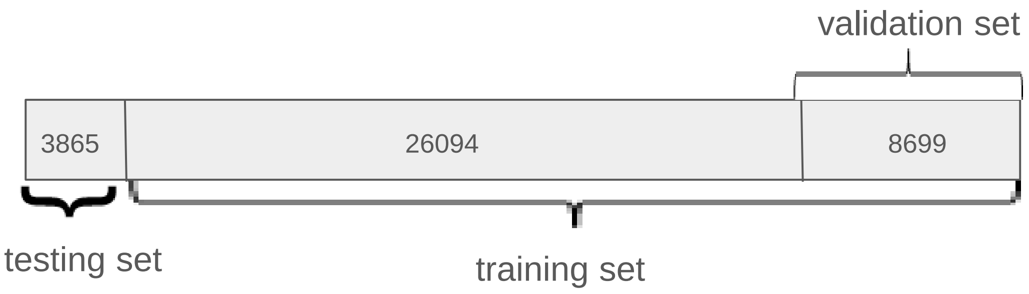
Employing Word Embeddings and Machine Learning For Effective Fake News Detection

CHAN, Ka Chun (20952857) LIU, Kwun Ho (20959984)

Dataset

- **[News dataset](#) (44919 rows)**
  - title
  - text
  - subject
  - date
  - class



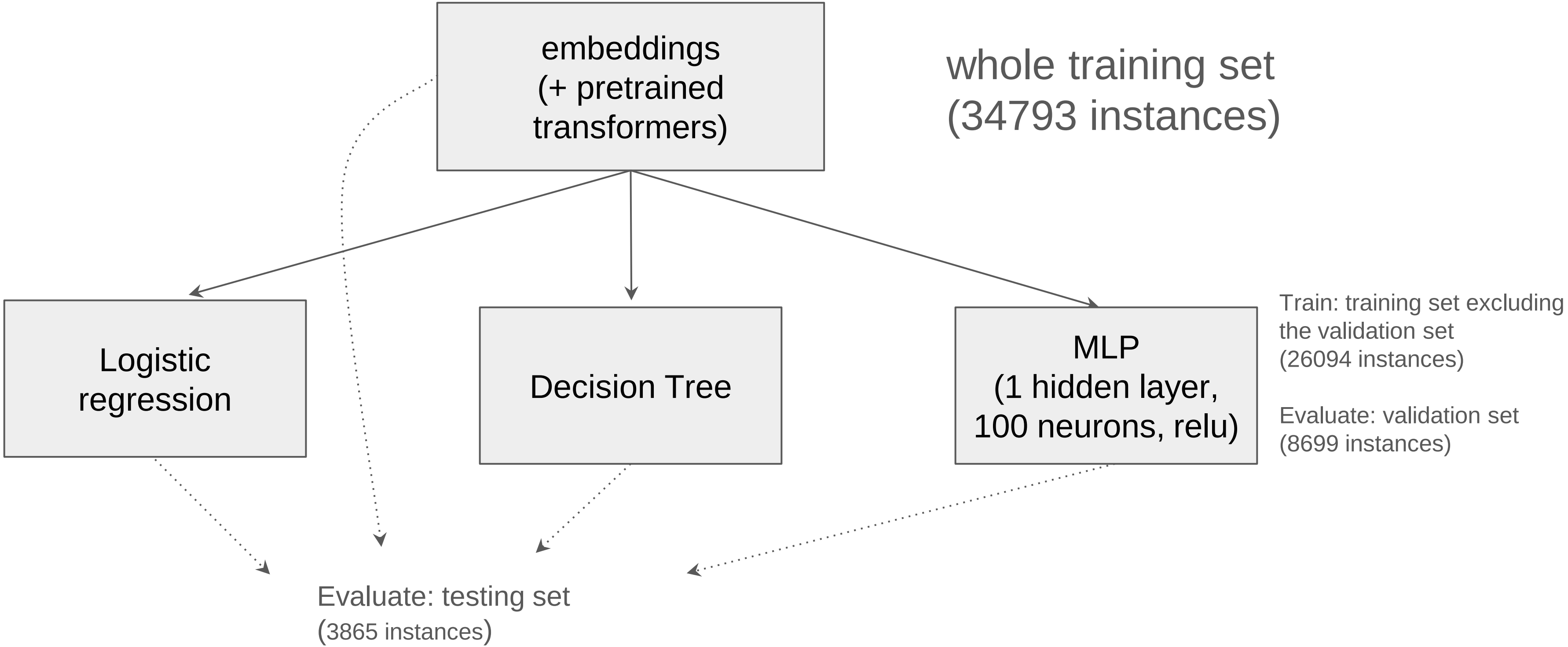
- We will keep columns `text` and `class`.
- Drop the duplicate rows: 38658 rows
- Split dataset into training set (34793 rows) and testing set (3865 rows) with testing ratio = 0.1 before doing word embeddings.
- In the training set (34793 rows), 25% of them (8699 rows) forms a validation set.

Machine Learning Tasks

- **Embeddings:** convert the text in the news article into vectors which can be fitted into classifiers
  - Word embeddings
  - Doc2vec
  - Term Frequency Inverse Document Frequency
  - Pre-trained Large Language Models
- **Classification:** take an embedded text (vector) in a news article as the input, classify it as either fake news (1) or not (0).
  - Logistic Regression
  - Decision Tree
  - MLP

Machine Learning Tasks

- Workflow



Word embeddings (Continuous Bag Of Words)

- Continuous Bag Of Words (CBOW)

  Learns the embeddings by training a neural network with a single hidden layer to predict a target word based on the context words within a fixed window.

