Classifying Tweets
Trump VS. Musk VS. Random

Group: Chonkers
Group members: Linus Holmberg, Noa Holmén, Anton Nordkvist

Problem formulation

In this project we used text data in the form of Tweets from Elon Musk, Donald Trump, and Random users to predict who the author of a given tweet is. The application in which something like this could be used is any context when wanting to identify an author. For instance, when handling bots or similar on twitter.

Data

Data Collection

We downloaded the Trump and Musk data from Kaggle.com. The Random Tweets were downloaded from data.world.

The Trump data was collected using data scraping on publicly available tweets using, specifically GetOldTweets. Musk data was also collected by web scraping publicly-available tweets from Twitter.com. No information exactly which technology used for the scraping was provided on kaggle. Furthermore, according to the contributor of the random tweets the data was provided by the Data For Everyone Library on Crowdflower. No specific information about how the data was collected was provided.

Data Types

The data we are trying to categorize are tweets divided into three labeled categories (Trump, Musk, Random), The input x are text data, so the data type is nominal. The target y is the label of who wrote the tweet, which is nominal as well.

Data Cleaning

The data needed some initial cleaning before we could go any further on our analysis. We handled four things in the initial cleaning process, i) case sensitivity, ii) duplicates, iii) special characters (plus links and tags), and iv) empty tweets.

First of all, we made all characters lowercase to avoid the case sensitivity of python. We realize that writing in caps may bear potential information. For instance, it may be so that someone writes in all caps more often. However, we do not believe that this possible lost information will have a great implant in this context. It will also help us reduce the size of the vocabulary, otherwise the words “CHONKY”, “chonky” and “ChOnKy” would be three different words.

Second, just eyeballing through the CSV file we realized that it looked like there were some duplicates (probably as a result of the scraping process). We decided to remove all duplicates since some of them looked like pure spam, and that may influence our classifiers.

Third, we had to decide on how to handle cases that were not pure text. Such as, special characters (!?/”), links, and emojis. Hashtags (#) we saved as the word string it is. This since it felt wrong to simply make a word of it, or simply remove the whole. Keeping it in its original form gives useful information if it is a recurring hashtag. Furthermore, we decided to change all “words” starting with http:// or https:// to “assigned_a_link”. This way, we can still keep information about which class (user) is more likely to share a link. About special characters and emojis, we simply removed them. An alternative approach would be to keep them since that also may be beneficial information but we figured it would be easier to work with cleaner text. Moreover, we did a similar thing with tags. Words starting with “@” were changed to “person_tag”. Furthermore, after the initial cleaning we saw that one of Musk's most common words was “amp”. After some searching we understood that this was Musk using the “&”sign. Therefore we changed all the ‘amp’ to and to make it comparable to the other classes.

Four, we realized that after the data cleaning we had some tweets with the word length 0. The main reason for the 0 length tweets is someone just tweeting an emoji. Moreover, both Trump and Musk had some Tweets in languages other than English (Persian, Hindi and Russian). For some reason these tweets were counted as a length zero. Since we had a big enough dataset, we simply dropped all data points with a zero length.

Data Considerations

There were some things that we considered about our data/categories that we have to keep in mind going forward in our analysis.

Years

Our initial thought was to only include the years overlapping in the data sets. The reason for this was that some things were not possible to tweet about in 2009 that could be tweeted about in 2020 (or at least more likely, e.g. COVID or Tesla). However, unfortunately we could only get our hands on data spanning over different time periods for the three categories. The Musk-data are from 2010-2022, Trump-data 2009-2020, and the Random-tweets are from 2007-2015. Based on this, it would be nice to just include the years 2010-2015 (overlapping years).

After visualizing the tweets by all groups per year, we saw that the categories were not equally distributed between categories over the years (See plot below).

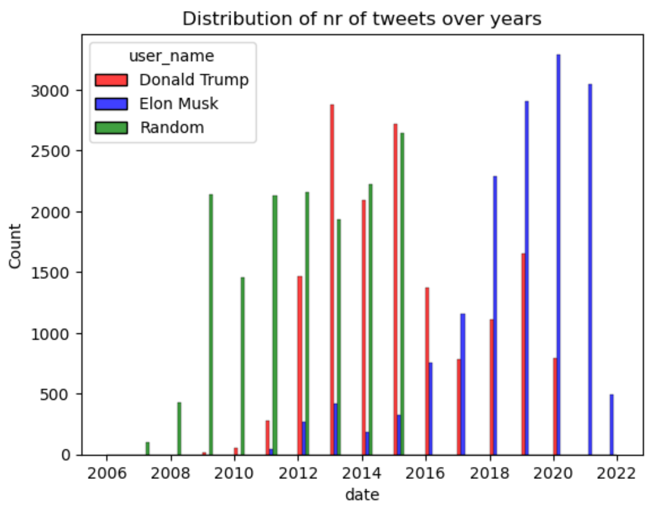


Fig. 1: Plot showing that Musk has more tweets after 2017 compared to Trump and Random.

As one can see above. Trump for instance was most active on twitter before Musk started being active for real. Therefore, we decided to just randomly select tweets from each category so that all categories contained an equal amount of tweets. Since we had the fewest tweets from Musk, we randomly selected tweets from the two other categories resulting in the same number of tweets. Which gave us 15129 Tweets per category, and a total dataset of 45657 Tweets.

