in the random forest can reach. At depth = 18, performance values reach a high, after which they continue to drop or stay below the depth = 18 value for several rounds.

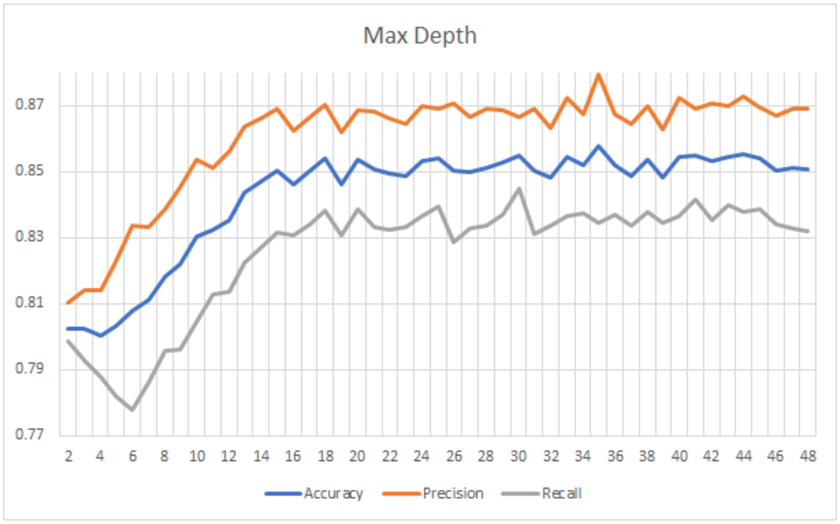

Figure 8: Random Forest Performance at Variable Max Depth

The final tuned random forest will consist of num_estimators = 17, and max_depth = 18

MLPClassifier

After tuning values for the mlp classifier, varying values for the max_iterations and hidden_layers parameters showed either an extremely small upward trend, or no clear trendline at all; these values were left up to the algorithm as the default values.

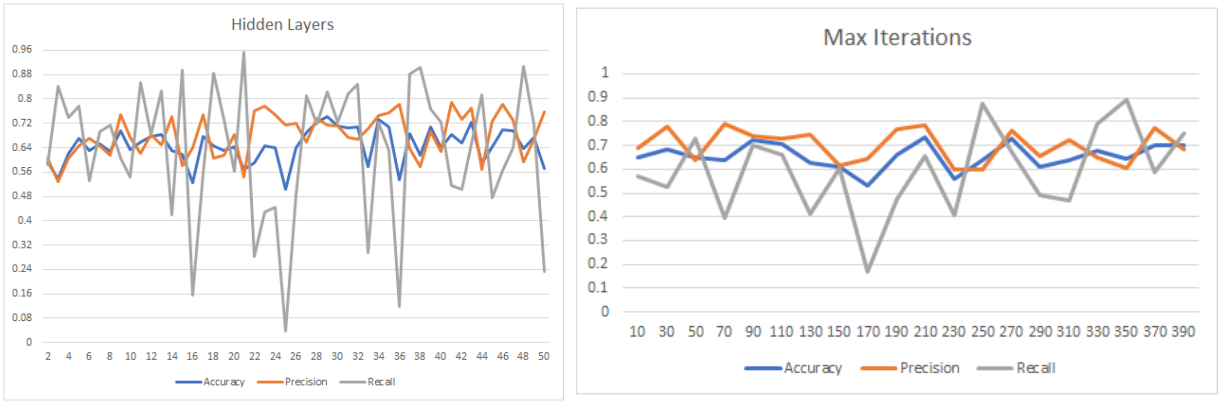

Figure 9: MLP Performance Hidden Layers and Max Iterations

The activation function for the hidden layers, the weight optimization solver, and the learning rate schedule for weight updates were also selected based on experiments with different values. In the end, the following parameters were chosen: activation=relu, solver=adam, learning rate=constant.

SVC

The SVC implementation could be impractical beyond tens of thousands of samples, depending on the dataset. The LinearSVC model is sometimes recommended in its place [28]. For this reason, tests were run between the two models, to determine which one would be a better fit. The SVC model was chosen, due to its better accuracy and precision averages.

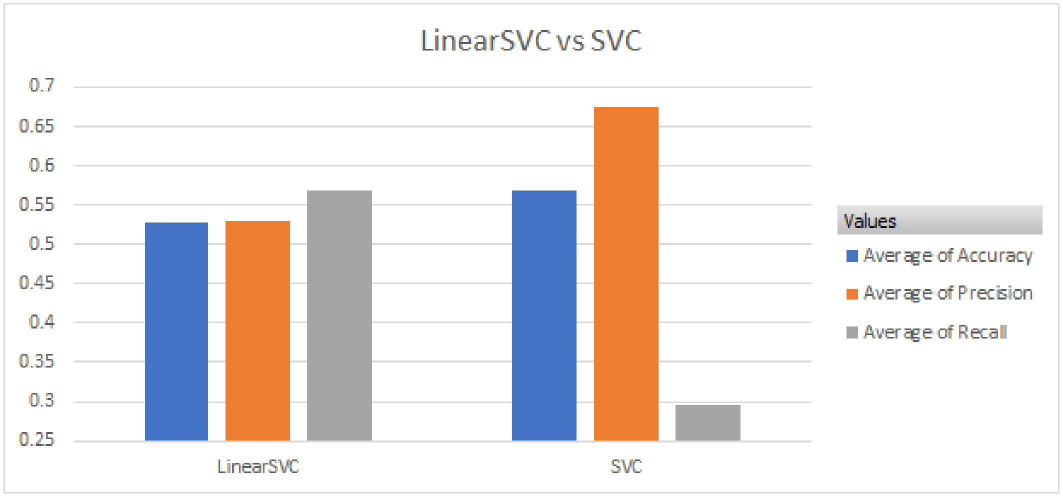

Figure 10: LinearSVC vs SVC

Experiments were also run on the kernel type to be used in the algorithm, and the regularization parameter c.