For example: window size = 2,

“ I am <u>studying</u> machine learning”

Word embeddings (Continuous Bag Of Words)

For example: window size = 2,

“ I am studying machine learning”

Context words: {“I”, “am”, “machine”, “learning”}

“I” : (1, 0, 0, 0, 0) = e1
“am” : (0, 1, 0, 0, 0) = e2
“studying”: (0, 0, 1, 0, 0) = e3
“machine”: (0, 0, 0, 1, 0) = e4
“learning”: (0, 0, 0, 0, 1) = e5

Aggregate context vectors by sum/ mean:

e1 + e2 + e4 + e5 = (1, 1, 0, 1, 1)

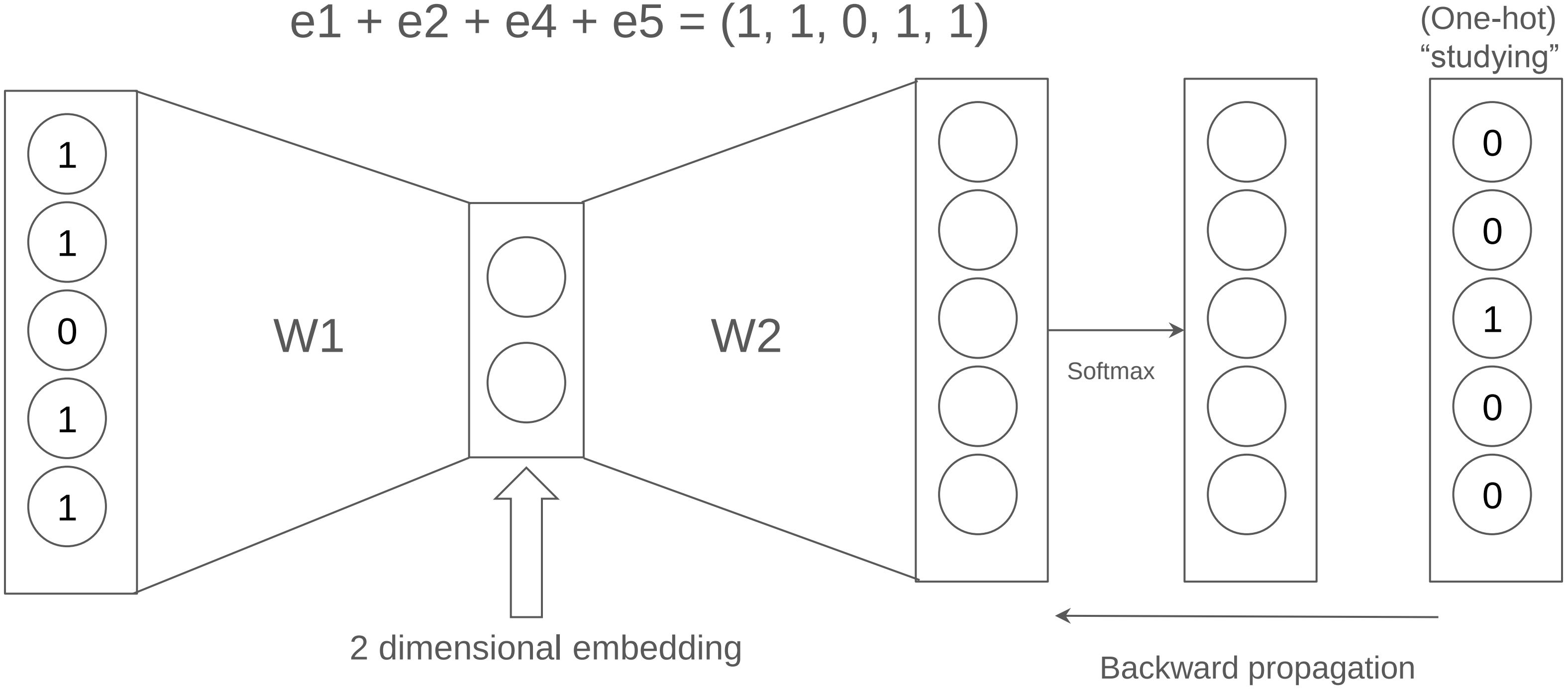
Word embeddings (Continuous Bag Of Words)

Loss function:
Cross-entropy loss

$$-\sum_{i=1}^{V} y_i \log p_i$$

Aggregate context vectors by sum/ mean:

e1 + e2 + e4 + e5 = (1, 1, 0, 1, 1)



Word embeddings (Skip-gram)

- Skip-gram

  Learns the embeddings by training a neural network with a single hidden layer to predict context words based on the middle word within a fixed window.

“Reverse of continuous bag of words”

For example: window size = 2,

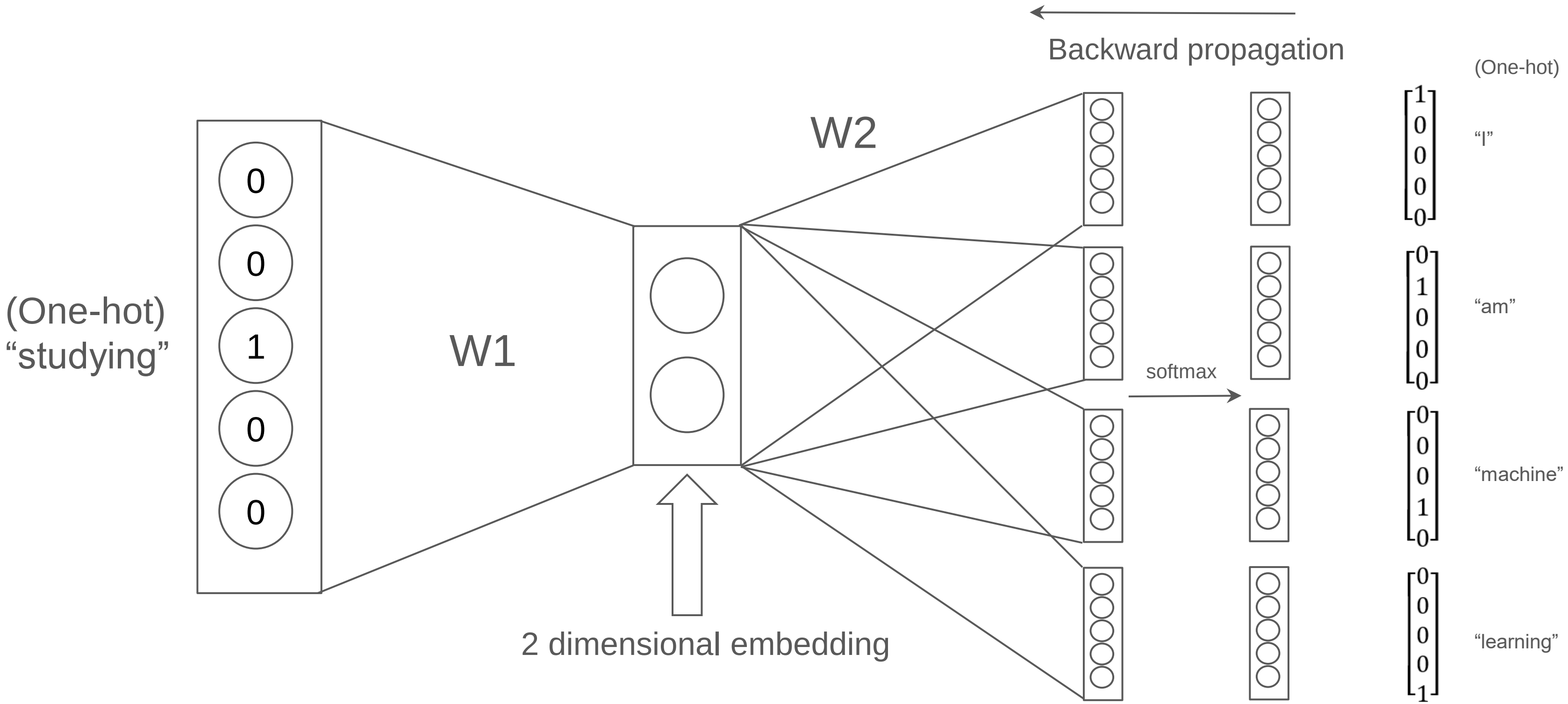
“ I am studying machine learning”

