Model Comparison on Fake News Detection

Group 3: 余振揚、莊上緣、鄭九彰、詹雅淇

Outline

01

Introduction

02

Dataset

03

Model Pipelines

04

Model Details

05

Results

01

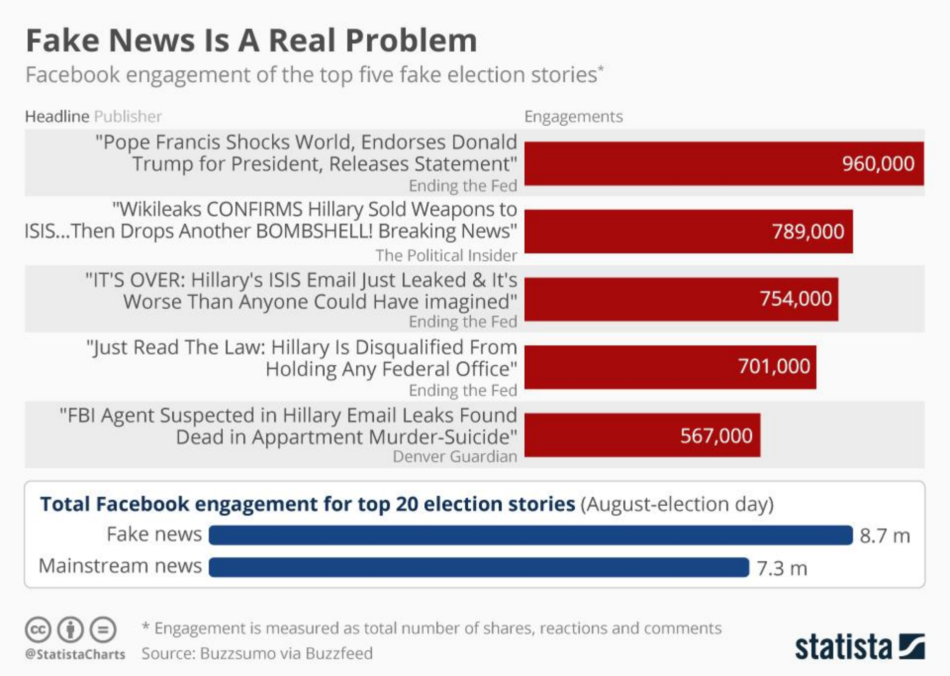
Introduction

Introduction

- **False news spread faster**:
six times faster than truthful content
(Vosoughi et al. 2018)

- **Hard to distinguish**:
70% of the users could not distinguish real from fake news (Vosoughi et al. 2018)

- **The spreading of false political news is more effective** than for false news about terrorism, natural disasters, science, urban legends, or financial information (Vosoughi et al. 2018).



Vosoughi S, Roy D, Aral S. The spread of true and false news online. Science. 2018;359(6380):1146–1151. doi: 10.1126/science.aap9559.

02

Dataset

Fake News Detection Challenge KDD 2020

資料來源

這個 fake news detection 資料來自 Kaggle 與 SIGKDD 2020 共同舉辦的活動：

Second International TrueFact Workshop: Making a Credible Web for Tomorrow

在這個活動中，參賽者需要設計一個系統來區分 claim(聲明)的真偽性。

資料連結

🔗Fake News Detection Challenge KDD 2020

Fake News Detection Challenge KDD 2020

Evaluation

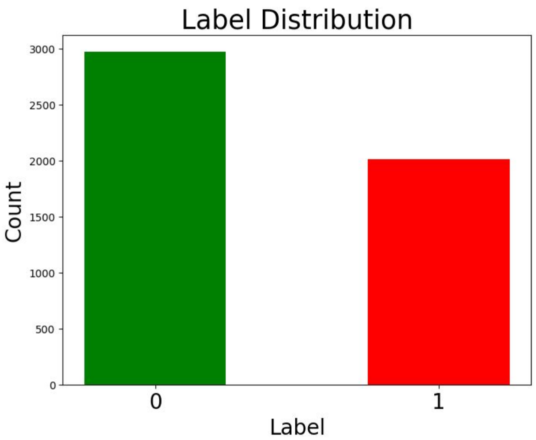
使用以下的 accuracy 公式來衡量模型準確性

$$accuracy = \frac{correct\ predictions}{correct\ predictions + incorrect\ predictions}$$

Fake News Detection Challenge KDD 2020

train.csv → (80% training set, 20% validation set)

- text : text of the article
- label : a label that marks the article as potentially unreliable
  - 1: fake / 0: true




```
                                                   text  label
0     Get the latest from TODAY Sign up for our news...      1
1     2d  Conan On The Funeral Trump Will Be Invited...      1
2     It’s safe to say that Instagram Stories has fa...      0
3     Much like a certain Amazon goddess with a lass...      0
4     At a time when the perfect outfit is just one ...      0
...                                                 ...    ...
4982  The storybook romance of WWE stars John Cena a...      0
4983  The actor told friends he’s responsible for en...      0
4984  Sarah Hyland is getting real.  The Modern Fami...      0
4985  Production has been suspended on the sixth and...      0
4986  A jury ruled against Bill Cosby in his sexual ...      0

[4987 rows x 2 columns]
```