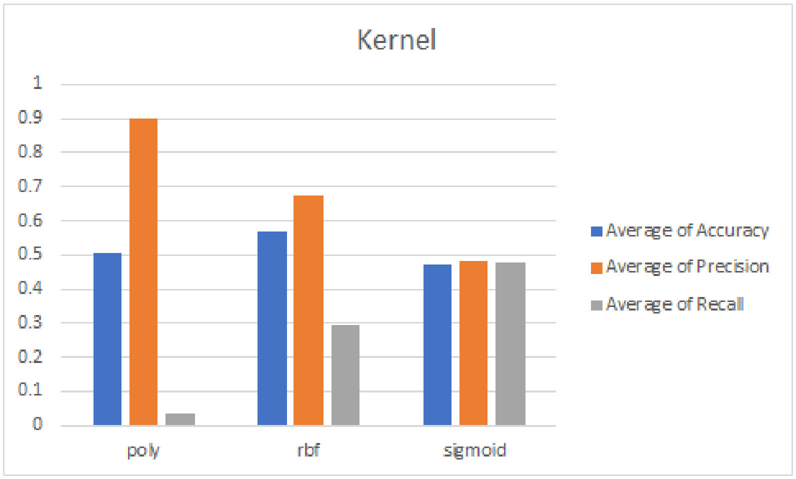

Figure 11: SVC Performance at Variable Kernels

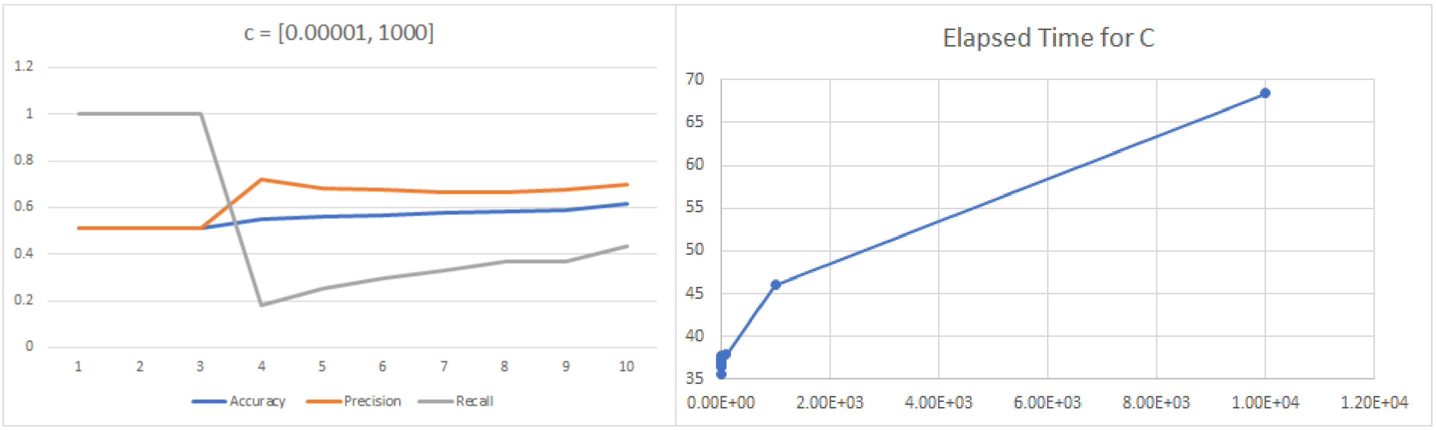

Figure 12: SVC C Values and Elapsed Time

The ‘rbf’ kernel was used for its increased accuracy and precision values. A c value of 100 was chosen due to fair metric values, as well as the comparatively average run time compared to other higher c values for this model.

KNeighborsClassifier

The primary parameter to tune for the KNN classifier was the number of neighbors. As the number of neighbors increases, a slight downward trend can be observed. A value of n=2 is chosen for the number of neighbors, as at n=1 each sample is using itself as a reference and could be prone to overfitting.

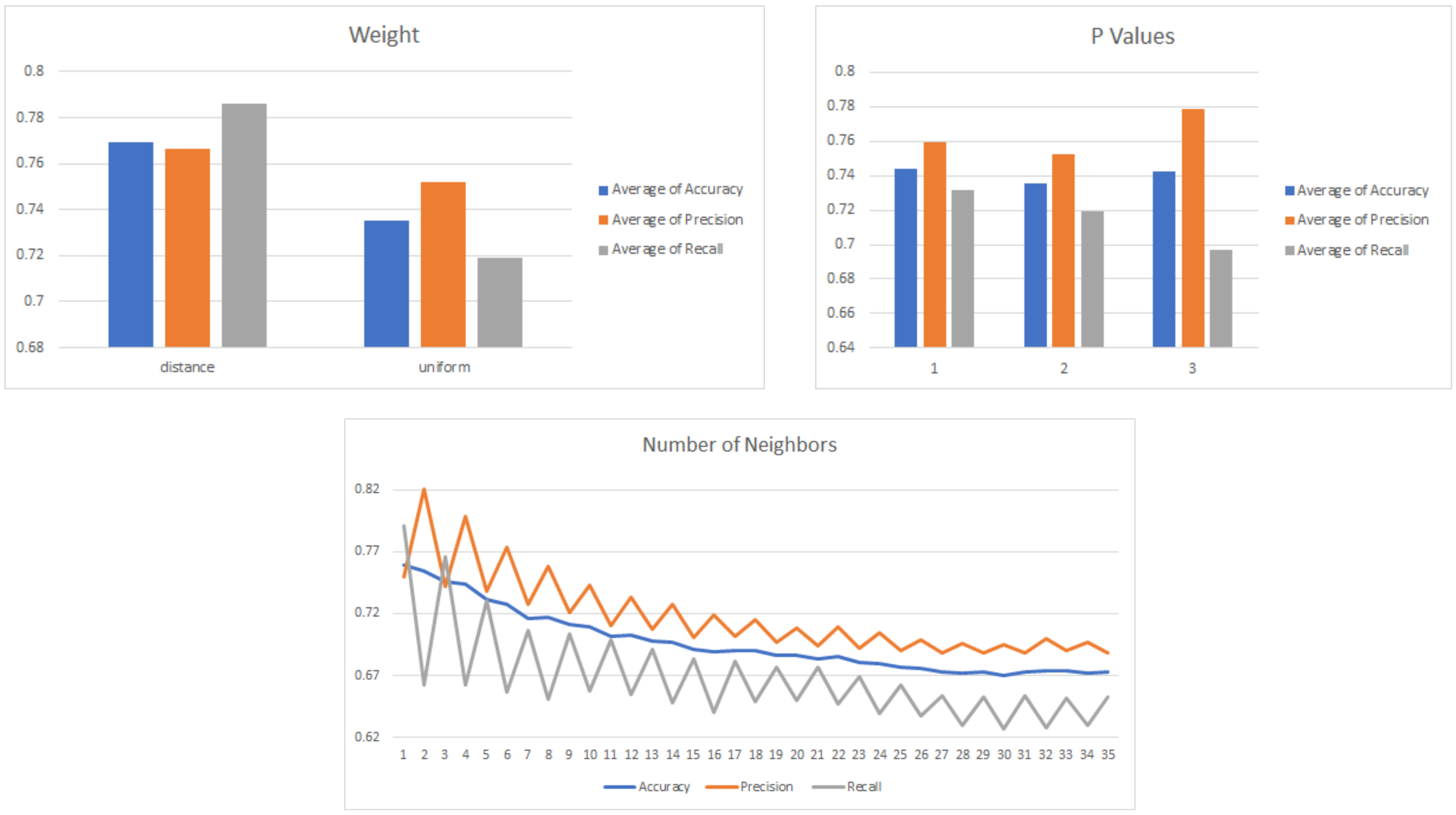

Figure 13: KNeighbors Parameter Tuning

Experiments were also performed for the weight and P value parameters. The final tuned KNN classifier will consist of n=2 for the number of neighbors, as well as weight=distance, and p=1.

