Sentiment Analysis on Tweets
Experimental Analysis, Comparative Analysis, and Model Choice

Amer Elsheikh
The American University in Cairo
Cairo, Egypt
amer.elsheikh@aucegypt.edu
900196010

Abdallah Abdelaziz
The American University in Cairo
Cairo, Egypt
Abdallah_taha@aucegypt.edu
900196083

1. Experimental Analysis

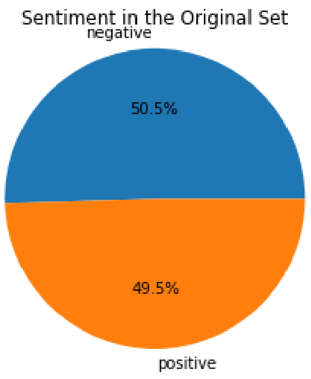
In this part, we draw the sample, test, and evaluate different supervised machine learning algorithms.

Sampling

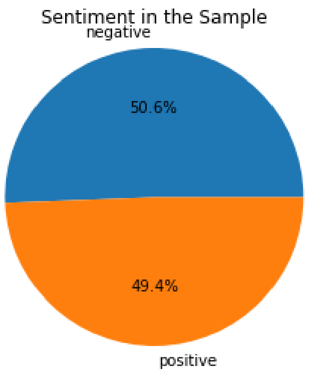
Since the dataset is big (more than 1.6M tweets), we decided to draw a sample that is representative of the dataset to test the performance metrics of different models in order to choose the best one. For that, we decided to take a random sample of **10% of the data, which is around 150k tweets,** believing that the sample would be representative for two reasons:

- Our dataset is huge, so big enough random sample should be representative.
- The polarity in our dataset was initially uniformly distributed with approximately 50-50 positive to negative, so any big enough random sample should still maintain that order that we do not need to use **stratified sampling.**

To make sure that our random sample actually preserved that order, we checked the distribution oof the polarity in the original data set and in the sample to find the following:

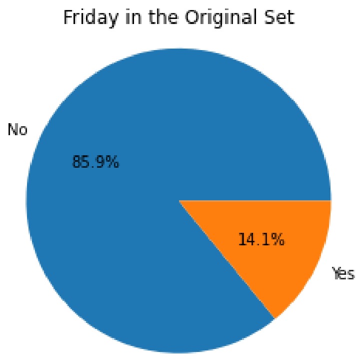


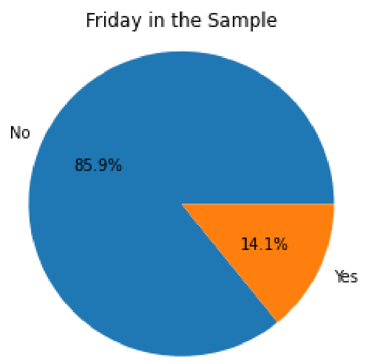
As compared to,



Which shows that the random sample totally preserved the distribution of our label, Polarity.

Moreover, the random sample preserved the distribution of all the features which ensures the sample is representative of the distribution. For example, charts below show the distribution of feature Friday, i.e., whether tweets were posted on Friday or not, in the original set vs. the sample:





Which shows that the sample preserved the distribution.

Lastly, we made sure that the sample chosen would be the same if the experiments were repeated again to ensure reproducibility. We did so by setting the parameter random_state parameter of the pandas function sample to an integer.