03

Model Pipelines

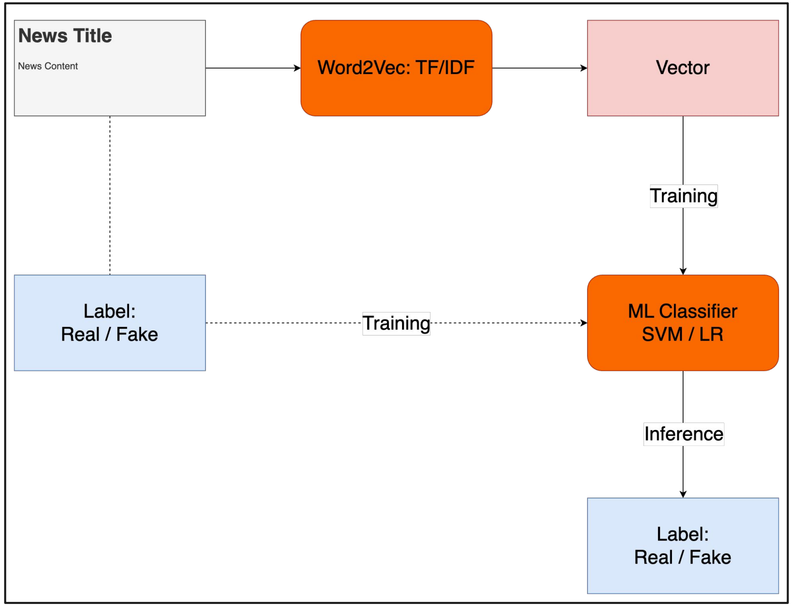
Model Pipelines

我們在此 dataset 上實作不同的 pipeline

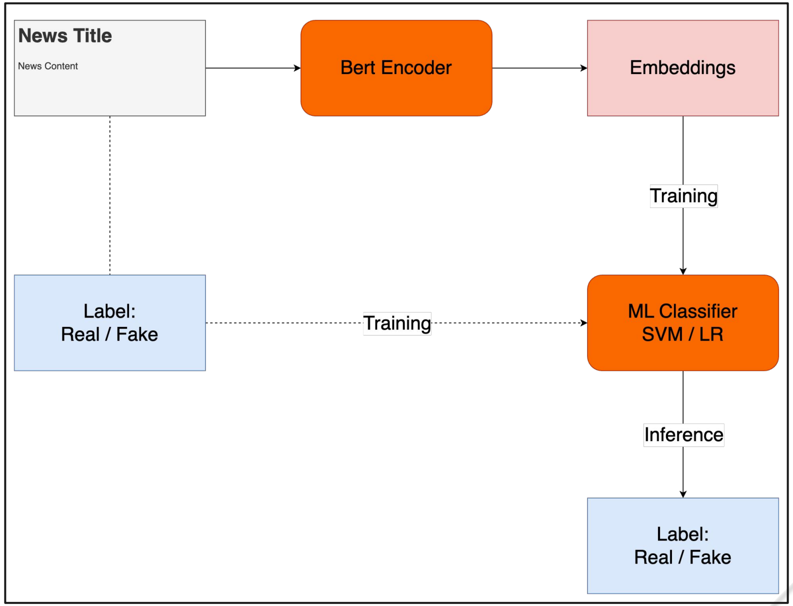
1.	Content →	TF-IDF → Vector →	ML Classifier(SVM/Logistic Regression)
2.	Content →	Encoder(BERT) → Embedding →	ML Classifier(SVM/Logistic Regression)
3.	Content →	Encoder(BERT) → Embedding →	Encoder(Fintuned) + Classifier
4.	Content →	TF-IDF+PMI → Vector →	Graph Construction → GCN Classifier
5.	Content →	Encoder(BERT) → Embedding →	Graph Construction → GCN Classifier