StackingClassifier

The tuning process for the final stacked classifier is shown below in figure x. The process began with the creation of an untuned StackingClassifier from the rest of the untuned classifiers chosen. The StackingClassifier had the highest accuracy (0.876), precision (0.895), and recall (0.858) of all classifiers.

After the initial tuning, the performance of the StackingClassifier model fell. Since the RandomForest seemed to be performing worse, even after some supposed tuning, parameters were removed and left up to the algorithm. As the worst performing model, the SVC model was also abandoned. After these final adjustments, with the average of 5 consecutive runs, the performance of the StackedClassifier increased from initial creation to final tuning. The accuracy rose from 0.877 to 0.882, the precision from 0.895 to 0.901, and the recall from 0.858 to 0.862.

This final classifier was stashed away and saved using the Python joblib package, so that it could be used to perform evaluations and predictions of more malicious tweets.

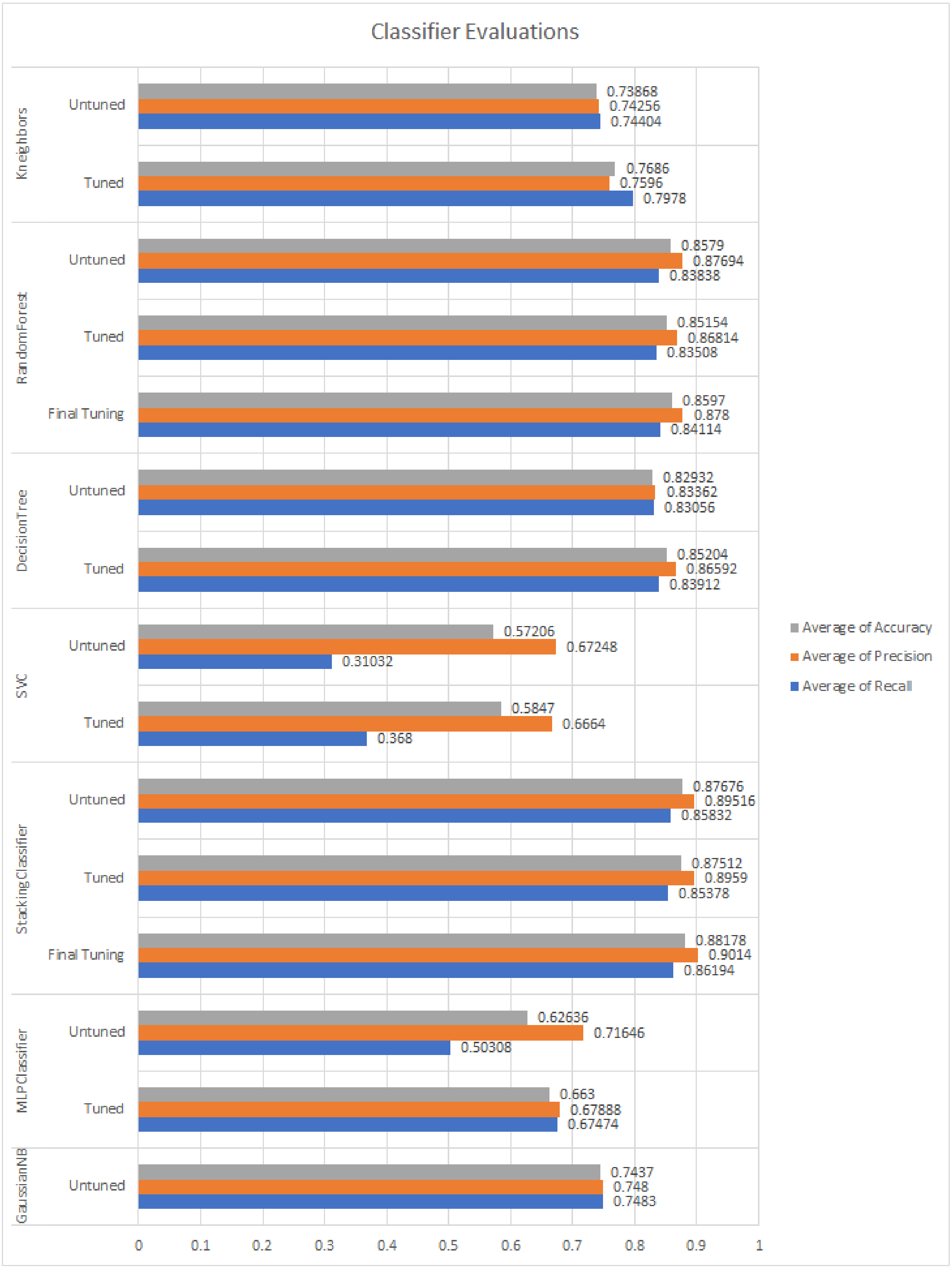

Figure 14: Tuning Process Iterations

3.7 Validation Experiments

A few final validation experiments were conducted to assess the performance of the final model. The makeup of the validation datasets is closer to the ratios that were naturally seen during collection, and does not resemble the balanced data sets that the models were trained with. It is expected that the accuracy of the StackingClassifier will decrease somewhat since it is experiencing samples that have yet to be seen. Additionally, the precision of the model will likely be low, it might classify many benign samples as 'malicious' since it has only experienced a balanced dataset before. Recall is expected to be lower than the training/testing evaluations.

The first validation experiment was conducted with datasets relating to COVID-19, like those that were initially collected. The model correctly predicted 62 out of 99 total malicious instances, with a recall of 63%. The model incorrectly flagged a number of benign samples as malicious, with a total of 95 false positives. This is to be expected, since the model had been trained on a balanced dataset.

The second validation experiment was conducted with data relating the to vaccination, using the tag keyword '#vaccine'. In terms of recall, the model actually performed better with the vaccine dataset than the additional COVID-19 data in the first validation experiment. The model was able to correctly identify 95 out of 137 malicious instances for a recall of 69%. The model's precision was also higher with 45% precision compared to 39% precision for the additional COVID-19 dataset. The model flagged 116 false positives.

The third validation test was conducted using a dataset completely unrelated to the COVID-19 data, using the tag '#ElectionResults2021', which had been trending on Twitter briefly on May 7th, 2021. The trained model struggled the most with this election dataset, seeing precision drop below 2% and recall scores drop significantly as well, to around 17%.

In conclusion, the model generalizes well to topics similar to the COVID data it has been trained on, but beyond similar data, it begins to have trouble correctly predicting the nature of samples. There is not enough data to tell, but poor performance on unrelated datasets might indicate some unique malicious characteristics established in the COVID dataset.