Re-representing the Features

In the last phase, we had the words for each tweet represented as a dictionary of frequencies. We did so instead of one hot encoding as we would have needed a tremendous amount of memory otherwise. However, now we have a smaller sample, we decided to represent this feature in a more memory using ways that will help us in experimenting different models. To do so, we used 3 different representation methods and tested the models on each one:

2. **Frequency Encoding:**
   We used bag of words but the value of each word in an instance is its frequency, not its occurrence. That was possible as our sample is moderate in size that after the encoding, its size turned out to be around 980 MB.
3. **Word2vec Embedding:**
   We used the pretrained model "glove-twitter-200", which was trained by Stanford scientists on more than 2B tweets converting more than 1.2M words, to embed each word into a vector of dimension 200 where similar words are closer to each other (i.e., have higher similarity). That Word2vec preserves the semantic relation between different words After calculating the Word2vec for each word in the tweet, we average those vectors to get a single vector for each tweet that will be used as the features of the tweet.
4. **Doc2vec**

Instead of averaging words embeddings of words in a paragraph, we tried a Paragraph Vector – Distributed Bag of Words (PV-DBOW) model. This model is analogous to Word2Vec Skip Gram. The paragraph vector is obtained by training a neural network to predict a target word from the full document's vector. This is suitable for the representation of our data as this model does **not** depend on the order of words.

Experimenting Models

We now experiment all the suitable supervised learning models on the three different representations of features to seek the best model:

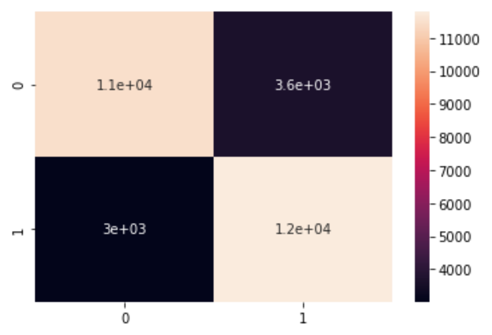
KNN model:

We chose not to experiment the KNN model as it is best suitable for small-midsized data sets; however, our original dataset is big, so the model will not be able to use all the training instances as it would need a tremendous amount of memory to do so. Moreover, the inference phase would require unreasonable amount of time as we need to calculate the pairwise distance between the test instance and all training instances, so that made the KNN option unreasonable.

Logistic Regression

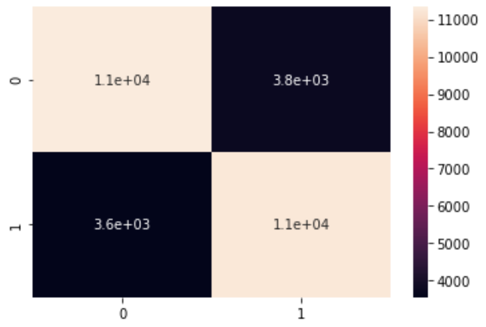
Frequency Encoding

The Logistic Regression was one of top performing models. The model is easy to interpret, easy to train and yields good results. The model does not assume any underlying distribution in the data which is one of its strengths and a reason for the consistent performance when changing data representation. The confusion matrix when data are represented in frequency encoding is given below. The model had accuracy of 78%.



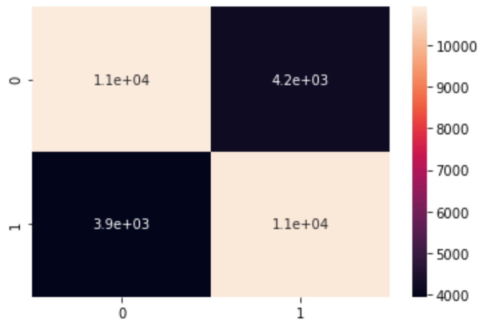
Word2Vec Embedding

Although this was expected to give a higher performance, this was not the case. This maybe due to the fact that embeddings were general and not specific to this dataset. The accuracy of the model was 75% and the confusion matrix is given below.



Doc2Vec Encoding

Again, this was supposed to give better performance, it was not the case. The Doc2Vec model was trained on the sample data. However, the sample data is not big enough and, hence, most probably tweets representations were not good enough. The accuracy of the model was 73%, and the confusion matrix is given below.



The table below summarizes the performance metrics for the three representations of data for logistic regression.