Thus, we have to make the assumption the year a tweet was posted won't influence the result of our predictive models in a meaningful way. We tried to test this in a simple manner. By setting the "year" variable as target and trying to predict which year a tweet was from. The classifiers did not perform well, which strengthens our belief in the assumption.

Average Tweet Length

When balancing the datasets by the number of tweets, we indirectly chose to ignore the tweet length. For instance, it could be that Trump generally writes shorter tweets compared to the other two classes. By extension, this would have the consequence that our models will have more 'Musk' and 'Random' words compared to 'Trump words' to train on. However, we do not believe this to be a big issue, but we will explore this further.

Text Normalization

In language, words have various inflections. For instance, when writing in singular versus plural (i.e. 'duck' vs. 'ducks'). Therefore, one could argue that these two words should be counted as the same by the model. For the scope of this project, we didn't feel like it was reasonable to get into either 'Stemming' or 'Lemmalisation' of words. Moreover, we believe that it is even more crucial to handle this kind of issue if working with a model trying to capture the semantics/sentiment in the text.

Re-Tweets

After working a lot with the data, we realized that there seemed to be an unequal distribution of tweets between the categories containing re-tweets. It seemed like our dataset included more re-tweets from Trump compared to the other classes. This could be problematic since the text actually written is not produced by the labeled author. This could potentially have the consequence of looking similar to the random tweets, or including the name of the one re-tweeting. We realized this too late in the process to be able to do anything about it.

Splitting the Data

As soon as we were satisfied with the cleaning of the data we set aside 10%, (4564 randomly selected data points) to be test data. The remaining data (41067 data points) was used as training data. We did not use any of the test data during the training process.

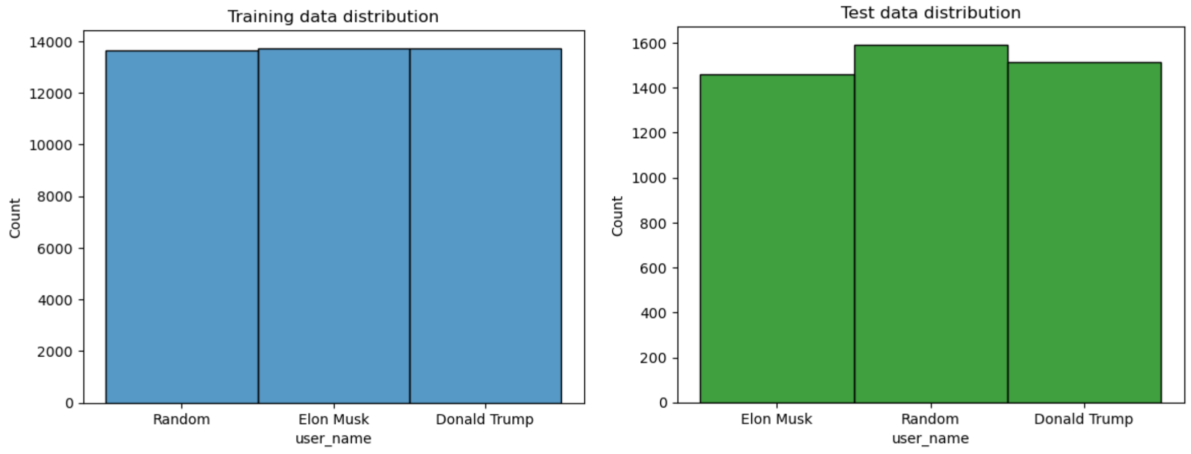


Fig. 2: Two bar plots showing that there is a similar class ratio (distribution) in the training data and the test data.

The distribution of tweets between classes were balanced in the training data and test data.

Descriptive analysis

Words per Tweet

Because of the consideration of words per tweet we mentioned earlier, we wanted to explore this further. Looking at the histograms below we cas suspect different distribution between the classes. Moreover, we suspect that Musk writes shorter tweets (M = 13.33, SD = 11.21) compared to Trump (M = 19.59, SD = 11.48). The random tweets seem to follow a nice gaussian distribution (M = 15.86, SD = 5.75).