Validation Experiment 1: Additional COVID-19 Data

n=1075	Predicted: Benign	Predicted: Malicious
Actual: Benign	TN = 881	FP = 95
Actual: Malicious	FN = 37	TP = 62

accuracy	0.877209
precision	0.394904
recall	0.626263

Validation Experiment 2: Vaccine Dataset

n=1226	Predicted: Benign	Predicted: Malicious
Actual: Benign	TN = 973	FP = 116
Actual: Malicious	FN = 42	TP = 95

accuracy	0.871126
precision	0.450237
recall	0.693431

Validation Experiment 3: Election Dataset

n=814	Predicted: Benign	Predicted: Malicious
Actual: Benign	TN = 696	FP = 106
Actual: Malicious	FN = 10	TP = 2

accuracy	0.857494
precision	0.018519
recall	0.166667

3.8 Unit Testing

3.8.1 Testing for VirusTotal

After feature selection was determined, some tests of feature extraction and VirusTotal functionality were conducted to ensure proper functionality of the processing scripts. VirusTotal testing was accomplished using known malicious URLs which were taken from a trusted source on the subject; Norton Safeweb [29] revealed 17ebook[.]com to be a malicious site. Sending this URL to VirusTotal V2 API endpoint "url/report" returned 3 positive flags from 87 total antivirus scanners, including flags from the Sophos, Quttera, and Dr.Web malware scanners.

3.8.2 Testing for Text Processing and NLP

A series of unit tests were performed on a sample piece of text to determine if the processing script performs as it should. After comparing the expected results to the actual results, the script performs as it is supposed to.

Sample Text Input

This is some sample text simulating a Twitter Tweet. Will the processing script work like it should?? LET'S FIND OUT!!! SO EXCITING! 😁😛😋 @SampleUser #SampleTweet

Labels for Text Classification (Using SpaCy NLP)

TEXT	LABEL
Twitter Tweet	ORG
Summary: PERSON: 0 GPE: 0 NORP: 0 ORG: 1 EVENT: 0	

Part of Speech Tagging (Using SpaCy NLP)

TEXT	POS (Part of Speech)

This	DET
is	AUX
some	DET
sample	NOUN
text	NOUN
simulating	VERB
a	DET
Twitter	PROPN
Tweet	PROPN
Will	VERB
the	DET
processing	NOUN
script	NOUN
work	NOUN
like	SCONJ
it	PRON
should	VERB
LET	VERB
‘S	NOUN
FIND	VERB
OUT	ADP
SO	ADV
EXCITING	ADJ

Summary:	
ADJ: 1	AUX: 1
ADV: 1	CONJ: 0
INTJ: 0	DET: 4
NOUN: 6	NUM: 0
PROPN: 2	PART: 0
VERB: 5	PRON: 1
ADP: 1	SCONJ: 1

Emotion/Sentiment Analysis (Using the NRC Emotion Lexicon A.K.A. EmoLex)

TEXT	EMOTION/SENTIMENT
text	{anger: 0, anticipation: 0, disgust: 0, fear: 0, joy: 0, negative: 0, positive: 0, sadness: 0, surprise: 0, trust: 0}
simulate	{anger: 0, anticipation: 0, disgust: 0, fear: 0, joy: 0, negative: 0, positive: 0, sadness: 0, surprise: 0, trust: 0}
script	{anger: 0, anticipation: 0, disgust: 0, fear: 0, joy: 0, negative: 0, positive: 1, sadness: 0, surprise: 0, trust: 0}
work	{anger: 0, anticipation: 0, disgust: 0, fear: 0, joy: 0, negative: 0, positive: 0, sadness: 0, surprise: 0, trust: 0}
find	{anger: 0, anticipation: 0, disgust: 0, fear: 0, joy: 0, negative: 0, positive: 0, sadness: 0, surprise: 0, trust: 0}
exciting	{anger: 0, anticipation: 1, disgust: 0, fear: 0, joy: 1, negative: 0, positive: 1, sadness: 0, surprise: 1, trust: 0}
Summary:	
anger: 0	negative: 0
anticipation: 1	positive: 2
disgust: 0	sadness: 0
fear: 0	surprise: 1
joy: 1	trust: 0

Text Attribute Analysis

ATTRIBUTE	EXPECTED OUTPUT	ACTUAL OUTPUT	MATCH
ATTRIB_avg_sentence_length	27.4	27.4	✓
ATTRIB_avg_word_length	5.48	5.48	✓
ATTRIB_length_of_tweet	161	161	✓
ATTRIB_number_of_capital_letters	29	29	✓
ATTRIB_number_of_emojis	3	3	✓
ATTRIB_number_of_exclamation_marks	4	4	✓
ATTRIB_number_of_hashtags	1	1	✓
ATTRIB_number_of_periods	1	1	✓
ATTRIB_number_of_question_marks	2	2	✓
ATTRIB_number_of_sentences	5	5	✓
ATTRIB_number_of_special_characters	10	10	✓
ATTRIB_number_of_uppercase_words	5	5	✓
ATTRIB_number_of_user_mentions	1	1	✓
ATTRIB_number_of_words	25	25	✓

* Note: Python doesn't count emojis when using the len(text) function

3.9 Misinformation Classification

In addition to malicious/benign tweet classifications, there was also some interest in performing similar analysis on tweets which might contain misinformation. The plan was to use the fact checking tools being developed by FullFact [30] to fact check any claims in the tweet text data, and perform supervised machine learning to discover any features which could be indicators of false claims. FullFact is currently being tested and made available to some individuals and organizations through a sign up process. Unfortunately, after signing up for access at the end of March, no response was received, and this section of the research could not be fully realized. Several days were spent fact checking tweets by hand, with a goal of finding at least 5 disputed claims to build a model on.

A set of rules/criteria were established to give the fact checking process some sort of order. Several criteria will be evaluated for each piece of text:

1. Contains claims: whether the text contains an actual claim to be fact checked or if it contains questions, only hashtags/links, generic info that isn't a claim, etc.
    a. Contains a full sentence that is a statement, not a question.
    b. Validity of the claim could be established factually and the statement is not an opinion of the writer (ex. Top 10 x Things, Pros and Cons of x)
    c. Contains a simple statement that can be proved from 1 or a few pieces of evidence, and is not too broad or overreaching, or lacking in research.
2. Claim validity: whether the claim included in the text is valid.
    a. A factually sound document, piece of evidence, or any type of source exists that can prove the validity of the claim.

If there is enough data collected through manual fact checking, model creation could be done in several different ways and the accuracy of these methods measured against each other.

1. Binary classification where the 2 classes are: 'claim valid', 'claim disputed'
    a. Additionally, for this instance, the tweets that contain no claims could either be included as part of the claim valid class, or cut completely from the input data. Again, both methods could be used and measured against each other.
2. Multiclass classification problem with 3 classes: no claims, claim valid, claim disputed

These models could be tested against each to determine where might be more suitable, and produce a better evaluation. Since fact checking will be done by hand, some criteria are established for these fields, to ensure a similar process is applied to each sample.