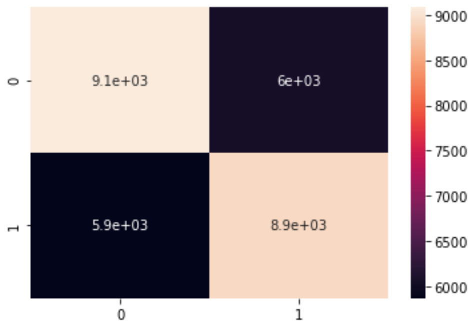
Features / Metrics	Frequency	Wrod2Vec	Doc2Vec
Precision	0.76	0.75	0.72
Recall	0.80	0.76	0.73
F1-score	0.78	0.75	0.73
Accuracy	0.78	0.75	0.73

The model fits the problem as it performs well, and it is known to have applications in the field of NLP.

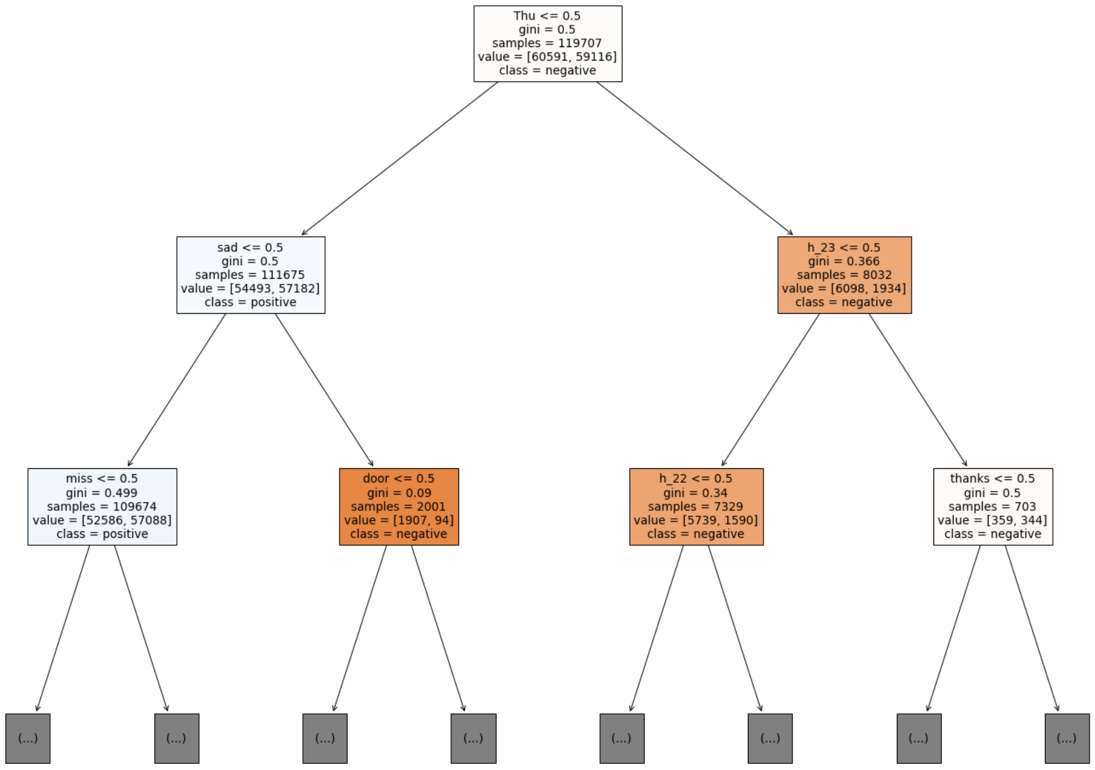
Decision Tree

Frequency Encoding

The decision tree is model suited for categorical data. The model depends on the information theory to decide features that give highest information gain. Since, tweets are small in size, the frequencies of the words are limited and hence, the model yield good results for this representation. The accuracy of the model is 69%. The confusion matrix is shown below.