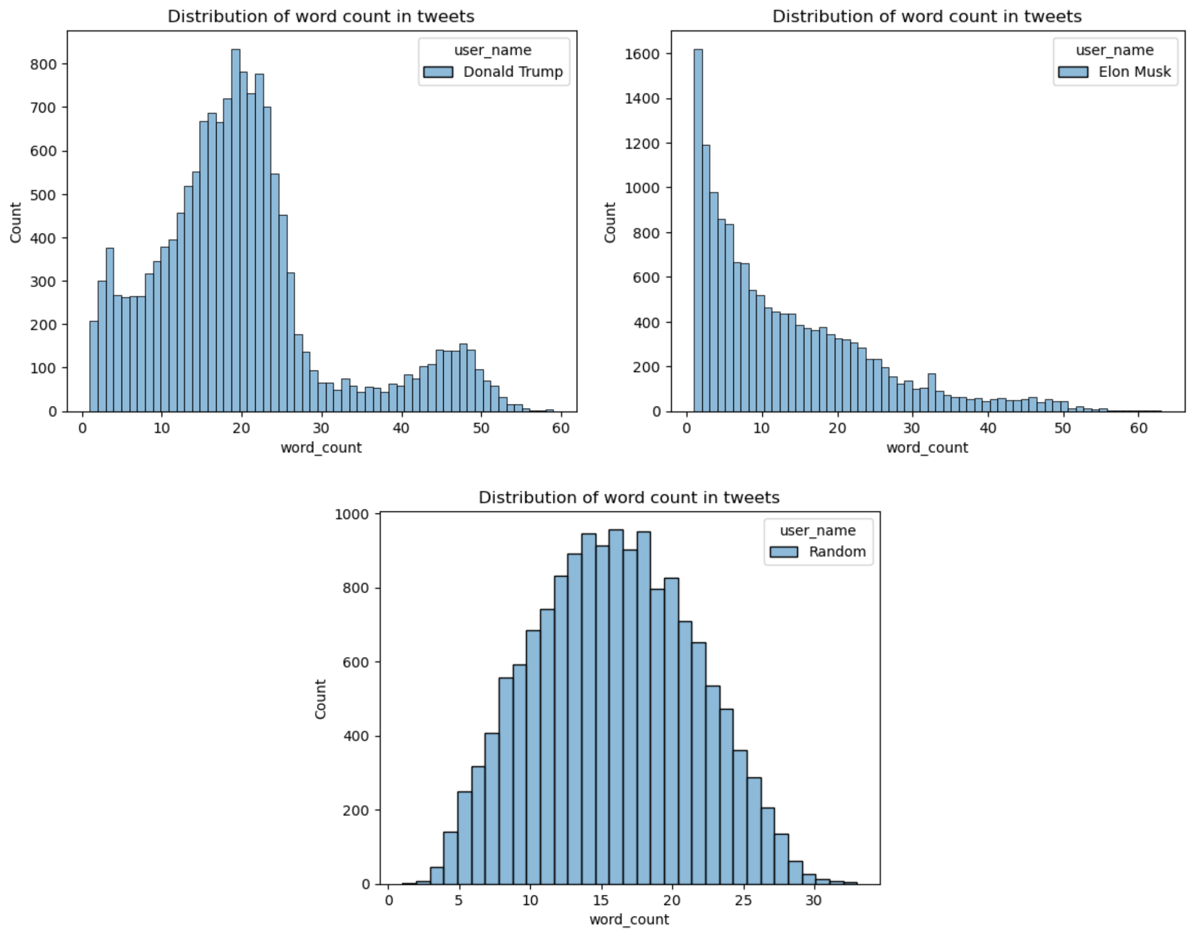


Fig. 3: The distributions of the number of words per tweet seem to be different between the classes. Random looks like a gaussian distribution, Trump a bimodal, and Musk a Chi-Square.

Word clouds

The word clouds contain the top 100 words except stopwords based on wordcloud's list of stopwords, “assigned_a_link” and “person_tag”.