In total, 114 tweets were checked for validity. Of these, 53 contained claims that could be investigated. And out of these 53 claims, only 1 was discovered that contained refuted information. The data was not completely chosen at random, only samples in the ‘benign’ category were chosen, since fact checking was completed on the author’s PC, they wished to avoid downloading any viruses and checked only benign samples. Additionally, to speed up the process, only text data in english was fact checked, therefore some bias could be introduced. Feeding this data into all classifiers, the following features were determined by the random forest to have the highest importance, the following values are averages over the course of 10 runs on the same data.

Feature Importance

TWEET_days_since_account_creation	0.198788
ATTRIB_number_of_emojis	0.109254
ATTRIB_number_of_capital_letters	0.072932
ATTRIB_avg_word_length	0.063205
POS_SCONJ	0.053988
TWEET_listed_count	0.050346
TWEET_favourites_count	0.047004
ATTRIB_number_of_words	0.046266
POS_ADJ	0.03472
POS_PRON	0.032914
SENTIMENT_negative	0.029871
POS_AUX	0.022617
ATTRIB_avg_sentence_length	0.018924

All models, as expected, had 100% accuracy, precision, and recall. The stacked classifier could not be constructed, due to needing more instances of the disputed claim class. But with only 1 instance of a disputed claim in the training data, all models are prone to extreme overfitting and are essentially useless.

It may be interesting to look at what features differed between the one tweet with a disputed claim and all others, but there is not nearly enough data to even hint that these features might be helpful in finding tweets that spread disinformation. In conclusion, as always, more data is needed.

4 Results and Evaluation

4.1 Data Collection Trends

Several days into the data collection phase of the project, more detailed collection logging was put into place, so that potential trends could be observed from these details. It was discovered that some hashtags contained a higher percentage of malicious URLs that others. The hashtag #ccpvirus topped with 33% malicious makeup, followed by #coronavirus at 14% and #covid and 13%, with the rest hovering around 9% or lower. The tag 'ccpvirus' stands for Chinese Communist Party Virus, and references the unfounded claim that COVID-19 is a Chinese bioweapon. It's important to note that although "ccpvirus" was one of the tags with the highest malicious percentage, it contained a very small number of samples, with 1 malicious tweet out of a total of 3 samples; the data is not completely conclusive. The results of tag collection and analysis are illustrated in more detail in Figure 15.

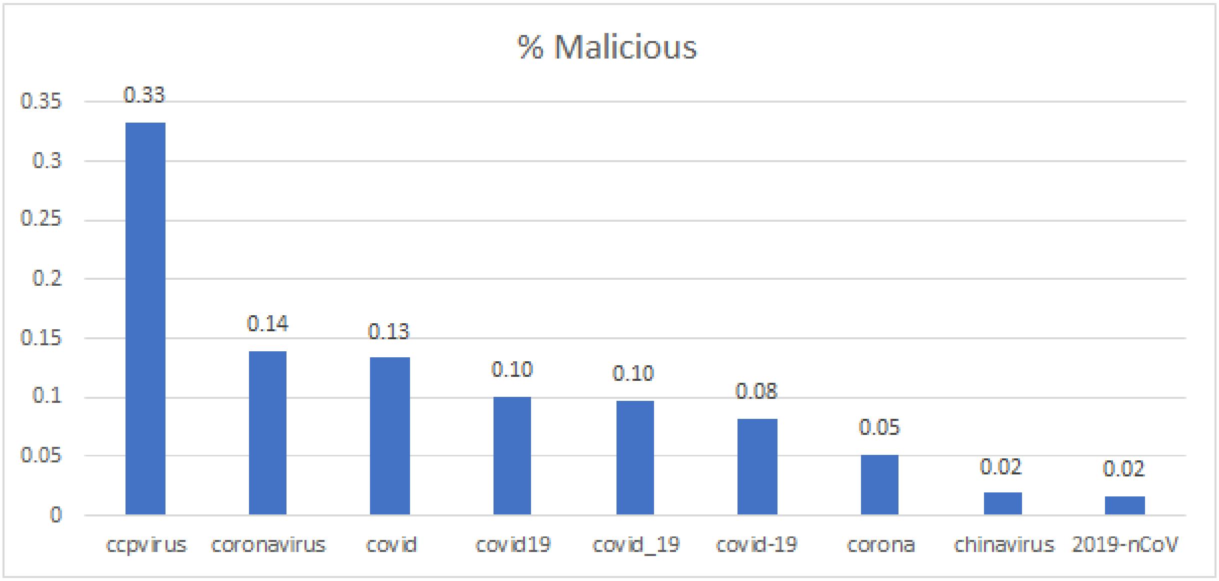

Figure 15: Malicious Percentage of Hashtags

The data collection script was run on 16 different days throughout the month of march, between the hours of 15:00 - 05:00 UK time. According to the CSSE Covid Dashboard from John Hopkins University [31], the global number of COVID-19 climbed throughout March 2021; starting at 305.864k cases on March 1st and finishing at 683.205k on March 31st. A slightly different trend is observed in the number of deaths, with 7.207k deaths on March 1st, a slight dip mid-month, and a rise in deaths once more at the end of the month, reaching 12.274k deaths on March 31st.

Cases doubled throughout the month with a 70% rise in daily deaths; this type of information does not go unnoticed by the public. And negative news is often taken advantage of by malicious actors throughout the world. This data was also recorded throughout the data collection process to determine if coronavirus news would be related to the percent of malicious tweets collected, assuming malicious actors could take advantage of public sentiment. Luckily, the percentage of malicious tweets to benign ones seemed to hold steady throughout the month of march, despite the fact that covid cases and deaths were rising. Despite the author's initial hypothesis, the percentage of malicious tweets collected daily did not seem to be related to the rise in COVID-19 cases and deaths observed during the month of March 2021. This could also be due to differing daily collection times, and the lack of collection on some days during the month. The proportion of malicious tweets collected daily is shown in Figure 16.

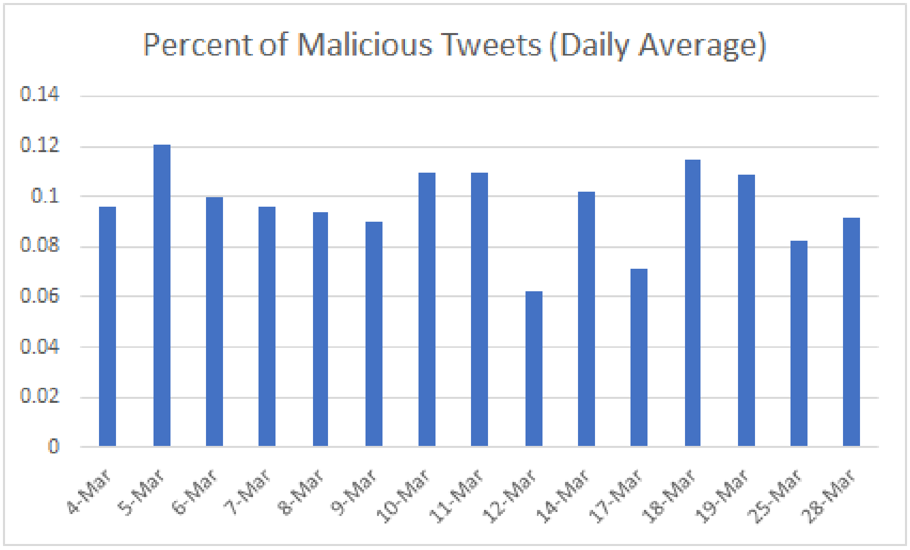

Figure 16: Malicious Tweet Percentage of Total (Daily Average)

Finally, some language trends were observed during data analysis, namely which languages had the highest amount of malicious samples percentage-wise. Topping the list was zh-tw (Chinese - Taiwan) with 29%, followed by fa (Farsi) at 21% and he (Hebrew) at 20% malicious makeup. These 3 languages also had a very small tweet sample size; more data might be required for a better accuracy rate.