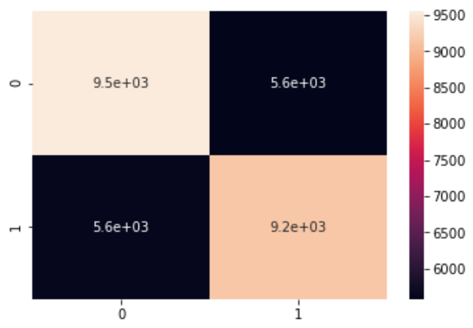


When investigating the constructed tree we see, as illustrated below, that whether the tweet is posted on Thursday is the feature with the highest information again followed by the negative word “sad”.



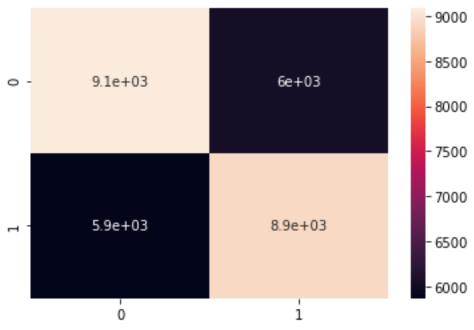
Word2Vec Embedding

When data was embedded, the performance of the decision tree dropped. This is reasonable as the algorithm is not very well suited to continues features. It is harder to find splits in this case than before. The accuracy of this model is 63% and the confusion matrix is shown.



Doc2Vec Embedding

Like Word2Vec embedding and for the same reason, the Doc2Vec embedding did not yield a very good result. The accuracy was 60%, the confusion matrix is shown below.



A table summarizing the performance metrics for decision tree is shown below.

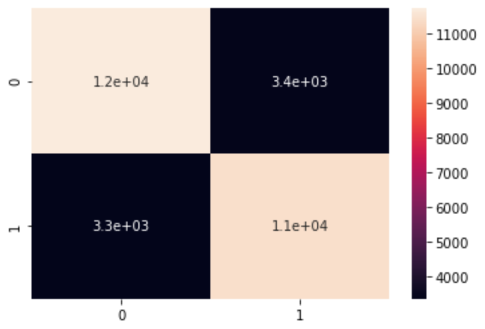
Features / Metrics	Frequency	Wrod2Vec	Doc2Vec
Precision	0.69	0.62	0.60
Recall	0.70	0.62	0.60
F1-score	0.69	0.62	0.60
Accuracy	0.69	0.63	0.60

The decision tree is not a good choice for the problem of sentiment analysis. This is clear from its performance and is also due the nature of the decision tree classifier as we have a lot of features, and the gain is not that significant for each word. In other words, information gain is the best way to use the features.

Naïve Bayes

Frequency Encoding

The Naïve Bayes model is a probabilistic model. The underlying assumption that features are conditionally independent given the label makes the models simple but efficient and robust to overfitting. Applying to our sample dataset, the results were promising. The accuracy of the model is 77%, and the confusion matrix is given below.

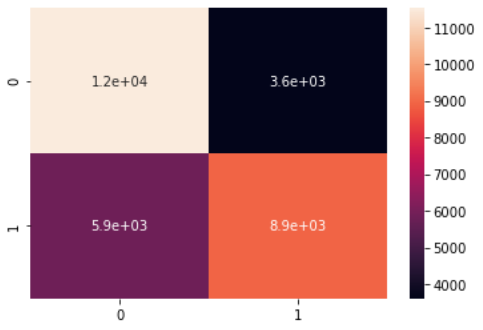


When seeing the associated probabilities after training, we find the in the op words probabilities given the label negative are “work”, “miss”, “back”, and “sad”. When the labels is given to be positive the words with highest probabilities include “good”, “love”, “like” and “lol”. It is worth mentioning that some days and hours are on the top of both lists.

Also, it should be noted that the model used is Multinomial Naive bayes which is suitable for text classification where sentences are represented as vectors of word counts of TFIDF.

