COMP 4211 Project Report

Employing Word Embeddings and Machine Learning For Effective Fake News Detection

CHAN, Ka Chun (20952857) kcchancl@connect.ust.hk
LIU, Kwun Ho (20959984) khliuae@connect.ust.hk

1. Introduction

Fake news is not a new phenomenon, although it has historically been exploited for commercial marketing purposes. The rapid growth of the Internet and social media platforms like X, Pinterest, and Reddit has led to the dissemination of misinformation with political and commercial objectives globally. In the 2016 "PizzaGate" incident, conspiracy theorists misread hacked emails from Hillary Clinton's campaign chairman, resulting in violence at Comet Ping Pong, a pizzeria erroneously accused of child trafficking. Fake news is still popular throughout the US presidential election, such as accusations that Kamala Harris was not born in the US and hence cannot run for president. The widespread fake news has significantly disrupted a city's social stability.

Our group aims to improve the current scenario by developing a machine-learning algorithm for detecting fake news.

2. Description of the dataset and any preprocessing

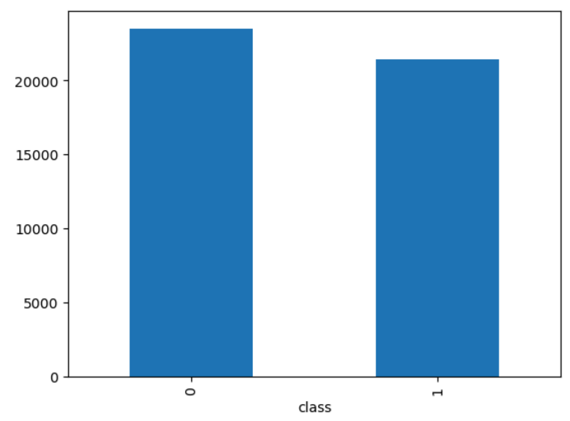
In this machine learning project, we will use a fake news dataset "Fake News Detection using Machine Learning" from Kaggle. In this dataset, we can find that there are 44919 instances of news articles in total, each of them having 6 features:

- **`Unnamed: 0`** (integers from 0 to 44918, the index)
- **`title`** (string)
- **`text`** (string)
- **`subject`** (string)
- **`date`** (date with format: {month} {day}, {year})
- **`class`** (int, 0 for real news; 1 for fake news)

For example, the first instance in the dataset is like this:

Unnamed:0	**title**	**text**	**subject**	**date**	**class**
0	Donald Trump Sends Out Embarrassing New Year'...	Donald Trump just couldn't wish all Americans...	News	December 31, 2017	0

We imported the dataset by calling **`news = pd.read_csv("News.csv")`**. We can have a look at the number of each label among the instances in the dataset.



From the figure shown above, we can see that the number of fake news is slightly lower than that of real news. However, the slight difference is acceptable, so we will directly use the dataset.

To process the dataset, we dropped the less relevant columns when determining whether a news article was a piece of fake news.

a. We dropped the **`Unnamed:0`** column since we used the **`pandas`** library to load the dataset and it has its own indexing system.
b. We dropped the **`title`** column because of two main reasons. First, it is easier for us to process the dataset when doing later machine-learning tasks. Second, hardly could we determine whether a news article is fake by solely referring to the title since it includes just a little information about the news. Something that appears in the title should also appear in the text. The information contained in the title should be a subset of the information contained in the text so we think that it is kind of redundant to keep the title column.
c. For the **`subject`** and **`date`** columns, we also dropped them as they are trivially less relevant to the authenticity of the news article. One may claim that the news article may be a piece of fake news if it is released in a certain period or if it belongs to certain categories. We have noticed this. However, in our project, it should be more meaningful to focus on the semantic meaning and the contents of the text in the news article rather than the background background like the date and the categories.

After the removal of certain columns in the dataset, we dropped the duplicated rows and shuffled the dataset to improve the generalization of our models. This is essential because the model might learn the patterns based on the sequential dependencies based on the order of the data and that is not desirable. After doing so, we get a resulting dataset with 38658 instances, each with 2 features (**`text`** and **`class`**).

Since we have two machine learning tasks - text embeddings and classification (they will be explained in the next section), we want to evaluate them more holistically. Therefore, we split out 10% of the dataset for evaluation purposes and named it as **`testing_set`**. The remaining is for training purposes and is named as **`dataset`**. So now, we have

`testing_set` with 3865 instances and **`dataset`** with 34793 instances. Once we have finished training the embedding models and the classifiers using **`dataset`**, we will use the trained embedding models to embed the texts of the instances in **`testing_set`** and apply the embeddings to the trained classifiers for evaluation.

For the word embedding methods, tokenization is needed and done as follows:

a. Filter out common English stopwords, e.g., "and", "the", "is", etc. as they carry absolutely no semantic meaning.
b. Remove the websites and some symbols like "!", ".", "...", ":", etc.
c. Do splitting wherever there's a whitespace.