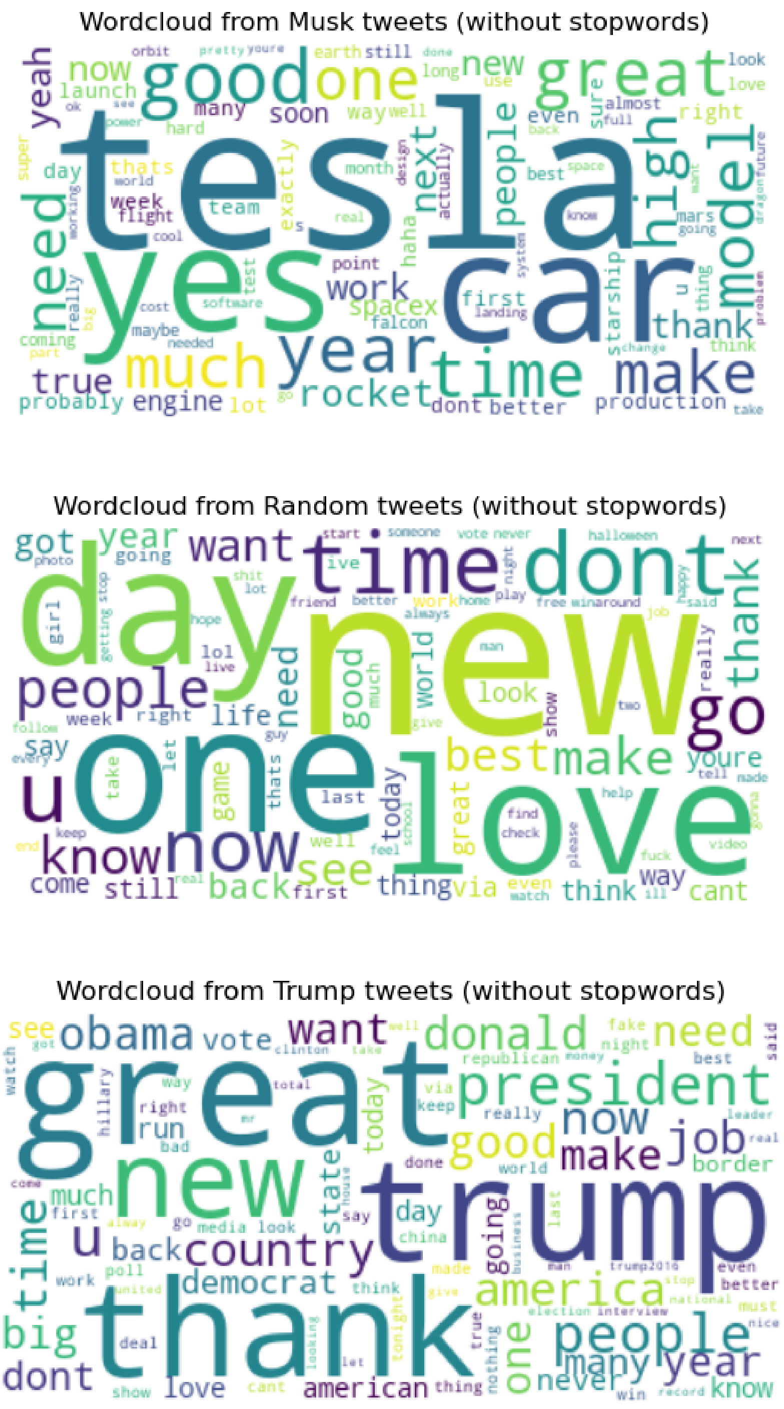


Fig. 4: The 100 most common words for each category, frequency is illustrated by size. For Musk, Tesla, car and yes are most frequent. For Random, new, one, and day. For Trump, Trump, great, and thank are most frequent.

Top 10 most frequent words

In the bar charts below we show the ten most frequently used words by each class and for all classes together. This is similar to the word clouds earlier, but give us a better precision of the top 10 words.

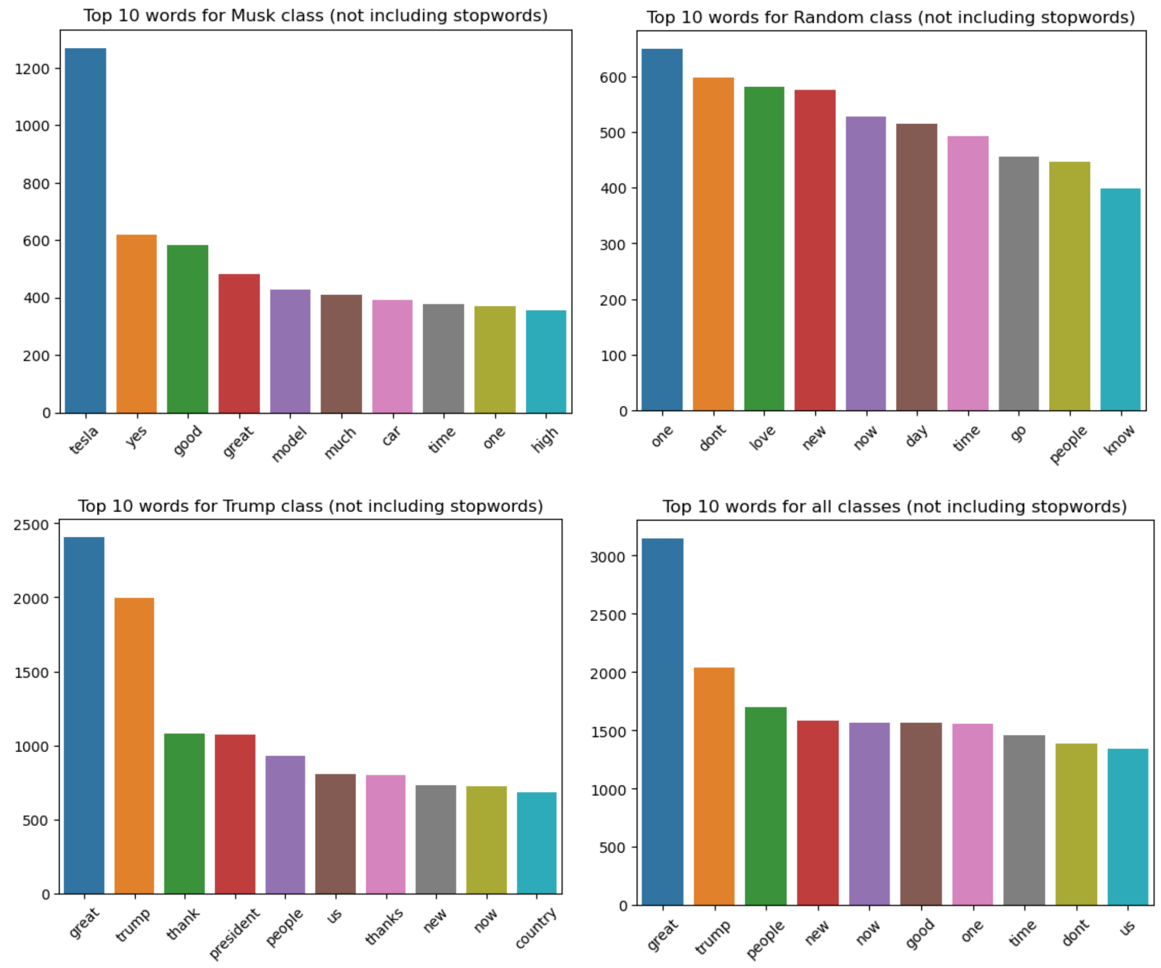


Fig. 5: The most common words overall are, "great", "trump", "people", "new", and "now".