“I” : (1, 0, 0, 0, 0) = e1
“am” : (0, 1, 0, 0, 0) = e2
“studying”: (0, 0, 1, 0, 0) = e3
“machine”: (0, 0, 0, 1, 0) = e4
“learning”: (0, 0, 0, 0, 1) = e5

Word embeddings (Skip-gram)

Loss function:
Cross-entropy loss

$$-\sum_{i=1}^{V} y_i \log p_i$$



Word embeddings

- Text embeddings by weighted aggregation
  Instead of just simply concatenating the word embeddings in the text, we embed the text as follows:
  Let’s say all the tokens in the text are
  [“A”, “B”, “C”, “D”]
  and their word embeddings 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. Then we can form an embedding matrix $\begin{bmatrix}1 & 0 & 0 & 1\\0 & 1 & 0 & 1\\0 & 0 & 1 & 0\end{bmatrix}$

Word embeddings

- Text embeddings by weighted aggregation
  Embedding matrix

$$
\begin{array}{cccc}
\text{“A”} & \text{“B”} & \text{“C”} & \text{“D”} \\
\multicolumn{4}{c}{\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}}
\end{array}
$$

1, 2, 3, 4

For example, we have a text (after tokenization) like this [“D”, “A”, “B”, “D”].

Then, we can form ar ... r

“A” appears in 2nd position

“B” appears in 3rd position

“C” doesn’t appear

“D” appears in the
1st and 4th position

$$
\begin{bmatrix} 2 \\ 3 \\ 0 \\ 1 + 4 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 0 \\ 5 \end{bmatrix}
$$

Word embeddings

- Text embeddings by weighted aggregation

  Embedding matrix

  $$\begin{array}{cccc} \text{“A”} & \text{“B”} & \text{“C”} & \text{“D”} \end{array}$$
  $$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

  Encoding vector

  $$\begin{bmatrix} 2 \\ 3 \\ 0 \\ 1+4 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 0 \\ 5 \end{bmatrix}$$

Then, we can multiply the embedding matrix and the encoding vector to get the embedding for the text.

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

Word embeddings

- Performance on the **validation set** and the **testing set**

  Setup:
  - window size =10, embedding dimension = 100, weighted aggregation
  - 3 classification models
    - Logistic regression
    - Decision tree
    - MLP classifier
      - One hidden layer,
      - Learning rate =0.001
      - 100 neurons,
      - RELU activation function

Word embeddings

- Performance on the **validation set** and the **testing set**

Logistic Regression

	CBOW		Skip-gram	
	Validation set	Testing set	Validation set	Testing set
accuracy	0.96	0.67	0.96	0.68
precision	0.94	0.94	0.94	0.94
recall	0.99	0.44	0.99	0.46
F1-score	0.97	0.60	0.97	0.62

Word embeddings

- Performance on the **validation set** and the **testing set**

Decision Tree

	CBOW		Skip-gram	
	Validation set	Testing set	Validation set	Testing set
accuracy	0.93	0.98	0.93	0.70
precision	0.94	0.98	0.94	0.97
recall	0.94	0.46	0.94	0.48
F1-score	0.94	0.63	0.94	0.64

Word embeddings

- Performance on the **validation set** and the **testing set**

MLP

	CBOW		Skip-gram	
	Validation set	Testing set	Validation set	Testing set
accuracy	0.98	0.68	0.97	0.66
precision	0.98	1.00	0.98	1.00
recall	0.98	0.43	0.97	0.40
F1-score	0.98	0.60	0.98	0.58

Word embeddings

- Performance on the **validation set** and the **testing set**

In our use case, recall is the metric that we care more about.
“Among these fake news, how many of them can we distinguish?”

However, the recall values on testing set in our model is not ideal (< 0.5).