3. <u>**Description of the machine-learning task(s) performed on the dataset**</u>

In this project, we have 2 staged machine-learning tasks:

a. **Text embeddings (unsupervised)**:
Input: a text in a news article
Output: an embedding (vector) representing the text

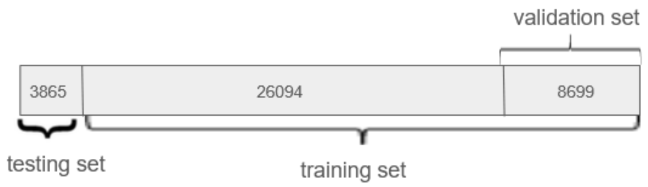
For text embeddings, we want to convert the text data into a numerical representation (vectors) that should capture the semantic, nuanced meanings and contextual relationships of the entire text in the news articles. Similar texts are represented by vectors that are close to each other in the embedding space. The closeness or similarity in the embedding space is defined by the Cosine Similarity

$$\theta = \cos^{-1}\left(\frac{\vec{v} \bullet \vec{u}}{||\vec{v}||\,||\vec{u}||}\right) = \cos^{-1}\left(\frac{\sum_{i=1}^{n} v_i u_i}{\sqrt{\sum_{i=1}^{n} v_i^2}\sqrt{\sum_{i=1}^{n} u_i^2}}\right)$$

where $\vec{v}$ and $\vec{u}$ are the embedding vector of two texts. When the embedding is well-defined, then the value of the cosine similarity θ should be small, when two texts have very different semantic meanings, and vice versa. The dot product is another way to define similarity in the embedding space.

b. **Classification (supervised)**:
Input: the embedding of text
Output: 0 or 1; where 0 means the text is real, otherwise it means it is fake

At this stage, within **`dataset`**, we will further split 25% of the instances as the validation set by calling **`train_test_split`** from **`sklearn`** package. This is, we have a **`testing_set`** of size 3865 and **`dataset`** (the training set) of size 34793. before the text embedding task. In the classification task, we further split out 8699 instances from the training set as the validation set.



For the classification task, we should focus more and prioritize improving the recall since we want to minimize the spread of fake news by ensuring most fake news articles are detected and classified as fake news by our model. Therefore in addition to accuracy, an ideal and robust model should have a good recall result on the **`testing_set`** so that the model can detect the fake news in a sample as many as possible.

4. **Machine learning method(s) used for solving the task(s)**

Throughout the experiments in the project, we used the basic Google Colab plan with T4 GPU for testing.
It seems that others have studied the dataset we are using in Kaggle. In others' implementation, for the text embedding task, only term-frequency-inverse document frequency is used and only logistic regression and decision tree are used for the classification task. In our project, in addition to these implementations, we will explore more options by adopting word embeddings (Continous Bag Of Words/ Skip-Gram[1]/ GloVe[2]) with weighted aggregation, Doc2Vec, and some pre-trained transformers (BERT/ GPT) for the text embedding task. For the classification task, on top of just using the logistic regression and the decision tree, we will use a feedforward neural network.

For text embedding task

a. **Word embeddings with weighted aggregation**

For word embeddings, we adopted Continous Bag Of Words and Skip-Gram algorithms to train the embedding models. Additionally, we use a set of pre-trained word embedding vectors from Global Vectors for Word Representation (GloVe) to compare to our locally trained models.

I. Continuous Bag Of Words (CBOW)

CBOW is an algorithm that learns the embeddings of the words (/tokens) by training a neural network with a single hidden layer to predict a target word based on the context words within a fixed window. In general, it has 2 hyperparameters: the window size and the embedding dimension. The embedding dimension is exactly the number of neurons in the hidden layer. The units in the input and output layers are both the number of tokens.

For CBOW, we first one-hot encode all the tokens in the texts in **`dataset`**. Within a text, given a target word, we select its surrounding context words within the window size. Then, we aggregate the one-hot vectors of the context words by sum or average and use the resulting vector as the input of the neural network.

[1] https://arxiv.org/pdf/1301.3781
[2] https://nlp.stanford.edu/projects/glove/

After forward propagation, we apply the softmax function in the output layer. Then, we use the one-hot vector of the target word to calculate the cross-entropy loss and do backward propagation to update the weights. After the convergence of the model, we can get the embedding of the target word in the hidden layer.

II. Skip-Gram (SG)
SG is an algorithm that learns the embeddings of the words (/tokens) by training a neural network with a single hidden layer to predict a set of context words by a single middle word surrounded by the context word. The hyperparameters of SG are the same as CBOW. The embedding dimension is also exactly the number of neurons in the hidden layer. We can see that SG is the reverse of CBOW.