ISO Language Code	Total Instances	Malicious Instances
he	4	1
fa	22	6
zh-tw	10	4

The language with the lowest malicious percentages were Swedish and Malayalam, at 1% each.

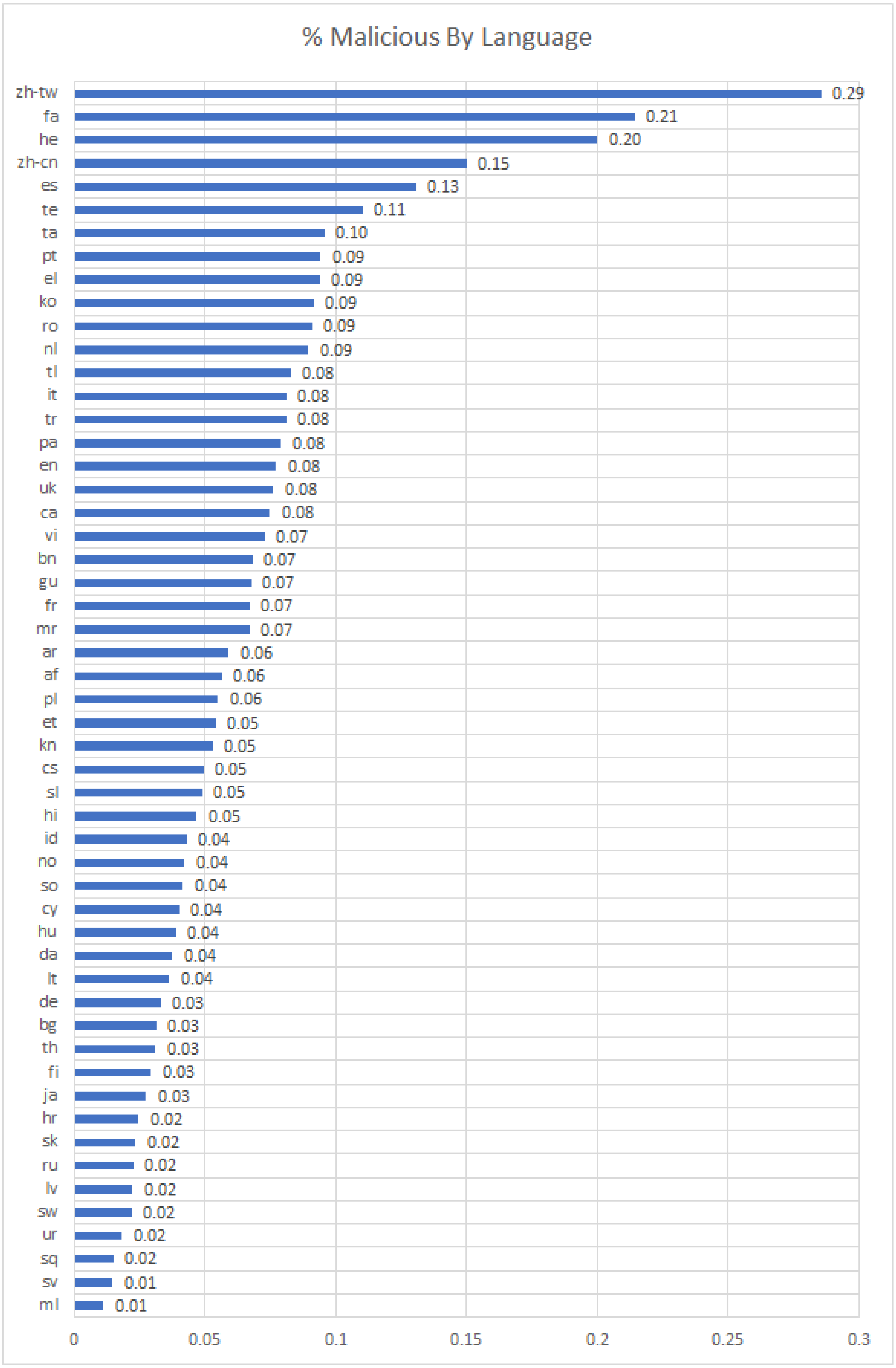

Figure 17: Percentage Malicious By Language Data

4.2 Feature Analysis

Using the feature_importance variable available in the RandomForestClassifier, it is possible to see some of the most important features used by the model in making a classification. Feature importance is calculated by using the mean and standard deviation of accumulation of the impurity, or the increase in node impurity at each split weighed by the probability of reaching that node.

According to the data at hand, the most important feature used in distinguishing between a benign or a malicious tweet was the URL length. The top six most important features consisted of tweet and account information, such as number of followers, favourites, and statuses. Following these, the next four important features consisted of tweet textual attributes, such as average word length, length of tweet, average sentence length, and number of special characters. The top 15 most important features included in the table below represent the average feature values extracted over 10 training runs with the mode. The top 15 features did not include any of the part-of-speech, label, or emotion and sentiment features which had initially been included in feature selection and analysis. The first POS feature to appear is the proper noun part-of-speech, with an importance value of 0.015954. For the POS features, the conclusion holds with Go et al 2009 [8], where POS features were found to not be very helpful in Twitter sentiment analysis. The first feature from the emotion & sentiment set, positive sentiment, appeared at position 25 and had an importance value of 0.008142, while the first feature from the label set, the organization label, appeared at position 28, having an importance value of 0.007048.

Feature Importance for Benign/Malicious Classification (Top 15)

TWEET_url_length	0.316275
TWEET_followers_count	0.052508
TWEET_favourites_count	0.045262
TWEET_statuses_count	0.044185
TWEET_listed_count	0.040215
TWEET_days_since_account_creation	0.03799
TWEET_friends_count	0.037069
ATTRIB_avg_word_length	0.029988
ATTRIB_length_of_tweet	0.029703

ATTRIB_avg_sentence_length	0.027849
ATTRIB_number_of_special_characters	0.024759
TWEET_days_since_tweet_creation	0.024035
ATTRIB_number_of_capital_letters	0.023291
ATTRIB_number_of_words	0.020093
ATTRIB_number_of_hashtags	0.016736

Some analysis was also performed on the final processed dataset of approximately 26,000 samples (half of the samples being malicious, and half being benign). The benign and malicious samples were split into separate datasets, and statistical analysis was performed on these separate datasets, including the discovery of minimum, maximum, mean, and standard deviation values.