Vocabulary length

In the bar plot below we explored the number of unique words per class (i.e. the class vocabulary). We believe that the reason that Musk seems to have a smaller vocabulary compared to the other classes may be because of what seem to be shorter tweets on average.

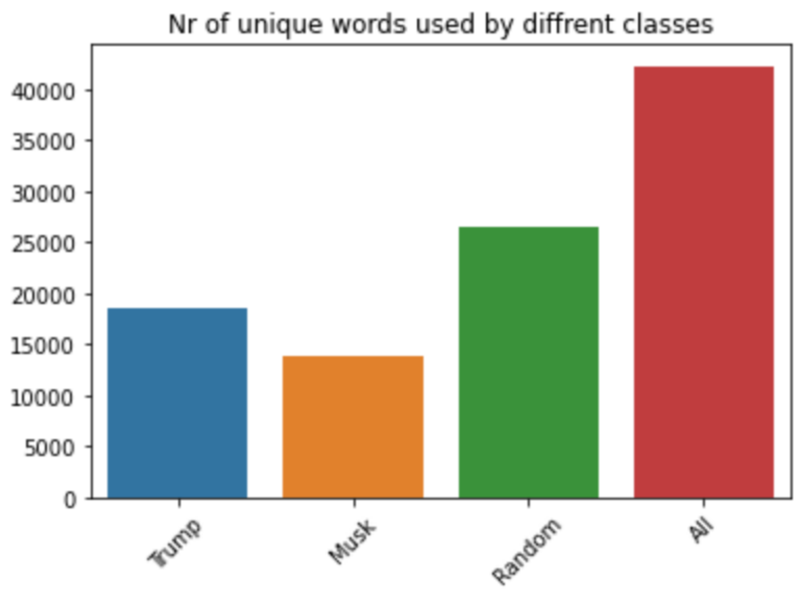


Fig. 6: The length of the vocabulary/number of unique words for each class and in the red bar for all classes together.

Further Descriptive Analysis

By looking at the word clouds one can clearly see differences in the 100 most used words of each class. This by itself is an indicator that a frequency based approach for classifying may be relatively good.

By looking at the plots of the top 10 words used in each class we can see that some words appear in two classes but no words appear in all of the classes. This can also be seen as an indicator that the words used in tweets differ quite a bit between the classes. If a tweet contains multiple words from Trump's top 10 words, chances are it will be classified as Trump.

Probability distribution

We are using text data and bag-of-words model. Therefore, the features we are interested in are the counts of each word (frequency) in the text. The models does not care for where in the text (a tweet) the word appears, it only cares for if the word appears.

X is a discrete random variable and each word in the vocabulary is a category. X follows a categorical distribution.

Parameter estimation

We have the number of categories which is the number of bars in our PMF, $k \in \{1,..., n\}$, and the probability of each category $p1,..., pk$, where $pi \geq 0, \sum_{i=1}^{n} pi = 1$.

We can estimate the number of categories by counting all of the words in our vocabulary, which in our data is: $k = 42315$. The probability for each category k is therefore:
$P(ki) = \frac{frequency\ of\ word\ in\ ki}{nr\ of\ words\ in\ total}$

Examples:

$$P(great) = \frac{3145}{742066} = 0.0042$$

$$P(tesla) = \frac{1272}{742066} = 0.0017$$

$$P(one) = \frac{1558}{742066} = 0.0021$$

Testing Differences in Tweet Length

After exploring the data we wanted to see if someone used fewer words in their tweets compared to the other classes. Based on our data exploration, we hypothesized it to be a difference between Elon Musk and Donald Trump.

H_0 = There is no significance between the tweet length in the classes (alpha = 0.05).

Furthermore, looking at the plots from earlier in the descriptive analysis, we suspected that the distributions are not equal between the classes. Moreover, the distributions for Trump and Musk do not look like gaussian distribution.

However, this does not really matter because of our large sample size. We have a i.i.d sample and we can make use of the CLT, and use a Welch t-test to compare the categories.

We did a Welch one-way ANOVA and found a significant difference between the groups (F = 1180.87, p < .01). Furthermore, we did subsequent Welch t-tests to investigate where the difference lied. These tests were corrected by the Bonferroni method, giving us an alpha of 0.0167. We found that Musk had significantly fewer words per tweet compared to Random (T = 24.71, p < .01). Moreover, we found that Trump writes significantly more words per tweet compared to Random (T = 35.91, p < .01). Thus, we can draw the conclusion that there is also a significant difference between Trump and Musk.

This difference could matter, since if Musk produces fewer words per tweet compared to the others, and has a smaller vocabulary. The model will be more likely to classify a tweet as his if it is a long tweet. Moreover, this information could be useful for using a classification approach such as a Gaussian naive bayes classifier or for gaussian mixture models.

Predictive analysis

We decided to use Multinomial naive Bayes (MnB) and Support vector machine (SVM) as our two predictive models. We wanted to use the MnB as it’s very simple which makes it interesting to see how well it can perform. We also tried Random forest models but the SVM models outperformed it on our data, which is why we decided to go with SVM.