Word embeddings

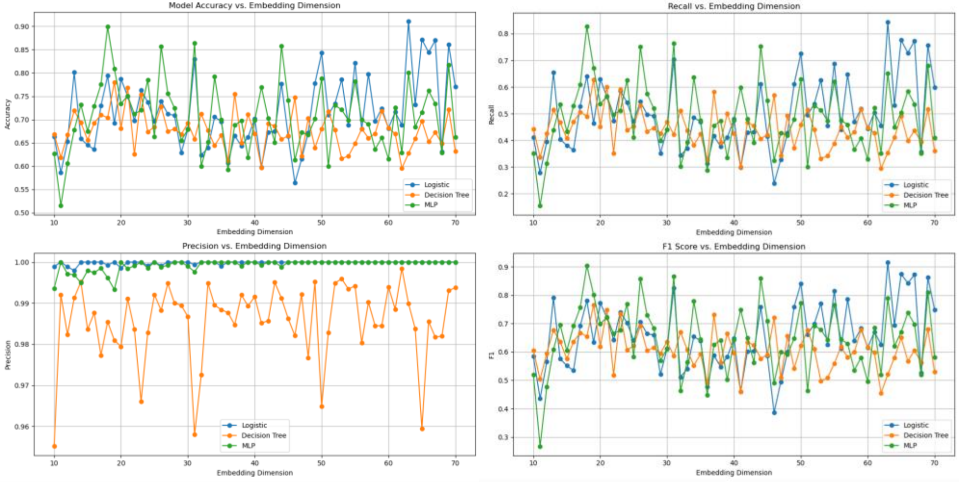
- Experiment: Dimensions of embeddings

  Setup:
  - window size =10
  - embedding dimension from 10 to 70
  - 3 classification models
    - Logistic regression
    - Decision tree
    - MLP classifier
      - One hidden layer, 100 neurons, RELU activation functions

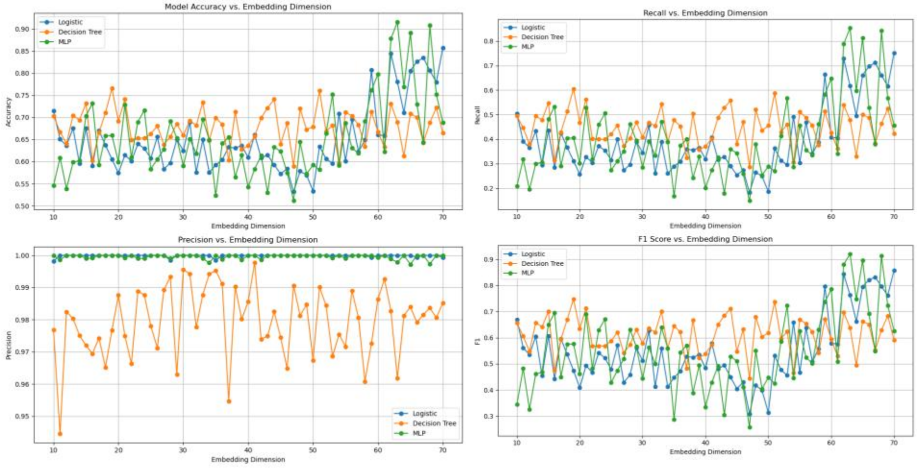
Word embeddings

Experiment: Dimensions of embeddings

CBOW



Skip-gram



Word embeddings (GloVe)

- Pre-trained word embeddings: [Global Vectors for Word Representation (GloVe)]
  - We are using `glove.6B.50d`
    - 6 billions tokens
    - Embedding dimension =50
  - Texts are embedded in the same method as before (weighted aggregation)

Word embeddings (GloVe)

- Performance on the **validation set** and the **testing set**

Logistic Regression

	Validation set	Testing set
accuracy	0.88	0.56
precision	0.85	0.56
recall	0.95	1.00
F1-score	0.90	0.72

Word embeddings (GloVe)

- Performance on the **validation set** and the **testing set**

Decision Tree

	Validation set	Testing set
accuracy	0.85	0.57
precision	0.86	0.59
recall	0.87	0.81
F1-score	0.87	0.68

Word embeddings (GloVe)

- Performance on the **validation set** and the **testing set**

MLP

	Validation set	Testing set
accuracy	0.92	0.57
precision	0.92	0.57
recall	0.93	0.98
F1-score	0.92	0.72

Doc2vec

- An extension of CBOW/ Skip-gram
  Instead of embedding each single word, Doc2Vec embeds the whole document (text) into vectors.

- For each document, an unique document identifier (vector) is assigned to it. For each word embeddings and the document identifier, the vectors are randomly initialised before the model training.

Tokenized documents
Doc1: [“dogs”, “are”, “great”, “pet”]
Doc2: [“the”, “cat”, “sits”, “on”, “the”, “mat”]

Document identifiers
Doc1: D1 [0.1, 0.2]
Doc2: D2 [0.3, 0.4]

Initial word embeddings
“dogs”: [0.0, 0.1] “are” :[0.2, 0.3]
“great”: [0.4, 0.5] “pet” :[0.5, 0.5]
“the”: [0.7, 0.2] “sits”