For SG, same as CBOW, we first one-hot encode all the tokens in the texts in **`dataset`.** Then, we use the one-hot vector of the middle word. Do forward propagation and apply the softmax function in the output layer. For each context word one-hot vector, accumulate the cross-entropy loss and then do backward propagation. We can get the embedding of the middle word in the hidden layer after the convergence of the model.

For CBOW and SG, in practice, we just need to call the **`word2vec`** object from the **`gensim`** library.

III. GloVe
GloVe is an algorithm in which training is performed on aggregate global word-word co-occurrence statistics from a corpus and the resulting vectors have linear substances of vector space. In our implementation, we used the pre-trained word embeddings `glove.6B.50d`[3] where we have embedding of 6 billion tokens, each with 50-dimension.

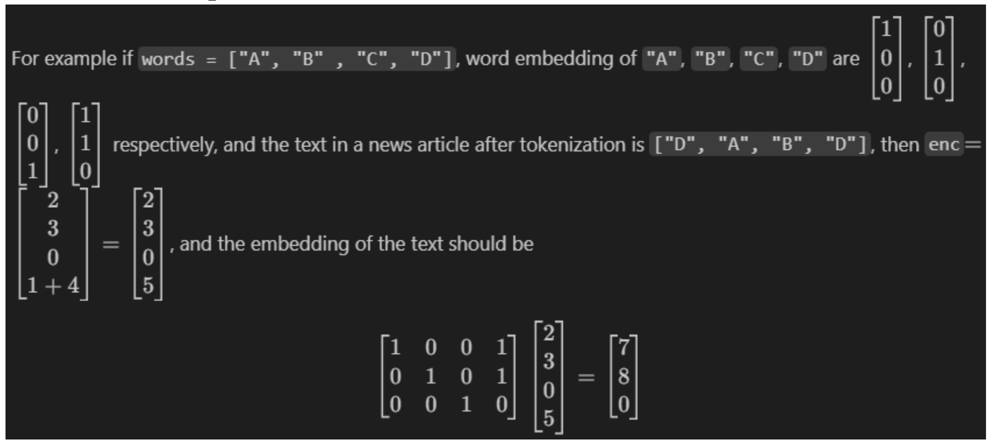
After getting the word embeddings from *I*, *II*, or *III*, we need to aggregate them together to get the text embedding for a new article as a new article has multiple words. The simplest way is to concatenate the word embeddings together. However, it has 2 disadvantages. First, a news article is very long (say 1000 words). The text embedding will be extremely long if we just simply do concatenation. Second, every news article is different in length. We need to do paddings and it should not work well.

Weighted Aggregation
After learning the d-dimensional embeddings of all the words/ tokens, let $\vec{b}_1, \vec{b}_2, \dots, \vec{b}_N$ be the word embeddings of $w_1,\ w_2,\ \dots,\ w_N$ respectively, where $\vec{b}_i\ \in\ R^d$ and N is the number of words in the set. Then, we define the encoding vector **`enc`** to be a vector with dimension $=\ N$ such that

[3] https://nlp.stanford.edu/data/glove.6B.zip

enc[i] = sum of positions that w_i appear in the text.

Here is an example for illustration:



For example if `words = ["A", "B" , "C", "D"]`, word embedding of `"A"`, `"B"`, `"C"`, `"D"` are $\begin{bmatrix}1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\end{bmatrix}$, $\begin{bmatrix}1\\1\\0\end{bmatrix}$ respectively, and the text in a news article after tokenization is `["D", "A", "B", "D"]`, then `enc` $= \begin{bmatrix}2\\3\\0\\1+4\end{bmatrix} = \begin{bmatrix}2\\3\\0\\5\end{bmatrix}$, and the embedding of the text should be

$$\begin{bmatrix}1 & 0 & 0 & 1\\0 & 1 & 0 & 1\\0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}2\\3\\0\\5\end{bmatrix} = \begin{bmatrix}7\\8\\0\end{bmatrix}$$

With this strategy, not only the ordering information of the words in the text can be preserved but the length of the resulting vector is also kept the same as the dimension of the word embeddings.

b. Doc2Vec

Doc2Vec is an extension of CBOW/ Skip-gram. Instead of embedding every single word, Doc2Vec directly embeds the whole text/ document into a vector.

I. Distributed Memory (DM)

Similar to CBOW, it uses the context words to predict the target word. Additionally, the randomly initialized document identifiers are also used to concatenate to the hidden layer. Do gradient descent with respect to the weights and the inputs by backward propagation. Upon convergence, we can use the document identifier as the text embedding.

II. Distributed Bag Of Word (DBOW)

Similar to skip-gram, instead of the middle word, it uses the document identifier to predict the context words. Then, do gradient descent with respect to the weights and the input to update the document identifier and use it as the text embedding.