MnB

Mathematical expression:

$$y = g(x_1, ..., x_n) = argmax(c \in \{trump, musk, random\})\ P(c) \prod_{i=1}^{n} P(x_i|c)$$

Parameters and estimation:

Vocabulary:
To build the vocabulary one simply makes a set of all words in the training data.

P(class) - prior:
To estimate prior for each class, we divide the number of tweets from each class by the total number of tweets. For example, $P(Trump) = \frac{\#\ of\ Trump\ tweets}{\#\ of\ total\ tweets}$

P(word|c) - The likelihood of a word given a class:
We use the vocabulary based on our training data and count how many times each word appears in each class. We then calculate the total number of words for each class and estimate the likelihood: $P(word_i|class_i)$.

Hyperparameters:

α - smoothing factor used in cases of zero likelihoods.

SVM

Parameters and estimation:

Hyperplane:
The basic function for the hyperplane (basic decision) looks like, f(x) = w^T x + b. Where x is the input data, w is the weight vector (for each feature), and b is bias. The T in the function corresponds to each feature (in our case word). We estimate the hyperplane by maximizing the margin between both classes, taking out hyperparameters into consideration.

Hyperparameters:

Kernel:
The kernel function is used to map the data into a higher-dimensional space, where it is linearly separable by a hyperplane with one dimension lower. There are different kinds of kernels. After playing around a bit, we decided to go with the RBF kernel.

$$K(X_1, X_2) = \exp(-\frac{\|X_1 - X_2\|^2}{2\sigma^2})$$

$K(X_1, X_2)$ is the expression for the kernel function using two data points. ||X1 - X2||^2 is the squared euclidean distance between these points. Sigma (σ) is a free parameter that is used to determine the width of the gaussian function.

Gamma:
Often the sigma is replaced with gamma (ɣ = 1 / 2σ^2). After introducing the gamma, the function can be written as: $K(X_1, X_2) = \exp(-\gamma \, \|X_1 - X_2\|^2)$.

Since gamma is a direct function of sigma it determines the width of the Gaussian function. A larger value of gamma leads to a narrower Gaussian, which results in a model that is more sensitive to individual data points. Gamma ranges between 0.0001 < gamma < 10.

C:
The C parameter controls the trade-off between the complexity of the model and the amount of training error allowed. The reason we use this is that in the real world, categories are rarely completely separable based on available data (i.e. there is often overlapping). The C we select decides how strict the model should be towards misclassifications when calculating the hyperplane. A larger C gives a stricter model. Thus more sensitive to the training data.

CountVectorizer/TfidfVectorizer

Before training the models we converted our text data into numerical vectors. This could be done by using sklearn CountVectorizer class or TfidfVectorizer class. The CountVectorizer converts the text documents to a matrix of token counts and the TfidfVectorizer converts it to a matrix of TF-IDF features.

The difference between CountVectorizer and TfidfVectorizer is that in Tfidf, “an inverse document frequency factor is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely” (from wikipedia).

We tested both models with both types of vectorization and the results were equal. We also decided to use the setting “min_df = 2”, min_df is short for minimum document frequency. This makes it so that we only train our model on words that occur in at least two documents (in our case tweets). Even though we cleaned our data there were a lot of words that only appeared in one tweet. Most of these were misspellings, sentences without spaces and other strange looking words. By setting “min_df = 2” we managed to reduce our vocabulary from approximately 30.000 words to 15.000 words.

We decided to run CountVectorizer for the MnB model and TfidfVectorizer for the SVM model.

Hyperparameter tuning

In order to find the best set of hyperparameters we used sklearns GridSearchCV class. For the MnB model we only needed to test one hyperparameter, α, and we found that using

$\alpha = 0.4$ or $\alpha = 0.5$ was the best, depending on if we used Count or Tfidf. For the SVM model we found that the best kernel to use were Radial basis function (rbf) with C = 3 and the default gamma value, scale, which according to the documentation is calculated as:

$$1 / (n\_features * X.var())$$

We did not try every unique combination of hyperparameters, i.e. we did not perform an exhaustive search. This would have been too computationally heavy for our computers and we did not look into using any cloud services. Instead, we started with the default values and tested our way forward, one hyperparameter at the time. As an example for the SVM model, we first tested what kernel was the best, and then we tested what C value was the best for that kernel etc.

Evaluation

Model A: Multinomial naive Bayes(alpha = 0.4) with CountVectorizer(min_df = 2).

Model B: Support vector machine(kernel = "rbf", C = 3, gamma = "scale") with TfidfVectorizer(min_df = 2).

We trained both classifiers using 10-fold cross-validation. As our evaluation metrics we used accuracy and macro-averaged f1-score. Below are the results obtained during the training.