The minimum values of all features in both datasets were generally identical or very close. Maximum values were also generally close across datasets, with the highest differences in maximum values being seen in the followers, friends, and listed count features. The malicious dataset had a friend count maximum that was 245,650 higher than the benign maximum, along with a listed count that was 3390 higher. The benign dataset had a follower count maximum that was 8,191,479 higher than the malicious dataset.

In terms of the mean values, only one feature in the POS, label, sentiment, and emotion features had a difference in mean values between the datasets that was greater than 0.20, which was the number of proper nouns at 0.2173, where the benign dataset had a greater number of proper nouns in each sample (on average) than the malicious dataset. The greatest difference between feature mean values were observed in the textual and tweet attribute features. The malicious dataset had significantly higher mean values for number of followers (166,581 higher than benign), statuses (53,985 higher than benign), friends (2,221 higher than benign), listed count (335 higher than benign), and interestingly enough, days since account creation (295 higher than benign). While the benign dataset had significantly higher values of favourites count (6522 higher than malicious set), and URL length (43 higher than malicious set).

Standard deviation was also analyzed between datasets. The malicious dataset had a significantly higher variation from the mean at the number of friends, followers, statuses, and number of special characters, whilst the benign dataset had a higher variation at number of favourites, listed count, days since account creation, and URL length.

Unfortunely, not enough data could be collected to gain further insights about which features might be relevant for classifying tweets containing misinformation. The discovery of only 1 sample with a disputed claim in the dataset would mean extreme overfitting and an essentially useless model.

4.3 Model Evaluation

The worst performing models were the SVC and the MLPClassifier. The especially poor performance of the SVC model could be an indicator that the problem space is non-linear, supported by the success of the DecisionTree and RandomForest models, which are both non-linear models. The performance metrics for the MLPClassifier also indicated poor performance. This could be caused by lack of sufficient samples required to train a neural network successfully, as discussed in the model selection section. The best performing models in regards to malicious Tweet classification in the context of COVID-19 data were the Random Forest and the Stacking Classifier.

5 Final Deliverable

5.1 Overview

After data collection, analysis, and classification training, a final project deliverable was created building on this data. This script runs classifications on tweet samples provided in CSV form. The script processes the following tweet and account data into a list a features as input to the classification model: text, created_at, retweet_count, favorite_count, verified, followers_count, friends_count, listed_count, favourites_count, statuses_count, user_created_at, url

These data points are processed into the list of labels, POS, and text attributes, and then classified by the trained model as either '0' for 'benign' or '1' for 'malicious'. The script then outputs a file resembling the initial input file, with an added column of malicious/benign predictions.

5.1 Usage and Examples

ModelPrediction.py is a tweet classification script that will classify samples as either malicious (1) or benign (0). The model used in this script is trained on approximately 26,000 COVID19 related tweets collected during the month of March 2021.

This script has one required argument which is the inputfile, and one optional argument of the google service account json (used for translation purposes). This following example line will call the ModelPrediction script to classify all samples in the sample.csv file WITHOUT translating any non-english tweets into english first.

```
>	py ModelPrediction.py sample.csv
```

This following example line will call the ModelPrediction script to classify all samples in the sample.csv file after translating non-english tweets into english.

```
>	py ModelPrediction.py sample.csv COVIDURLS.json
```

The inputfile is expected to contain a header with the following data points, order is irrelevant. Additionally, the inputfile can contain as many other fields as you like as long as the required fields are included, the additional fields will be simply ignored.

text, created_at, retweet_count, favorite_count, verified, followers_count, friends_count, listed_count, favourites_count, statuses_count, user_created_at, url

The script will create 2 files. The first is the output file, which will be titled [inputfile]_OUTPUT.csv, and will contain processed features of the tweet information. This is the data that will be used by the classifier to determine maliciousness. The prediction file, titled [inputfile]_PREDICTION.csv will contain the original data included in the inputfile, along with an additional column which will have classifier predictions: 0 for benign and 1 for malicious In order for this script to function correctly, the files 'emolex_wordlevel_92.txt' (used for sentiment analysis) and 'sclf_trained_model.pkl' (trained classifier model) must be left intact in the same directory as the script.

6 Future Work

The results of this process and research were used to create a trained model capable of identifying malicious COVID-19 tweets based on tweet and account input data. In future work and development of this sort of filter, a fuzzy classifier could be implemented instead of a binary one. This could be used to identify tweets which are on some arbitrary threshold of being malicious, and only investigate these specified tweets.

Work could be continued on the improvement of this classifier with the addition of more features as discussed in section 2.2. These expanded features could include the use of emoticon libraries which map sentiment to emoticons, as well as the translation of acronyms to words to aid in sentiment analysis. The addition of popular word unigrams could also prove helpful to classification. Additionally, the use of more URL features, as discussed in section 2.1, could have aided in classification.

Additionally, work could be continued on the misinformation classification section that was started and halted in this project. With further development of automatic fact-checking tools, or crowdsourced manual fact checking, a large dataset of tweets containing disputed claims could be discovered. This dataset could be used to discover textual and account features that might be more likely to spread disinformation.

7 Reflection

The entire process of data collection, processing, and model building was something that I had previously done on a much smaller scale than this project. I delved deeper into model tuning and comparison between models in this project.

As my research progressed, there were some things I discovered that could improve the quality of my model or the dataset, but I didn't have enough time to implement them, or it would mean re-doing a lot of my early steps. One of the biggest things I discovered was the use of shuffling in the feature selection stage. This would include the removal of features, followed by a re-run of the model to determine if it performed worse, better, or the same. This is a way of determining feature importance that takes some time to perform, depending on the libraries used.

Additionally, throughout the model training and tuning process, I didn't pick a clear metric to rely on when selecting certain model parameters, simply saying 'this parameter gives higher accuracy or precision'. When I was near to finishing the project, I thought that recall should have been my metric of choice if I wanted my models to identify as many malicious instances as possible, even though I generally picked parameters that resulted in higher accuracy and precision over recall. If I had a chance to start over, I would have paid more attention to what kind of metric I wanted my model to focus on rather than just arbitrarily picking a few.

In terms of time constraints, there were some features that I didn't implement mostly because of time constraints, a lot of which would have been very interesting to look at. Several papers mentioned in section 2.2 addressed the success of unigrams for sentiment analysis, but since these unigrams features might take a lot of time and research to implement, as well as the fact that they were discovered later into feature selection, they were excluded from the project. Additionally, some other interesting features were discovered later into research which were also left out, including: emoticon processing, finding/counting mispellings, and finding/counting acronyms. Had I allotted more time for the feature selection phase, I could have discovered features which created a better model.