In practice, we just need to call the **`Doc2Vec`** object in the **`gensim`** library.

c. Term-frequency-inverse document frequency

Term-frequency-inverse document frequency (Tf-idf) is another word embedding method similar to CBOW. The scope of Tf-idf is to measure the importance of words to documents in a dataset.

Unlike CBOW, Tf-idf goes a step further by considering the importance of words in the context of the entire corpus. TF-IDF algorithm assigns weights to words based on their frequency in a document (Term Frequency) and their rarity across the corpus (Inverse Document Frequency). The final Tf-idf value is the product of the two terms.

Term Frequency
Represents the frequency of a word `t` in a document `d`.
`Tf(d,t) = (count of t in d)/(number of words in d)`
After obtaining all the term frequency values, normalization is needed to ensure fairness given that different news have different text lengths.

Inverse Document Frequency
We first need to define *Document Frequency,* which is the number of documents containing the word `t` divided by the number of documents in the corpus(dataset). Then, calculate the inverse of the *Document Frequency* and take the `log` value of it to obtain the *Inverse Document Frequency*.
`Idf(d) = log(# of d/DF(d))`

In our use case, we would utilize the Tf-idf vectorizer in scikit-learn to assist us in converting the news' text input into different text vectors, similar to Word2Vec. The text vectors obtained by the vectorizer would be then used to train our classifier models.

d. **Pre-trained transformers**
The concept of transformers has been introduced in the lectures, as the lecture note suggests, transformer models are a type of deep learning model that is used for natural language processing (NLP) tasks. They can learn long-range dependencies between words in a sentence and form contextual outputs based on the inputs given.

Given how advanced and powerful modern transformers are, we are interested in comparing pre-trained transformers with our previous word embedding methods. In our project, we would use and fine-tune the BERT model and the GPT model for the tasks, using the models' tokenizers to obtain embedding output and feed it into our classification layers. With BERT being an encoder-only architecture and GPT being a decoder-only architecture, the output results would be interesting to see and further explore.

In practice, we would import both the BERT and the GPT2 model and their respective tokenizers in the `transformer` library.

For classification tasks
After text embeddings, we can apply multiple classifiers to do the binary classification task. For consistency, we keep the architectures of the classifiers the same throughout the experiment.

a. Logistic regression
b. Decision tree
c. Feedforward neural network
   I. single hidden layer
   II. 100 neurons

III. RELU activation function
IV. Optimizer: Adam

5. Experiments and results

a. Word embeddings with weighted aggregation

The flow of our experiment 1

Hyperparameter: embedding dimension = 100, windows size = 10, max_iter = 10000

i. Train the text embeddings by CBOW/ SG by using `Word2Vec` from `Gensim` library as described in the last section using `dataset` (the whole training set) and get the text embedding on each of the instances in `dataset` by weighted aggregation.
ii. Do train test split on `dataset` with a 25% test ratio, called the test split as the validation set
iii. Train the classifiers (logistic regression/ decision tree/ MLP) using the train split of `dataset`
iv. Evaluate the classifier using the validation set
v. Evaluate the classifier and the embedding using the `testing set`

Experimental result

Logistic regression				
	Validation set		**Testing set**	
	CBOW	**SG**	**CBOW**	**SG**
Accuracy	**0.93**	**0.93**	**0.76**	**0.77**
Recall	**0.98**	**0.98**	**0.62**	**0.64**
Precision	**0.91**	**0.91**	**0.92**	**0.93**
F1-score	**0.94**	**0.94**	**0.74**	**0.76**

Decision Tree				
	Validation set		**Testing set**	
	CBOW	**SG**	**CBOW**	**SG**
Accuracy	**0.90**	**0.93**	**0.79**	**0.76**
Recall	**0.91**	**0.98**	**0.65**	**0.60**
Precision	**0.91**	**0.91**	**0.94**	**0.95**

F1-score	**0.91**	**0.94**	**0.77**	**0.73**

MLP				
	Validation set		**Testing set**	
	CBOW	**SG**	**CBOW**	**SG**
Accuracy	**0.96**	**0.93**	**0.90**	**0.89**
Recall	**0.99**	**0.98**	**0.86**	**0.81**
Precision	**0.96**	**0.91**	**0.95**	**0.98**
F1-score	**0.97**	**0.94**	**0.90**	**0.89**

Observations

i. Low performance on the testing set is relatively low compared to the validation set. This can be easily explained by the fact the embeddings are not trained using the testing set. It implies that the text embeddings are not generalized enough.