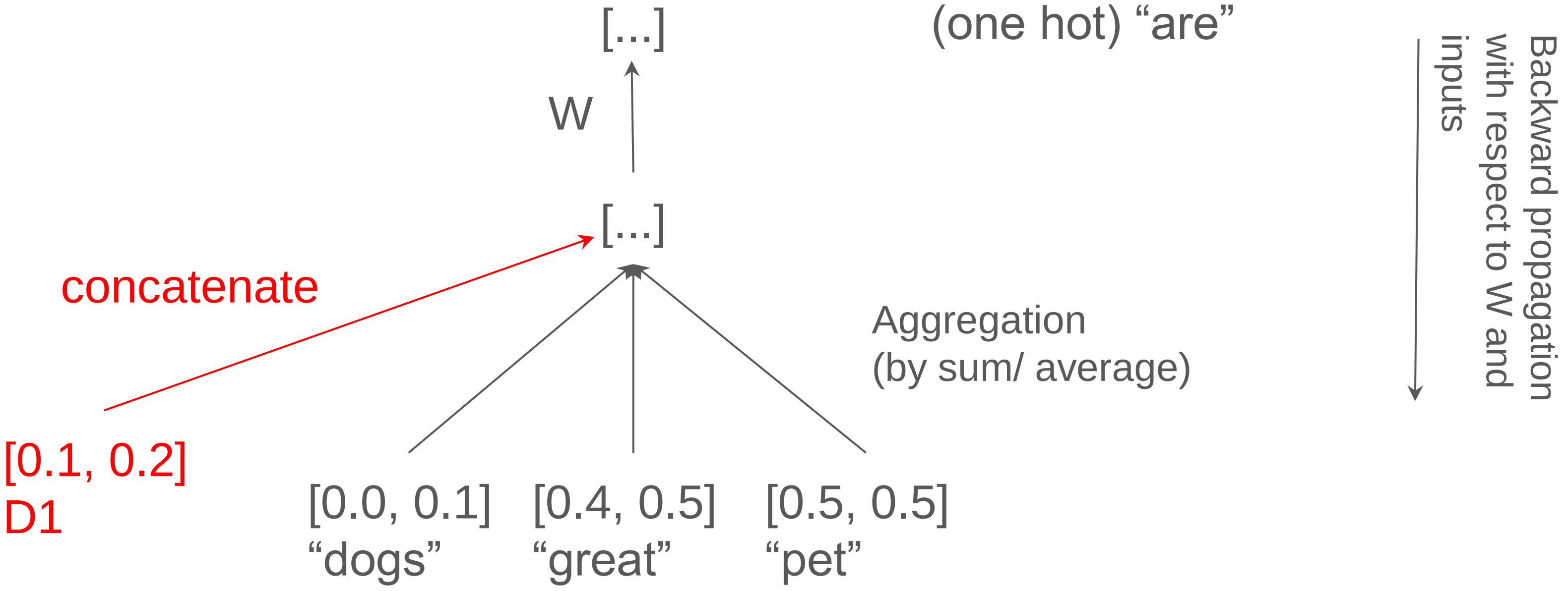
Doc2vec

Initial word embeddings
"dogs": [0.0, 0.1] "are" :[0.2, 0.3]
"great": [0.4, 0.5] "pet" :[0.5, 0.5]
"the": [0.7, 0.2] "sits" :[0.3, 0.5]
"on": [0.8, 0.6] "mat" :[0.8, 0.2]

Tokenized documents
Doc1: ["dogs", "are", "great", "pet"]
Doc2: ["the", "cat", "sits", "on", "the", "mat"]

Document identifiers
Doc1: D1 [0.1, 0.2]
Doc2: D2 [0.3, 0.4]

- Distributed Memory (DM)
  Similar to CBOW, use context words to predict the target word.
  In addition to the context words, the document identifiers are also used.



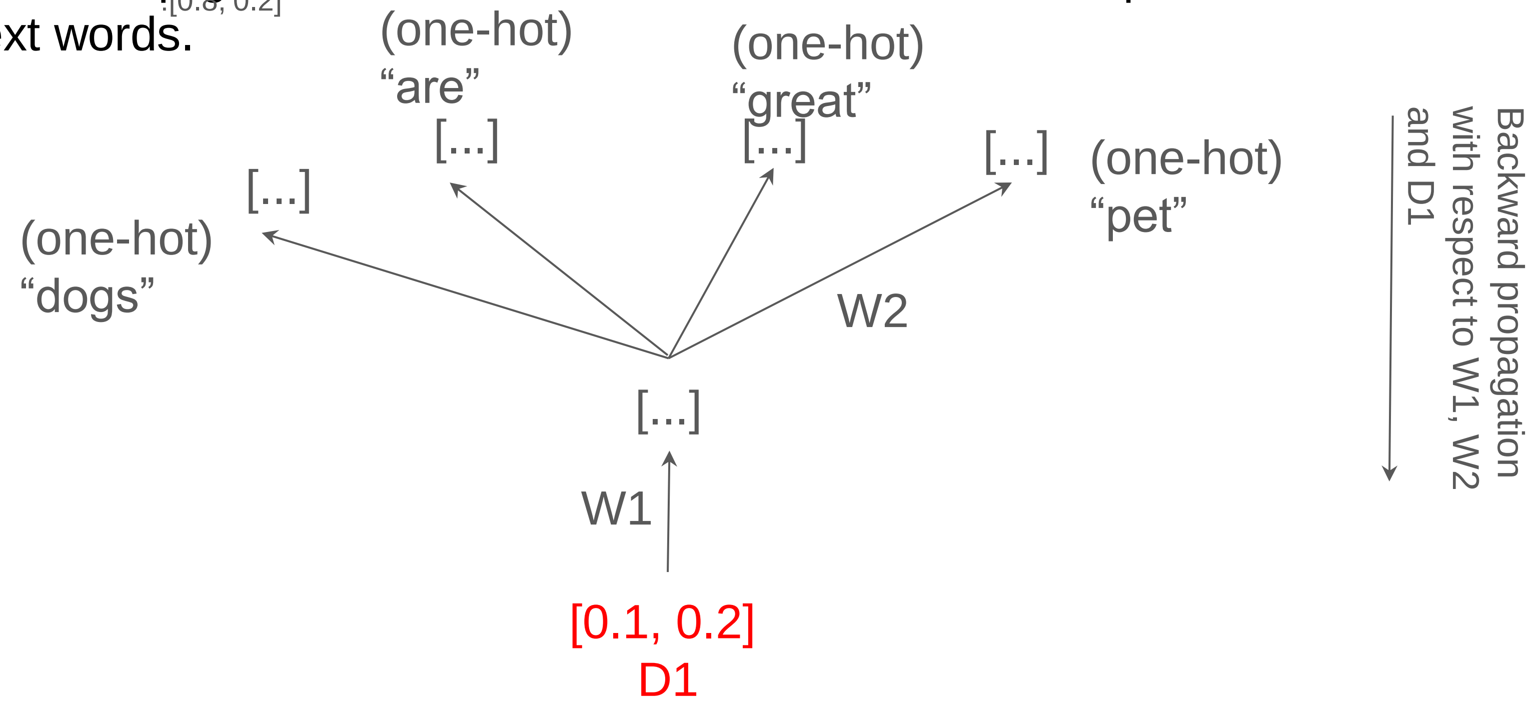
Doc2vec

Initial word embeddings
"dogs": [0.0, 0.1] "are" :[0.2, 0.3]
"great": [0.4, 0.5] "pet" :[0.5, 0.5]
"the": [0.7, 0.2] "sits" :[0.3, 0.5]
"on": [0.8, 0.6] "mat" :[0.8, 0.2]

Tokenized documents
Doc1: ["dogs", "are", "great", "pet"]
Doc2: ["the", "cat", "sits", "on", "the", "mat"]

Document identifiers
Doc1: D1 [0.1, 0.2]
Doc2: D2 [0.3, 0.4]

- Distributed Bag Of Words (DBOW)
  Similar to skip-gram, but it uses the document identifier to predict the context words.