Accuracy:

	fold 1	**fold 2**	**fold 3**	**fold 4**	**fold 5**	**fold 6**	**fold 7**	**fold 8**	**fold 9**	**fold 10**
MnB	0.8597	0.8605	0.8544	0.8388	0.8541	0.8485	0.8468	0.8509	0.8607	0.8541
SVM	0.8955	0.8924	0.8860	0.8836	0.8853	0.8907	0.8880	0.8919	0.8958	0.8919

Macro f1:

	fold 1	**fold 2**	**fold 3**	**fold 4**	**fold 5**	**fold 6**	**fold 7**	**fold 8**	**fold 9**	**fold 10**
MnB	0.8597	0.8604	0.8543	0.8387	0.8540	0.8485	0.8468	0.8508	0.8604	0.8541
SVM	0.8955	0.8924	0.8861	0.8836	0.8853	0.8907	0.8880	0.8919	0.8957	0.8918

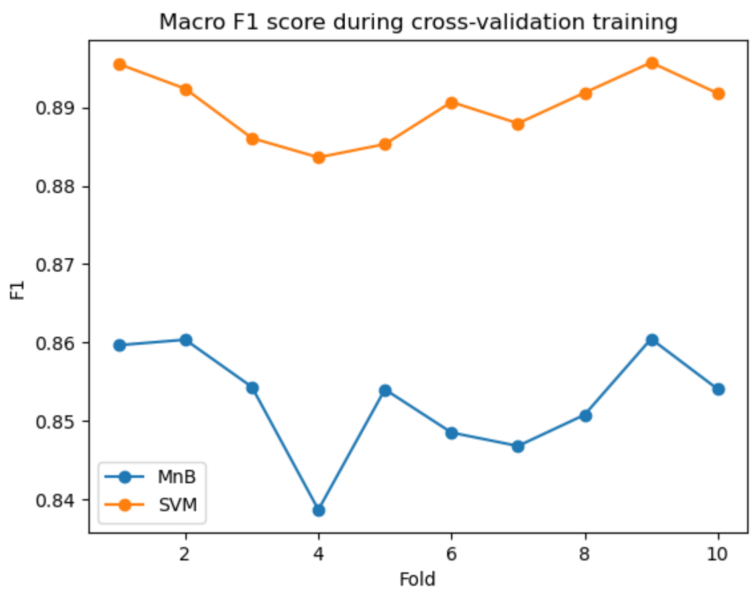


Fig. 7: 10-fold Cross-validation process during training. Blue line shows the MnB model macro f1-score (approx. 0.85) and the orange line shows the SVM models (approx. 0.89).

We decided to use the macro f1-score as our metric sample statistics and compared our classifiers with a paired t-test.

Classifiers: MnB(A) and SVM(B)

Data: Macro f1-score

Random variable and assumption: $P_1^A,..., P_{10}^A$, $P_1^B,..., P_{10}^B$. The differences $(P_i^A - P_i^B)$, are i.i.d and follow a gaussian distribution.

Parameter of interest: $\mu A - B$

Parameter estimate: $mA - B = \frac{1}{K}\sum_{i=1}^{K}(P_i^A - P_i^B)$

Hypotheses H0 and HA:

$H_0: \mu A - B = 0$,

$H_A: \mu A - B \neq 0$

Significance level/ α: 0.05

We used scipy.stats.ttest_rel in order to test our null hypothesis and the results we obtained were:

statistic = -24.2721, p-value = 1.6366e-09. We therefore reject the null hypothesis, there is a statistically significant difference in the results obtained from the two classifiers.

We also tested our assumption of the differences from the two classifiers following a gaussian distribution with scipy.stats.normaltest. This function tests the null hypothesis that a sample comes from a gaussian distribution. The results we obtained were:
statistic = 1.6915, p-value = 0.4292. We failed to reject the null and our assumption holds.

After having evaluated the classifiers performance we prefer the SVM model.

Applying final model on test data

The results from the final model applied on test data are presented below.

Overall accuracy: 89.18 %

Class	**Precision**	**Recall**	**F1-score**
Donald Trump	0.90	0.89	0.90
Elon Musk	0.90	0.89	0.89
Random	0.88	0.90	0.89

Fig. 8: Precision, recall and F1-Score all were about 0.89 for all classes.

The results obtained are very balanced between the classes.

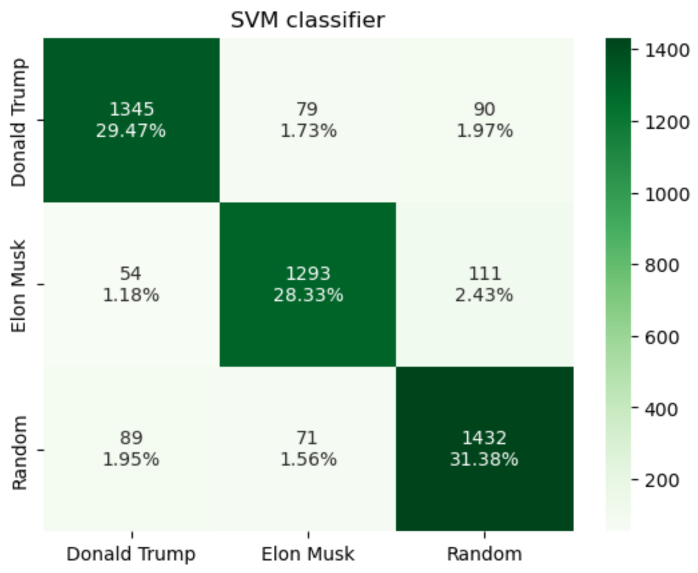


Fig. 9: Confusion matrix of the results obtained by using the SVM model on test data. Random being the most commonly predicted. The most common misclassification was classifying Musk as random.