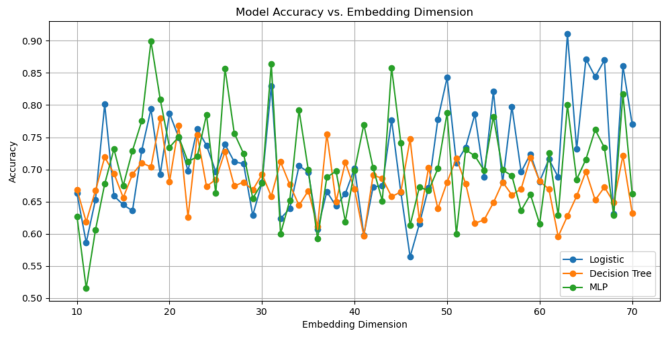
ii. MLP has a relatively good performance on the testing set.

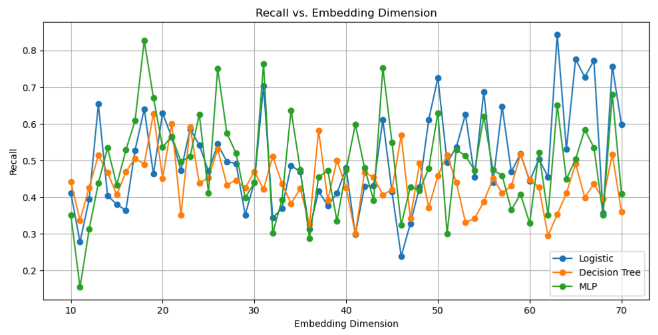
The flow of our experiment 2

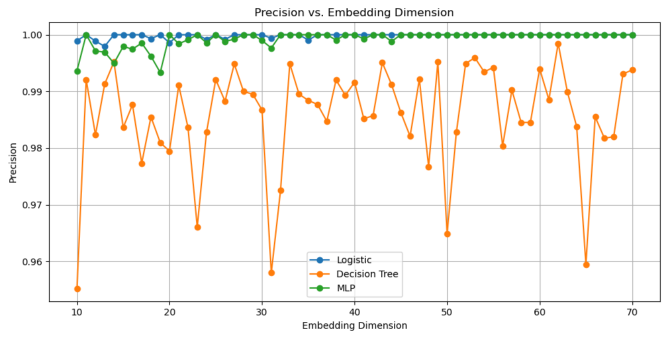
In this experiment, we adjusted the embedding dimension **from 10 to 70**. We observe the variation of the performance in terms of accuracy, recall, precision, and the F1-score. The flow is similar to that of experiment 1 just that we didn't do evaluation the validation set.