Doc2Vec

- Performance on the **validation set** and the **testing set**

Logistic Regression

	DM		DBOW	
	Validation set	Testing set	Validation set	Testing set
accuracy	0.94	0.94	0.99	1.00
precision	0.93	0.93	0.99	1.00
recall	0.96	0.96	1.00	1.00
F1-score	0.94	0.95	1.00	1.00

Doc2Vec

- Performance on the **validation set** and the **testing set**

Decision Tree

	DM		DBOW	
	Validation set	Testing set	Validation set	Testing set
accuracy	0.85	0.86	0.93	0.94
precision	0.86	0.88	0.93	0.95
recall	0.87	0.87	0.94	0.95
F1-score	0.87	0.87	0.94	0.95

Doc2Vec

- Performance on the **validation set** and the **testing set**

MLP

	DM		DBOW	
	Validation set	Testing set	Validation set	Testing set
accuracy	0.95	0.95	0.99	0.99
precision	0.95	0.97	1.00	1.00
recall	0.95	0.95	0.99	0.99
F1-score	0.95	0.96	1.00	1.00

Term Frequency-Inverse Document Frequency (TF-IDF)

- Similar to BOW, TF-IDF is a word embedding method
- It calculates the importance of words to documents in a collection of corpus.
  - The higher frequency the word appear in a text, the higher meaning assigned to that word
  - But is compensated by total word frequency in the corpus (whole dataset).
    - Solved the major drawback for Bag Of Words

Term Frequency-Inverse Document Frequency (TF-IDF)

- Product of two terms
  - Normalised Term Frequency
  - Inverse Document Frequency

Term Frequency (TF)	Inverse Document Frequency (IDF)
In a document d, the frequency represents the number of instances of a given word t. Normalization will also be performed to ensure fairness in results.	Number of documents containing word t divided by # of documents is the document frequency (DF) of t, and we want to take the inverse and log of it to test the relevance of t.
TF(d,t) = count of t in d / number of words in d	IDF(d,t) = log(# of d / DF(t))

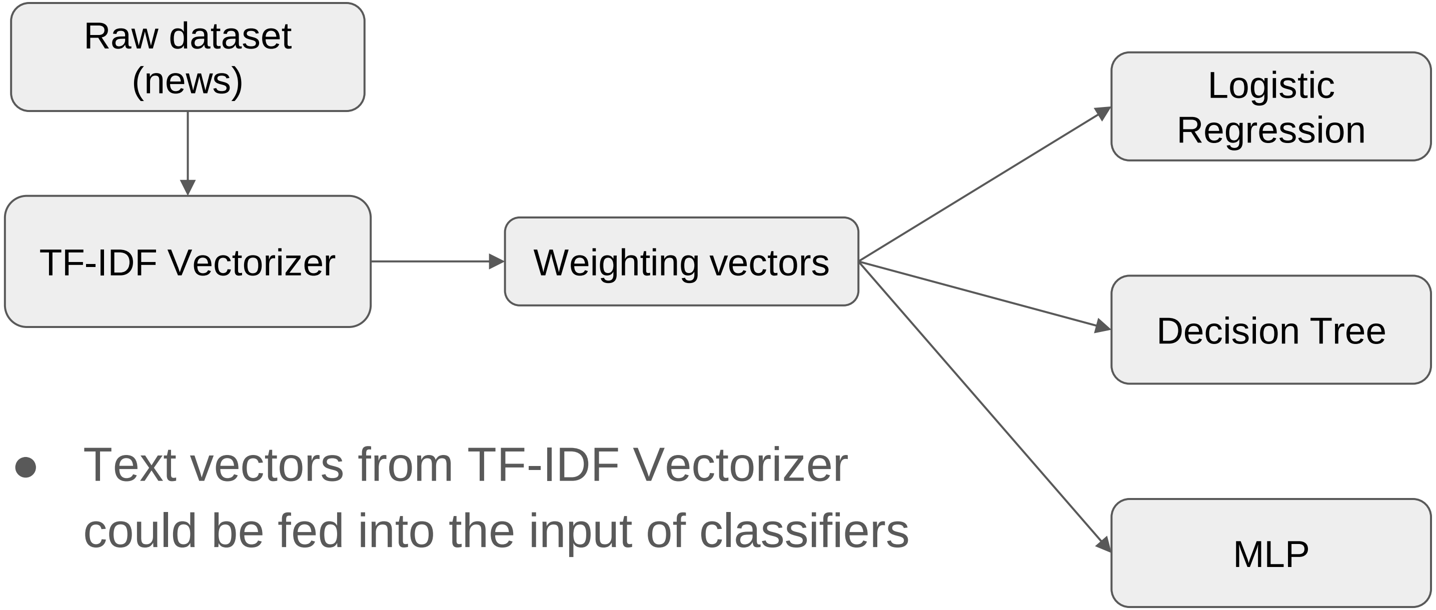
- The words with higher scores of weight are deemed to be more significant.

Term Frequency-Inverse Document Frequency (TF-IDF)

- Example texts
    - Today is a sunny day!
    - He is handsome.
    - She is going to play today.

Word	TF (Before normalization)	IDF
is	$\frac{1}{5}$	0 (useless)
today	$\frac{1}{3}$	0.2385
sunny	$\frac{1}{6}$	0.4771

Term Frequency-Inverse Document Frequency (TF-IDF)



- Text vectors from TF-IDF Vectorizer could be fed into the input of classifiers

Term Frequency-Inverse Document Frequency (TF-IDF)

- Performance on the **validation set** and the **testing set**

Logistic Regression

	Validation set	Testing set
accuracy	0.9836	0.9816
precision	0.98	0.98
recall	0.98	0.98
F1-score	0.98	0.98

True Positive	2087
True Negative	1707
False Positive	49
False Negative	22

Term Frequency-Inverse Document Frequency (TF-IDF)

- Performance on the **validation set** and the **testing set**

Decision Tree

	Validation set	Testing set
accuracy	0.9939	0.9943
precision	0.99	0.99
recall	0.99	0.99
F1-score	0.99	0.99

True Positive	2100
True Negative	1743
False Positive	13
False Negative	9

Term Frequency-Inverse Document Frequency (TF-IDF)