One more thing which was outside the scope of the project but could have created a better classifier had to do with tuning the responses received from the VirusTotal API. The reliability of each of the 70+ scanners used by VirusTotal could have been researched, with the final version including different weights for each of the scanners depending on their reliability, since false positives are completely possible with VirusTotal. In this case, the dataset size might have been significantly decreased and caused the quality of some of the models to degrade, such as the MLPClassifier, which performs better with more training data. The decreased size of the dataset might have revealed features which were truly indicated of malicious behaviour, but this is a tradeoff that always must be considered.

References

[1] Chen, J. 2020. COVID-19: Cloud Threat Landscape. paloaltonetworks, unit 42. https://unit42.paloaltonetworks.com/covid-19-cloud-threat-landscape/

[2] Tombs, N., & Fournier-Tombs, E. 2020. Ambiguity in authenticity of top-level Coronavirus-related domains. *Harvard Kennedy School (HKS) Misinformation Review*. https://doi.org/10.37016/mr-2020-036

[3] Developing Story: COVID-19 Used in Malicious Campaigns. Security News, www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/coronavirus-used-in-spam-malware-file-names-and-malicious-domains.

[4] Ispahany, J. and Islam, R., 2020. Detecting Malicious URLs of COVID-19 Pandemic using ML technologies. *arXiv preprint arXiv:2009.09224*

[5] S. Lee and J. Kim, "WarningBird: A Near Real-Time Detection System for Suspicious URLs in Twitter Stream," in IEEE Transactions on Dependable and Secure Computing, vol. 10, no. 3, pp. 183-195, May-June 2013, doi: 10.1109/TDSC.2013.3

[6] Luciano Barbosa and Junlan Feng. 2010. Robust sentiment detection on Twitter from biased and noisy data. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters* (*COLING '10*). Association for Computational Linguistics, USA, 36–44.

[7] Gamon, Michael. 2004. Sentiment classification on customer feedback data: Noisy data, large feature vectors, and the role of linguistic analysis. Proceedings of the 20th International Conference on Computational Linguistics. 10.3115/1220355.1220476.

[8] Go, Alec & Bhayani, Richa & Huang, Lei. 2009. Twitter sentiment classification using distant supervision. Processing. 150.

[9] Apoorv Agarwal, Boyi Xie, Ilia Vovsha, Owen Rambow, and Rebecca Passonneau. 2011. Sentiment analysis of Twitter data. In *Proceedings of the Workshop on Languages in Social Media* (*LSM '11*). Association for Computational Linguistics, USA, 30–38.

[10] World Health Organization. 2015. World Health Organization Best Practices for the Naming of New Human Infectious Diseases. WHO/HSE/FOS/15.1

[11] The Honeynet Project. 2012. Cuckoo 0.3.1 Released. https://www.honeynet.org/2012/01/03/cuckoo-0-3-1-released/

[12] VirusTotal - Documentation. https://support.virustotal.com/hc/en-us/categories/360000162878-Documentation

[13] Hashtagify. https://hashtagify.me/hashtag/covid

[14] Raytheon BBN Technologies. 2012. OntoNotes Release 5.0 with OntoNotes DB Tool v0.999 beta. https://catalog.ldc.upenn.edu/docs/LDC2013T19/OntoNotes-Release-5.0.pdf

[15] SpaCy V2.3.5, en_core_web_sm version 2.3.1. https://spacy.io/models/en#en_core_web_sm

[16] Mohammad, S., & Turney, P. 2013. Crowdsourcing a Word-Emotion Association Lexicon. *29(3), 436–465.* https://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm

[17] Maciej A. Mazurowski, Piotr A. Habas, Jacek M. Zurada, Joseph Y. Lo, Jay A. Baker, & Georgia D. Tourassi. 2008. Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance. *Neural Networks, 21(2), 427-436.*

[18] Wei Q, Dunbrack RL Jr. The role of balanced training and testing data sets for binary classifiers in bioinformatics. *PLoS One*. 2013;8(7):e67863. Published 2013 Jul 9. doi:10.1371/journal.pone.0067863

[19] Raschka, Sebastian. "StackingClassifier." *StackingClassifier - Mlxtend*, rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/.

[20] sklearn V0.0 https://scikit-learn.org/stable/

[21] Richard Maclin and David W. Opitz. 2011. Popular Ensemble Methods: An Empirical Study. *CoRR, abs/1106.0257.*

[22] Plate, T., Band, P., Bert, J., & Grace, J. 1997. A Comparison between Neural Networks and other Statistical Techniques for Modeling the Relationship between Tobacco and Alcohol and Cancer. In Advances in Neural Information Processing Systems. MIT Press.

[23] G. M. Foody & M. K. Arora. 1997. An evaluation of some factors affecting the accuracy of classification by an artificial neural network, International Journal of Remote Sensing, 18:4, 799-810, DOI: 10.1080/014311697218764

[24] Hush, "Classification with neural networks: a performance analysis," *IEEE 1989 International Conference on Systems Engineering*, 1989, pp. 277-280, doi: 10.1109/ICSYSE.1989.48672.

[25] Scikit-learn.org. 2021. *Classifier comparison — scikit-learn 0.24.2 documentation*. https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

[26] Scikit-learn.org. 2021. *Choosing the Right Estimator — scikit-learn 0.24.2 documentation*. https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

[27] Mithrakumar, M. 2019, November 12. *How to tune a Decision Tree?* Medium. https://towardsdatascience.com/how-to-tune-a-decision-tree-f03721801680.

[28] Scikit-learn.org. 2021. *SVC — scikit-learn 0.24.2 documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html?highlight=svc#sklearn.s vm.SVC

[29] Safeweb.norton.com. 2021. *Is This Website Safe | Website Security | Norton Safe Web - community buzz*. https://safeweb.norton.com/buzz

[30] Full Fact. 2021. *Automated Fact Checking - Full Fact*. https://fullfact.org/about/automated/

[31] Johns Hopkins Coronavirus Resource Center. 2021. *COVID-19 Map - Johns Hopkins Coronavirus Resource Center*. https://coronavirus.jhu.edu/map.html

[32] Scikit-learn.org. 2021. *Decision Trees — scikit-learn 0.24.2 documentation*. https://scikit-learn.org/stable/modules/tree.html

[33] Scikit-learn.org. 2021. *RandomForestClassifier — scikit-learn 0.24.2 documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html?highlight=random%20forest#sklearn.ensemble.RandomForestClassifier

[34] Scikit-learn.org. 2021. *MLPClassifier— scikit-learn 0.24.2 documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html?highlight=mlp%20classifier#sklearn.neural_network.MLPClassifier

[35] Saxena, R., 2017. *Gaussian Naive Bayes Classifier implementation in Python*. Dataaspirant. https://dataaspirant.com/gaussian-naive-bayes-classifier-implementation-python/

[36] Scikit-learn.org. 2021. *SVC. scikit-learn 0.24.2 documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html