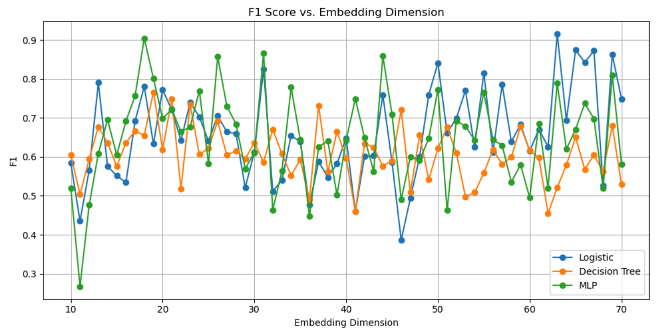
Experimental result

i. CBOW

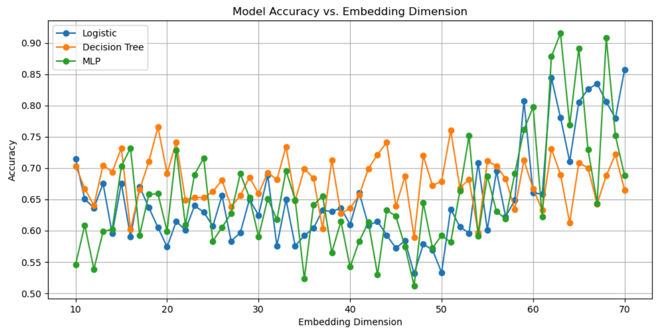


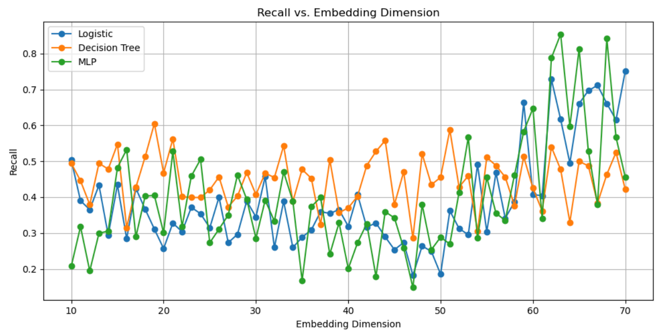


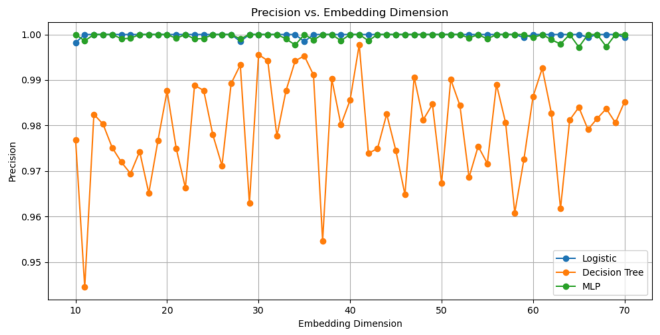


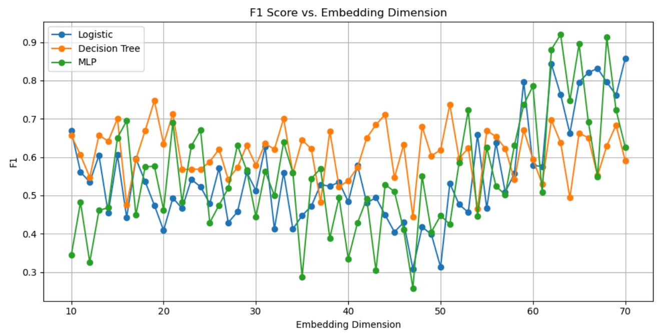


ii. SG









Observations

i. Using each of the CBOW and SG strategies, for logistic regression, decision tree, and feedforward neural network, the trend of the accuracy, recall, and F1 score is the same along the embedding dimension.

ii. For both CBOW and SG, if we use logistic regression and the feedforward neural network, the precision is very stable at around 1 along the embedding dimension. There is a little fluctuation (above 0.94) in precision if we adopt the decision tree.

The flow of our experiment 3

We now experiment on GloVe pre-trained vectors which are in 50-dimension. The setup is similar to experiment 1. The difference is just we do not need to train the embeddings.

Experimental result

Logistic regression			**Decision Tree**		
	Validation set	**Testing set**		**Validation set**	**Testing set**
Accuracy	**0.87**	**0.55**	**Accuracy**	**0.85**	**0.51**
Recall	**0.94**	**1.00**	**Recall**	**0.87**	**0.78**
Precision	**0.85**	**0.55**	**Precision**	**0.87**	**0.54**

F1-score	**0.89**	**0.71**	**F1-score**	**0.87**	**0.64**

MLP		
	Validation set	**Testing set**
Accuracy	**0.91**	**0.55**
Recall	**0.97**	**1.00**
Precision	**0.88**	**0.55**
F1-score	**0.92**	**0.71**

Observations

i. Overall (both validation and testing set), the performance is worse than the text embedding by CBOW and SG.
ii. We always get low accuracy $\approx\ 0.5$, low recall $\approx\ 0.5$, and high recall $\approx\ 1.0$ in the results of the three classifiers. We can infer that the model always predicts the news articles as fake news.

Conclusions from experiment 1, 2, 3

i. In our use case, we should emphasize more on the recall than the precision. According to the result we get, if we use CBOW, we should use high-dimension embeddings and logistic regression. If we use SG, we should use high-dimensional embeddings and the feedforward neural network.
ii. The pre-trained embeddings (GloVe) might not apply to our use case since we are likely to get too "skeptical" classifiers which classify the news articles as fake news most of the time. We shall train our own text embeddings instead of using the pre-trained ones.

b. Doc2Vec

The flow of our experiment

Same as the experiment 1 in the last section. The only difference is that we train the text embedding by DM/ DBOW by **`Doc2Vec`** from **`Gensim`** library with 50-dimension embeddings.

Experimental result

Logistic regression				
	Validation set		**Testing set**	
	DM	**DBOW**	**DM**	**DBOW**

Accuracy	**0.93**	**0.99**	**0.93**	**0.99**
Recall	**0.95**	**1.00**	**0.95**	**1.00**
Precision	**0.93**	**0.99**	**0.93**	**0.99**
F1-score	**0.94**	**1.00**	**0.94**	**0.99**

Decision Tree				
	Validation set		**Testing set**	
	DM	**DBOW**	**DM**	**DBOW**
Accuracy	**0.84**	**0.93**	**0.82**	**0.94**
Recall	**0.85**	**0.95**	**0.83**	**0.94**
Precision	**0.85**	**0.94**	**0.85**	**0.95**
F1-score	**0.85**	**0.95**	**0.84**	**0.95**

MLP				
	Validation set		**Testing set**	
	DM	**DBOW**	**DM**	**DBOW**
Accuracy	**0.95**	**1.00**	**0.94**	**1.00**
Recall	**0.95**	**1.00**	**0.94**	**0.99**
Precision	**0.95**	**1.00**	**0.97**	**1.00**
F1-score	**0.95**	**1.00**	**0.95**	**1.00**

Observations/ conclusions

i. Doc2Vec has much better overall performance than word embedding with weighted aggregation. The overfitting problem that appeared in experiment 1 in the last section doesn't occur here. The model is well-generalized as the performance on the validation set and testing set are similar.

ii. Almost all texting embeddings by Doc2Vec and classifier combinations have decent performance (all metrics > 0.9), except the DM+Decision Tree combination got all metrics around 0.85.

c. Term Frequency Inverse Document Frequency

The flow of our experiment

Similar fashion as the previous parts. In this part, we will obtain the text embedding by `TfidfVectorizer` from `Scikit-learn` library.