- Performance on the **validation set** and the **testing set**

MLP

	Validation set	Testing set
accuracy	0.9832	0.9816
precision	0.98	0.98
recall	0.98	0.98
F1-score	0.98	0.98

True Positive	2088
True Negative	1706
False Positive	50
False Negative	21

Term Frequency-Inverse Document Frequency (TF-IDF)

- Among three classifiers, decision tree has a slightly better edge
  - TF-IDF creates a sparse representation of the text inputs
  - Simpler feature space
  - More interpretable and less complex
    - Effective splits for decision tree
- Overall better than previous traditional methods
  - GloVe
  - Doc2Vec
  - CBOW/Skip-Gram

Decision Tree

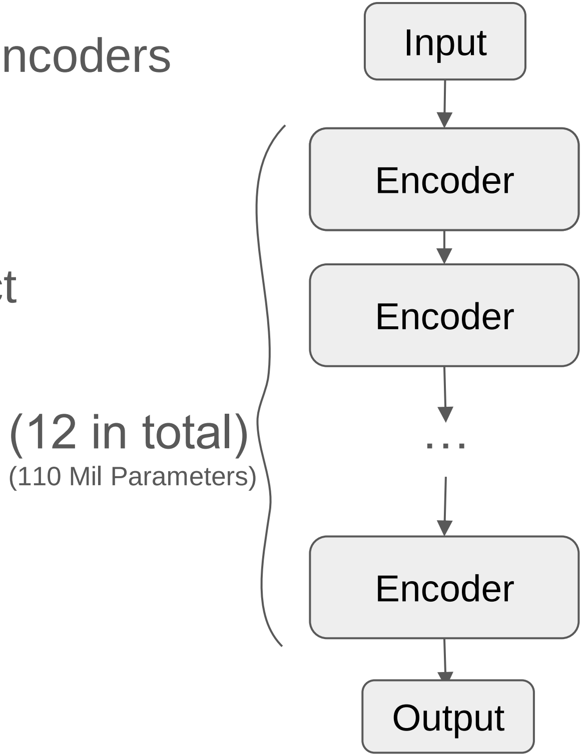
	Validation set	Testing set
accuracy	0.9939	0.9943
precision	0.99	0.99
recall	0.99	0.99
F1-score	0.99	0.99

True Positive	2100
True Negative	1743
False Positive	13
False Negative	9

Pre-trained Transformers (BERT)

- A type of transformer that only consists of encoders
- Only requires unlabelled data to train on
- Self-supervised training

We will use the base BERT model for our project

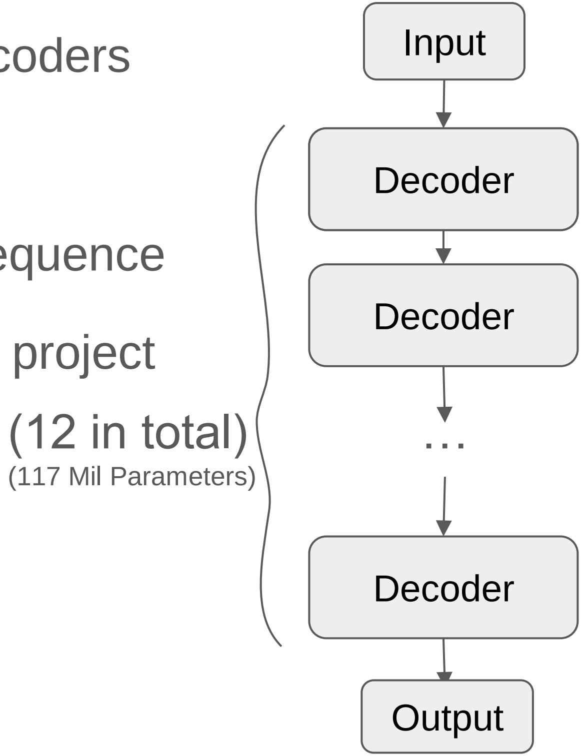


Pre-trained Transformers (GPT)

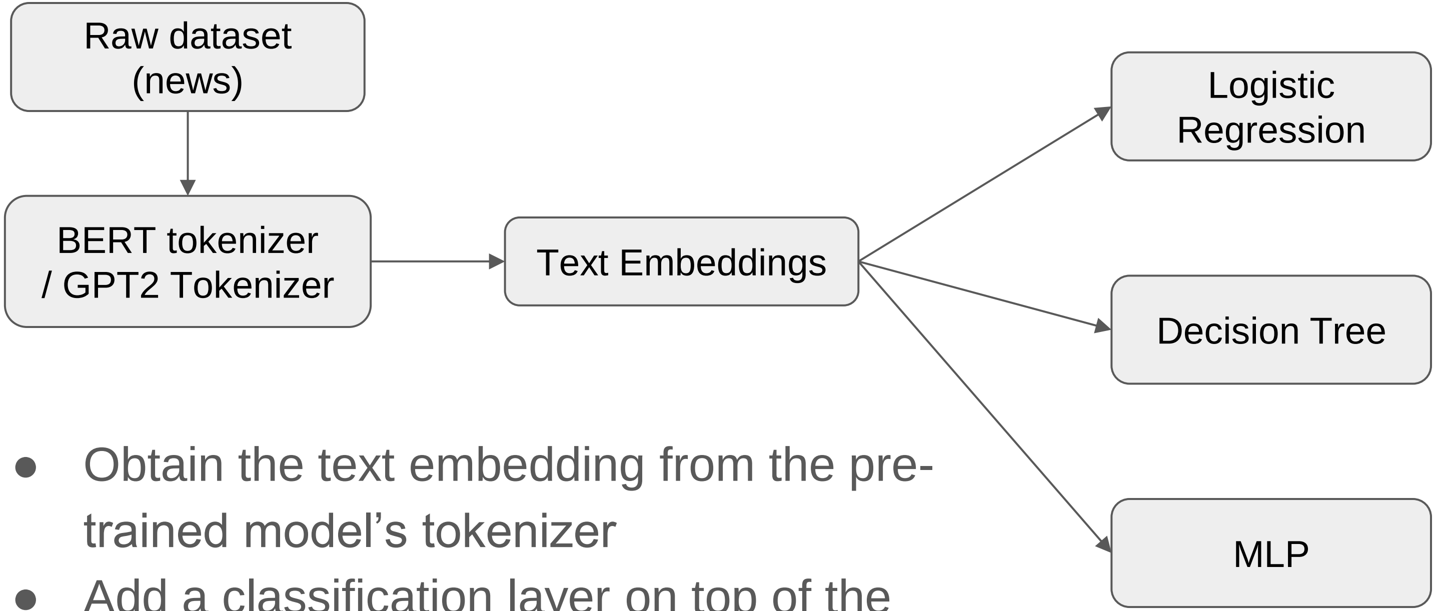
- Contrast to BERT, GPT only consists of decoders
- Only requires unlabelled data to train on
- Self-supervised training
- Powerful at predicting the next token in a sequence