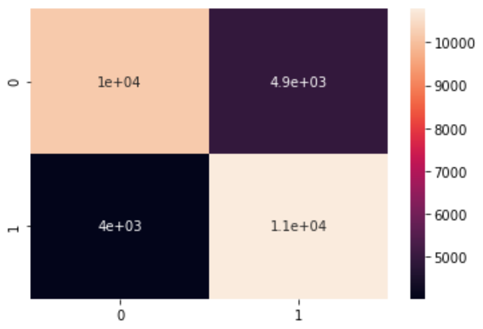
Word2Vec Embedding

When words are embedding the Multinomial Naive Bayes was not valid anymore and a Gaussian Naïve Bayes Classifier was used instead. This classifier assumes an underlying Gaussian distribution for the data. This can be approximately true under the condition of large enough data which is the case. However, because it is an approximation, the performance decreased to 68% which is a huge decrease. The confusion matrix is shown below.



Doc2Vec Embedding

For this model the accuracy increased and this maybe because data are more suited to the assumption about their distribution. The confusion matrix is shown below. The accuracy of this model is 70%.



The table below summarizes the performance metrics for the Naïve Bayes Model.

Features / Metrics	Frequency	Wrod2Vec	Doc2Vec
Precision	0.77	0.71	0.69
Recall	0.77	0.60	0.73
F1-score	0.77	0.65	0.71
Accuracy	0.77	0.68	0.70

Naïve bayes is good but not the best as it assumes conditional linear independence while this is clearly not the case in the problem of sentiment analysis.

Neural Network model:
For the Neural Network model, we used sklearn's Multi-layer Perceptron classifier (**MLPClassifier**), but such a parametric model needs a lot of tuning and experimentation to get the best results. To do so, we tried many values for the following parameters:

- Number of hidden layers: we tried 2 and 3 hidden layers, and both gave nearly same metrics
- Structure of the layers: we tried layers of sizes (16, 16), (16, 8), (32, 16), and (32, 16, 8) but all of them nearly gave the same metrics.
- learning_rate_init: we tried to put it with 0.01, 0.001, and 0.005, but also all of them gave similar matrices.

However, there was one important parameter that changed the performance of the MLPClassifier which is called "early_stopping". All in all, early_stopping has two cases:

- early_stopping = False, which is the default, makes the neural network repeats the training until either maximum number of predetermined iterations, max_iter, is reached or when the tolerance error is less than certain tolerance, tol.
- early_stopping = True makes the network uses 10% of the training data as validation set, and the networks terminates training when one of the two conditions mentioned in the previous point happens **OR** when the validation score is not improving by at least small tolerance value, tol, for 10 (predetermined number) consecutive iterations.

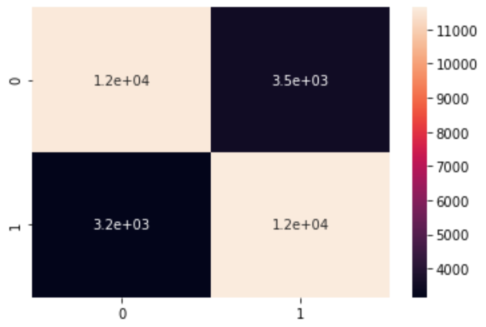
We tried the same MLPClassifier model two times on the features representation using **frequency encoding** with a single difference in early_stopping once set to True and another to False to get the following result:

Stopping / Metrics	False	True
Precision	0.73	0.77
Recall	0.74	0.79
F1-score	0.73	0.78
Accuracy	0.73	0.78

From the table, it is clear that setting "early_stopping" as True improves all metrics of the model. That is reasonable as it prevents the **overfit** that can be done over the training data using the validation set. That is clear in comparing number of iterations of both models: the first one, in which stopping is false, reached maximum number of iterations which is 30. On the other hand, the second model only did 12 iterations and stop which means it was about to **overfit.**

From the above parameters analysis, we decided to set early_stopping to True and try the model for the three types of features we determined:

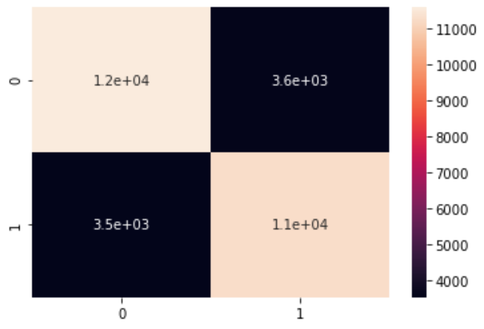
Frequency Encoding:
Neural network models consist of a set of layers feeding each other from the input layer up to the output layer, and through that process many weights and biases are used to calculate the ouput of each node. The model itself should learn these weights by minimizing an error function through backpropagation. Used with frequency encoding, the MLPC model gave the highest accuracy we



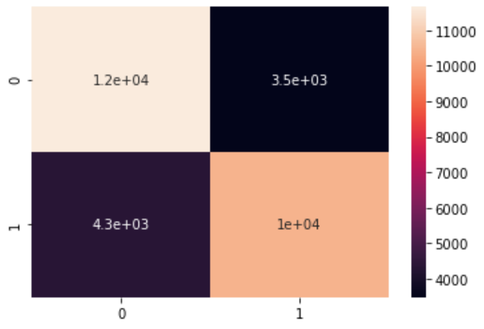
got along all models, which is 78%. The confusion matrix is shown below.

Word2Vec Embedding:
If the features were represented as Word2Vec, then the MLPC model would have accuracy of 76%. It is unexpected to get lower accuracy than frequency encoding. One reason for that is that embeddings used

embeddings were general and not specific to this dataset. The confusion matrix is shown below.



Doc2Vec Embedding
Lastly, if the features were represented as Doc2Vec, then the MLPC model would have accuracy of 74%. Similarly, the accuracy was expected to be better than the previous one, but the opposite happened. One reason for that is the sample data is not very big, and maybe the tweets vectors were not very reflective because of that. The confusion matrix is shown below.



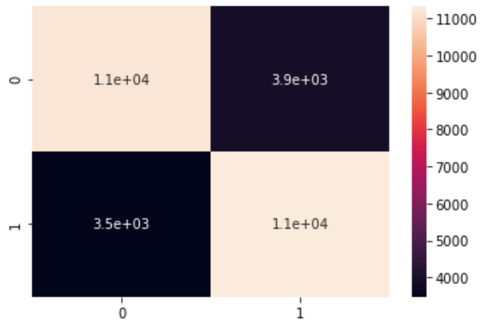
The table below summarizes the performance metrics for the Neural Network Model.

Features / Metrics	Frequency	Wrod2Vec	Doc2Vec
Precision	0.77	0.76	0.75
Recall	0.79	0.76	0.71
F1-score	0.78	0.76	0.73
Accuracy	0.78	0.76	0.74

Neural network is a good fit for this problem. Neural network is good for a variety of problems and input features in general.

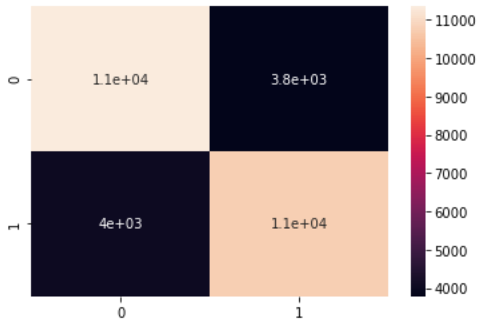
SVM model:
SVM models are supervised machine learning algorithms that use kernels to transform data into higher dimension space in which datapoints are separable. However, kernels such as polynomial are not suitable for large datasets. Hence, we use a linear kernel.

Frequency Encoding
The accuracy of this model is 74%. The confusion matrix is shown below.