Experimental result

Logistic regression			**Decision Tree**		
	Validation set	**Testing set**		**Validation set**	**Testing set**
Accuracy	**0.98**	**0.98**	**Accuracy**	**0.99**	**0.99**
Recall	**0.98**	**0.98**	**Recall**	**0.99**	**0.99**
Precision	**0.98**	**0.98**	**Precision**	**0.99**	**0.99**
F1-score	**0.98**	**0.98**	**F1-score**	**0.99**	**0.99**

MLP		
	Validation set	**Testing set**
Accuracy	**0.98**	**0.98**
Recall	**0.98**	**0.98**
Precision	**0.98**	**0.98**
F1-score	**0.98**	**0.98**

Observations/ conclusions

i. Tf-idf has presented an overall consistent and great result compared to the previous methodologies such as Word2Vec and Doc2Vec, possibly due to the Tf-idf architecture eliminating the drawback of CBOW. There are nearly no differences between the three types of classifiers for Tf-idf.

ii. Besides the accuracy, the recall performance is also consistent and has a high value overall, which shows that the number of true predictions is high and not skewed to one class.

iii. There is a slight advantage in accuracy for decision trees compared to logistic regression and MLP, we assumed the reason is due to Tf-idf Vectorizer creating a sparse representation of the text inputs, which may benefit the learning of decision trees with its simplicity and more interpretability.

d. **Pre-trained Transformers**

The flow of our experiment

In this part, we will import the pre-trained transformer models **`BERT`** and **`GPT2`** from **`transformers`** library, and utilise the tokenizers to obtain the text embeddings. Embedded inputs are fed into classifier layers similar to previous methods.

Experimental result

Logistic regression				
	Validation set		**Testing set**	
	BERT	**GPT**	**BERT**	**GPT**
Accuracy	**0.99**	**1.00**	**0.99**	**1.00**
Recall	**0.99**	**1.00**	**0.99**	**1.00**
Precision	**0.99**	**1.00**	**0.99**	**1.00**
F1-score	**0.99**	**1.00**	**0.99**	**1.00**

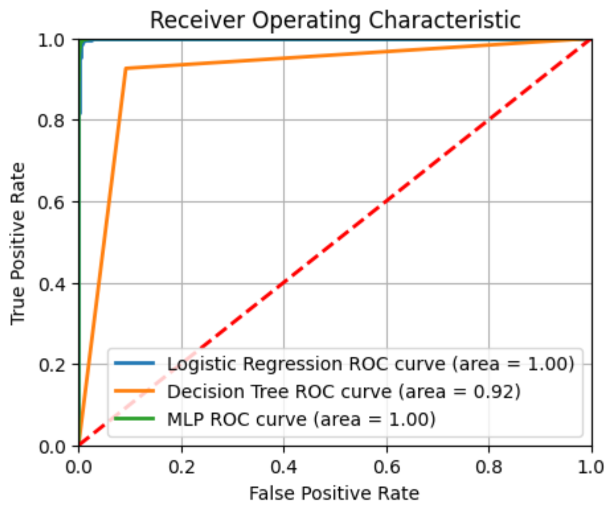
Decision Tree				
	Validation set		**Testing set**	
	BERT	**GPT**	**BERT**	**GPT**
Accuracy	**0.91**	**0.97**	**0.92**	**0.96**
Recall	**0.92**	**0.97**	**0.92**	**0.96**
Precision	**0.92**	**0.97**	**0.92**	**0.96**
F1-score	**0.92**	**0.97**	**0.92**	**0.96**

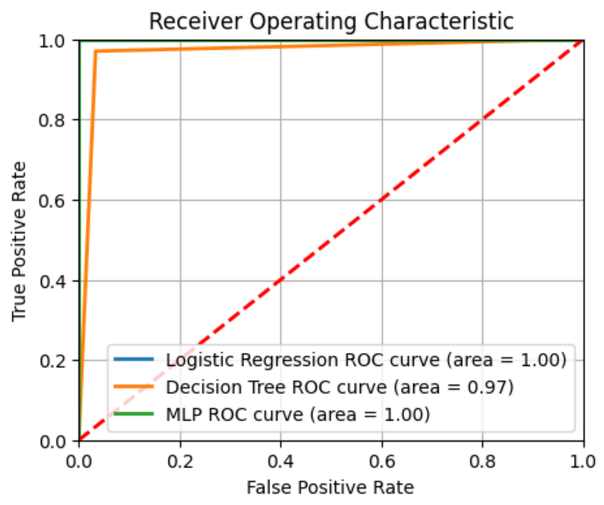
MLP				
	Validation set		**Testing set**	
	BERT	**GPT**	**BERT**	**GPT**
Accuracy	**0.99**	**1.00**	**0.99**	**1.00**
Recall	**0.99**	**1.00**	**0.99**	**1.00**
Precision	**0.99**	**1.00**	**0.99**	**1.00**