We will import and use the GPT2* model for our project

(* OpenAI GPT-2 English model)



Pre-trained Transformers (BERT / GPT)



- Obtain the text embedding from the pre-trained model's tokenizer
- Add a classification layer on top of the pre-trained model

Pre-trained Transformers (BERT)

- Performance on the **validation set** and the **testing set**

Logistic Regression

	Validation set	Testing set
accuracy	0.9925	0.9920
precision	0.99	0.99
recall	0.99	0.99
F1-score	0.99	0.99

True Positive	2087
True Negative	1747
False Positive	15
False Negative	16

Pre-trained Transformers (BERT)

- Performance on the **validation set** and the **testing set**

Decision Tree

	Validation set	Testing set
accuracy	0.9178	0.9200
precision	0.92	0.92
recall	0.92	0.92
F1-score	0.92	0.92

True Positive	1942
True Negative	1614
False Positive	148
False Negative	161

Pre-trained Transformers (BERT)

- Performance on the **validation set** and the **testing set**

MLP

	Validation set	Testing set
accuracy	0.9943	0.9959
precision	0.99	0.99
recall	0.99	0.99
F1-score	0.99	0.99

True Positive	2090
True Negative	1759
False Positive	3
False Negative	13

Pre-trained Transformers (GPT)

- Performance on the **validation set** and the **testing set**

Logistic Regression

	Validation set	Testing set
accuracy	0.9995	0.9995
precision	1.00	1.00
recall	1.00	1.00
F1-score	1.00	1.00

True Positive	2101
True Negative	1762
False Positive	0
False Negative	2

Pre-trained Transformers (GPT)

- Performance on the **validation set** and the **testing set**

Decision Tree

	Validation set	Testing set
accuracy	0.9687	0.9588
precision	0.97	0.96
recall	0.97	0.96
F1-score	0.97	0.96

True Positive	2016
True Negative	1690
False Positive	72
False Negative	87

Pre-trained Transformers (GPT)

- Performance on the **validation set** and the **testing set**

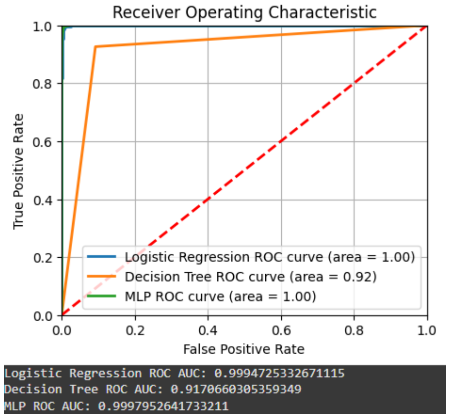
MLP

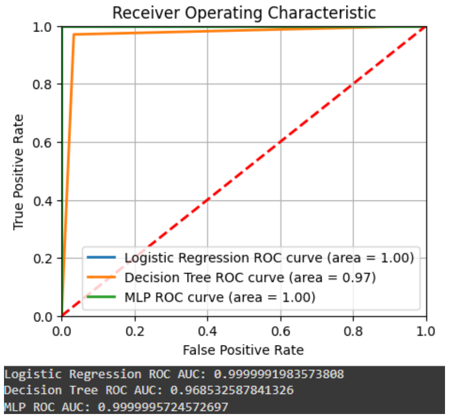
	Validation set	Testing set
accuracy	0.9995	0.9997
precision	1.00	1.00
recall	1.00	1.00
F1-score	1.00	1.00

True Positive	2102
True Negative	1762
False Positive	0
False Negative	1

Pre-trained Transformers (BERT / GPT)

- Unlike TF-IDF -> BERT/GPT both have significantly lower accuracy on decision tree compared to other classifiers
- Reasons possibly due to:
  - Tokenized text embeddings are high-dimensional
  - Contextually rich
  - High complexity -> overfitting (captured noise rather than generalization)





Pre-trained Transformers (BERT / GPT)

- Reduce dimensionality and complexity of the embeddings
  - Principal Components Analysis (PCA)
- Use random forest instead of decision tree
  - Prevent overfitting
  - Better suited for high dimensionality

Original BERT + decision tree accuracy/recall: 0.92

PCA with n_componets = 30 (BERT)

```
Accuracy:
0.9457408897574434


Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.94      0.94      3916
           1       0.95      0.95      0.95      4783

    accuracy                           0.95      8699
   macro avg       0.95      0.95      0.95      8699
weighted avg       0.95      0.95      0.95      8699
```

	Metric	Count
0	True Positives (TP)	4555
1	True Negatives (TN)	3672
2	False Positives (FP)	244
3	False Negatives (FN)	228

Random Forest using PCA data (BERT)

```
Accuracy:
0.9660880560984021


Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.95      0.96      3916
           1       0.96      0.98      0.97      4783

    accuracy                           0.97      8699
   macro avg       0.97      0.96      0.97      8699
weighted avg       0.97      0.97      0.97      8699
```

	Metric	Count
0	True Positives (TP)	4681
1	True Negatives (TN)	3723
2	False Positives (FP)	193
3	False Negatives (FN)	102

Further possible exploration with our results

- Domain-specific contexts : Not limited to political news -> medical news?
- Multilingual Text Classification : Different languages?

- Discover the potentials of ensemble methods -> combining the use of pre-trained model with our word-embedding methods for higher accuracy