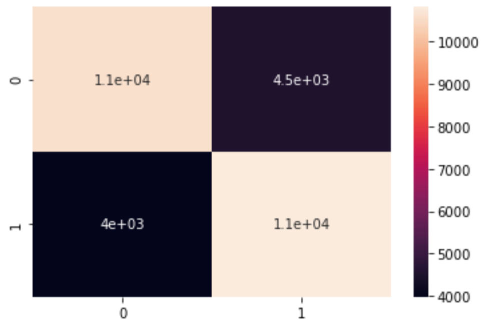


The accuracy is fair given that we are a using a simple linear kernel.

Word2Vec Embedding
The case is similar to other models as accuracy is lower than when data was represented using the frequency of the words. The accuracy of the model is 74%. The confusion matrix is also shown below.



Doc2Vec Embedding
The accuracy decreases again as we embed tweets using Doc2Vect model. It becomes 72%. And the confusion matrix is shown below.



The table below summarizes the performance metrics for the SVM model.

Features / Metrics	Frequency	Wrod2Vec	Doc2Vec
Precision	0.74	0.74	0.71
Recall	0.76	0.73	0.73
F1-score	0.75	0.73	0.72
Accuracy	0.75	0.74	0.72

The accuracies are generally high compared to other models considered in these experiments. The SVM fits the problem but putting using only linear kernels. As other kernels do not scale well.

Averaging the 3 accuracies for each model, we find that the best three models are Logistic Regression, Neural Networks and SVMs.
See the table below for the averaged accuracies.

Model	Logistic Regression	Decision Trees	Naïve Bayes	Neural Network	SVM
Average Accuracy	0.7533	0.6400	0.7167	0.7600	0.7367

2. Comparative Analysis

As mentioned before, the best three models are Neural Networks, Logistic Regression, and SVM. This is expected due to the nature of the problem in hand.

When comparing the accuracies of the three models with respect to different representations of the data we see a general trend as all models have the highest accuracy when words are represented as a frequency vector and the least accuracy when words are embedded using a Doc2Vec imbedding. However, we notice that Neural Network is the model least susceptible to this change. In other words, it does not change as much by changing the representation. It also has the highest accuracy for all data representations. This may be due the versatility of types of input a neural network can have and also because of the number of weights that can be learned to fit a wide range of problems. For logistic regression and SVM, the change in accuracies across different feature representations is identical but logistic regression has a better accuracy. This is intuitive because of two reasons:

- We are using a linear kernel for the SVM, so it is as if we are comparing linear discriminator to a logistic regression classifier.
- Logistic regression model works well with large datasets and is easier to scale.

For other performance metrics, variances are similar as well and generally are highest when words are represented in the frequency format and lowest when words are in the Doc2Vec format.

When considering training time, we find that the logistic regression is the best, followed by the neural network and then the SVM. This is under the assumption that a neural network stops training when it starts to overfit.

This leads to another point of comparison: overfitting. The neural network easily overfits. This is because of the huge number of weights it learns though the training. Neural network can memorize data instead of finding a generalization. In contrast, logistic regression generalizes well. Furthermore, it can be optimized through **regularization** to boost its ability to generalize and prevent it from overfitting. For linear SVM, overfitting is not a crucial problem as we are using a hyperplane to discriminate different points in the space.

One aspect of similarity is that all of them use gradient descent and are prone to finding local optimums instead of global ones.

One thing worth noting is that neural networks use a lot of memory space to store weights in contrast to logistic regression and SVM.

3. Model Choice

Considering the comparative analysis above and the nature of the problem, we decided to use logistic regression and to represent tweets as a vector of words frequencies. This decision was because of the following reasons:

1. Logistic regression is the simplest model among the three best models, yet it gives the highest accuracy when tweets are represented as a frequency vector.
2. Logistic regression is memory efficient and does not use a lot of memory space.
3. Logistic regression trains fast and does not overfit easily which is not the case with neural network and SVM.