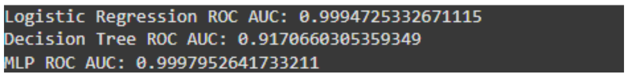
F1-score	**0.99**	**1.00**	**0.99**	**1.00**

ROC-AUC graph for

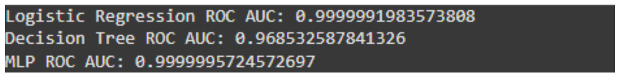
BERT model (Left) and **GPT** model (Right)







```
Logistic Regression ROC AUC: 0.9994725332671115
Decision Tree ROC AUC: 0.9170660305359349
MLP ROC AUC: 0.9997952641733211
```



```
Logistic Regression ROC AUC: 0.9999991983573808
Decision Tree ROC AUC: 0.968532587841326
MLP ROC AUC: 0.9999995724572697
```

Observations/ conclusions

i. For logistic regression and MLP classifier, consistent and excellent performance can be observed which is similar to Tf-idf. However, there is a significant drop in both accuracy and recall value when it comes to the decision tree classifier.

ii. The performance drop is also reflected in the ROC-AUC graph, where the decision tree model tends to misclassify much more data. Further experiment in the decision tree model is needed.

iii. Another major thing that is observed is the amount of time taken to obtain the text embeddings using BERT's tokenizer and GPT's tokenizer. Around 22 minutes is needed for BERT and 19 minutes for GPT, which is a significant increase compared to previous methodologies such as Tf-idf only takes less than a minute.

iv. After considering different perspectives of the model and its outcome performance, we have come to a conclusion that it may not be efficient enough compared to methods like Tf-idf or Doc2Vec, the minimal performance increase is not worth the extra training time for this project. However, for the scenario of training on a relatively small dataset, where previous methods might not be able to train and converge to a high-performance value, that way fine-tuning a pre-trained transformer for the accuracy boost might be worth it.

Further experiments/explorations

i. Given the relatively poor result for the decision tree model in BERT, we have decided to investigate the matter and try to further explore ways to improve the model.
ii. Due to the nature of transformers, where contextual meanings of the training dataset are also learned and put in the embedding text output, we assumed the reason causing the poor accuracy in the decision tree is the complexity of embedding text outputs. Those new texts after going through BERT model are:
    a. High-dimensional
    b. Contextually rich
    c. High Complexity

    The decision tree might not be able to generalise well, causing overfitting.
iii. Thus, we considered reducing the dimensionality of the embedding outputs with Principal Component Analysis (PCA with n_components = 25), and using a Random Forest (essentially multiple decision trees) instead of a Decision Tree to classify.

Experimental Results			
	BERT + Decision Tree (Original)	**BERT (PCA=25) + Decision Tree**	**BERT (PCA=25) (Random forest)**
Accuracy	**0.91**	**0.946**	**0.966**
Recall	**0.92**	**0.95**	**0.97**
Precision	**0.92**	**0.95**	**0.97**
F1-score	**0.92**	**0.95**	**0.97**

Experiment conclusions

i. By making simple adjustments to the embedding text vectors from the BERT model, we could already see large improvements in accuracy and recall factor.
ii. We concluded that by reducing the dimensionality of the embedded texts, the decision tree/random forest models would learn and generalise more easily, obtaining better classification results.
iii. However, more parameter tuning would be required in order to achieve the same performance as Logistic regression or MLP. We believe that it may not be as feasible to train and tune in other real-life scenarios, and the decision tree might just not be a good classification layer to add on top of the pre-trained transformers, instead, logistic regression or MLP should be considered for a word-embedding binary classification task like this project.

6. **Division of labor in teamwork**

CHAN, Ka Chun (contribution: 50%)

a. Doc2Vec
    - Presentation
b. Term-frequency-inverse document frequency
    - Code, Report, Presentation
c. Pre-trained transformation (BERT/ GPT)
    - Code, Report, Presentation

LIU, Kwun Ho (contribution: 50%)

a. Data preprocessing
    - Code, Report, Presentation
b. Word embeddings (CBOW/ SG/ GloVe) with weighted aggregation
    - Code, Report, Presentation
c. Doc2Vec
    - Code, Report