By looking at the confusion matrix we can see that the Trump class had the most errors (169), but Musk (165) and Random (160) were not far behind. The most errors came from when the actual tweet was written by Musk but the model misclassified it as Random (111 or 2.43%).

Conclusion

In this project we used tweets from three different classes; {Trump, Musk, Random} in order to train a classifier and then try to predict which class a given tweet belongs to.

The motivation for us was primarily to work with text data as this was something we haven't done before. It was also a problem we thought would be funny to work on, which also was an important factor for us.

The motivation for a business to solve this problem could be to try and stop bots impersonating real people on Twitter or other social media sites. Our project wouldn't be the only thing to use but could play a part.

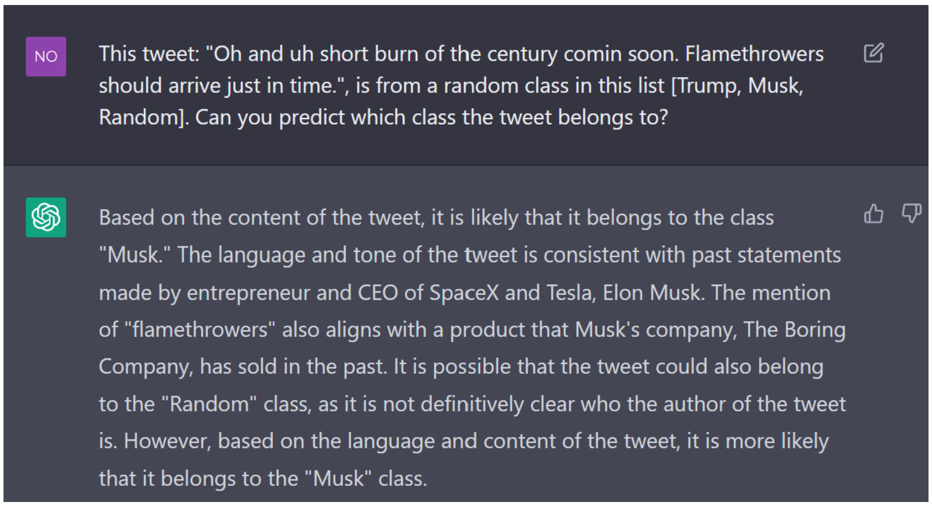
Both of the models we used to classify tweets performed better than we initially thought would be possible. The accuracy for our final SVM model was 89.17% which is quite high, however we don't have any similar models to compare this to. It is worth mentioning that it's not really possible to reach 100% accuracy on a problem like this since there will always be some tweets that are too difficult to classify. If we assume that this amount of tweets are between 2-5% of the total tweets then we have a very high accuracy on our model.

The reason why the results were better than initially expected might be because we simply overestimated the complexity of the problem, but we could also see in the descriptive analysis that the content of the tweets differed quite a bit. This makes it much easier for our model to classify which class the tweet belongs to.

The next step would be to gather more data and add more specific users to see if the model still performs the same. It's possible that adding more classes would be a problem for the bag-of-words models, especially if we add users with the same interests, i.e. tweets of the same subject. It would be interesting to see if our model would still perform well if we used three users interested in the same subject, such as three politicians, as our classes.

It would also be interesting to try to solve the same problem but with a more complex method such as neural networks. Methods such as word2vec which also capture word associations would probably perform well.

We tried some example tweets with ChatGPT which is a large language model chatbot and it managed to correctly predict most of them. It's interesting considering that ChatGPT is a large language model that hasn't specifically been trained on tweets but it can "understand" the content of the tweet and therefore make a prediction. See image below for an example:


NO

This tweet: "Oh and uh short burn of the century comin soon. Flamethrowers should arrive just in time.", is from a random class in this list [Trump, Musk, Random]. Can you predict which class the tweet belongs to?

Based on the content of the tweet, it is likely that it belongs to the class "Musk." The language and tone of the tweet is consistent with past statements made by entrepreneur and CEO of SpaceX and Tesla, Elon Musk. The mention of "flamethrowers" also aligns with a product that Musk's company, The Boring Company, has sold in the past. It is possible that the tweet could also belong to the "Random" class, as it is not definitively clear who the author of the tweet is. However, based on the language and content of the tweet, it is more likely that it belongs to the "Musk" class.

Fig. 10: Example of the large language model chatbot "ChatGPT" correctly predicting that Musk was the author of a tweet.

Source

Musk Tweets:
https://www.kaggle.com/datasets/ayhmrba/elon-musk-tweets-2010-2021?fbclid=IwAR1RLpjkuuuAvd6n1XBQSt8_p-i473HeJ7UB9oTzzIR_PBQZlQ3LSkbm-Ko

Trump Tweets (realDolandTrump):
https://www.kaggle.com/datasets/austinreese/trump-tweets?select=trumptweets.csv&fbclid=IwAR2DWn6UIsZNrJqwOee_vzyzV850gY_-YXI4AjFb3t_P9rbaJr2ydG8s4Ew

Random Tweets:
https://data.world/data-society/twitter-user-data?fbclid=IwAR1FcelZTZ-SpxwB0vn1Q7locTyLRMahdv6D_NkYujKccBBEFEwQHOAFJGo