Model Pipelines

1. TF-IDF + ML

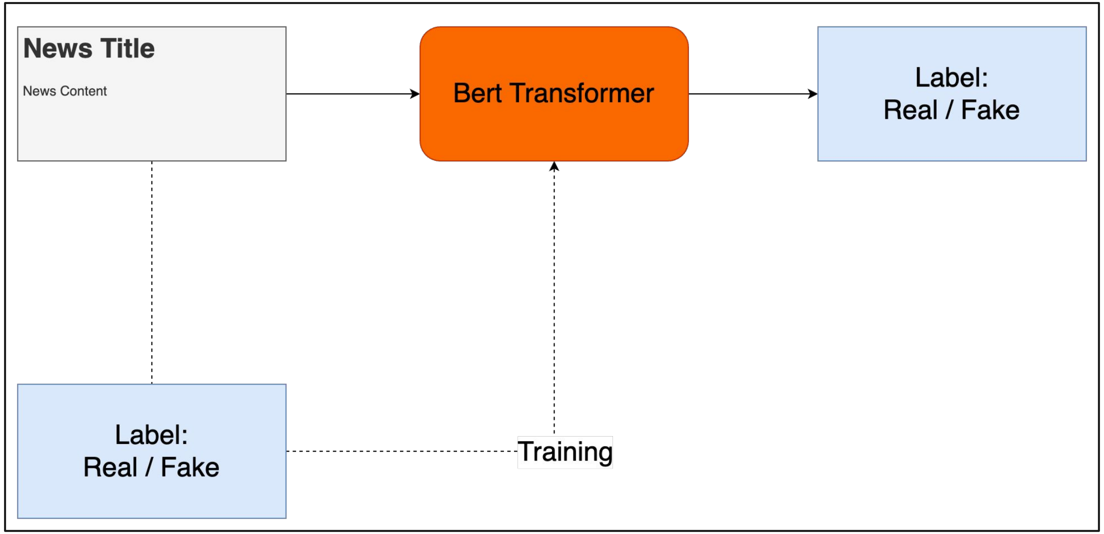


2. Embedding + ML



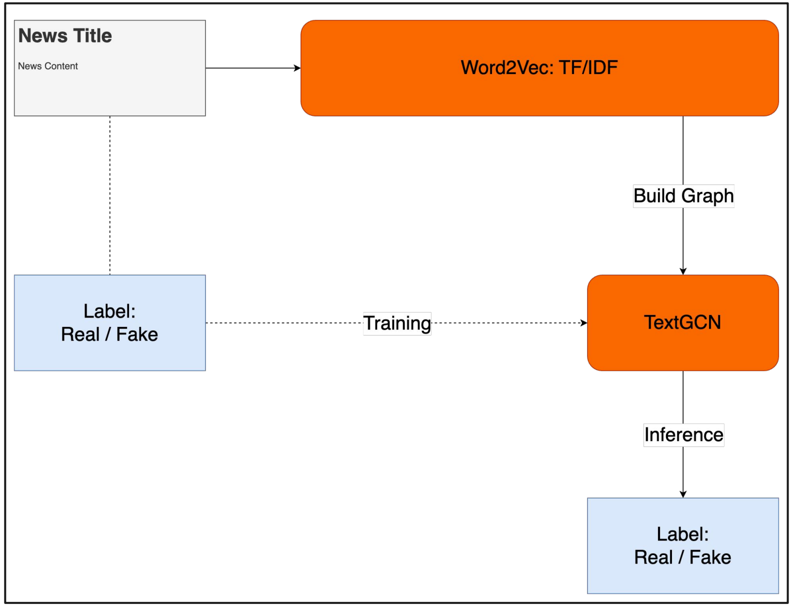
Model Pipelines

3. Encoder(Fintuned) + MLP Classifier

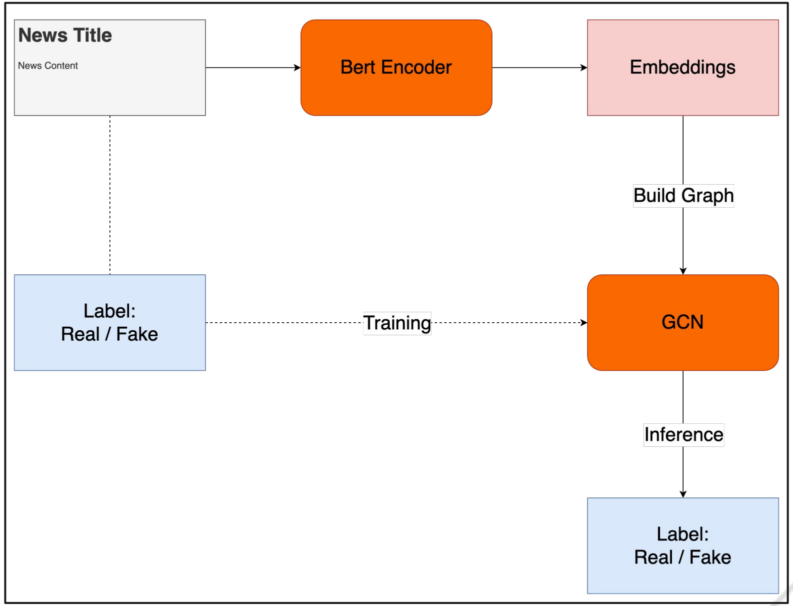


Model Pipelines

4. TF-IDF + TextGCN



5. Embedding + GCN



Model Pipeline

1. Content → Word2Vec(TF-IDF) → Vector → ML Classifier
2. Content → Encoder(Bert → Embedding → ML Classifier
3. Content → Encoder(Bert) → Embedding → Transformer Classifier
4. Content → Word2Vec(TF-IDF) → Vector → TextGCN
5. Content → Encoder(Bert) → Embedding → GCN Classifier

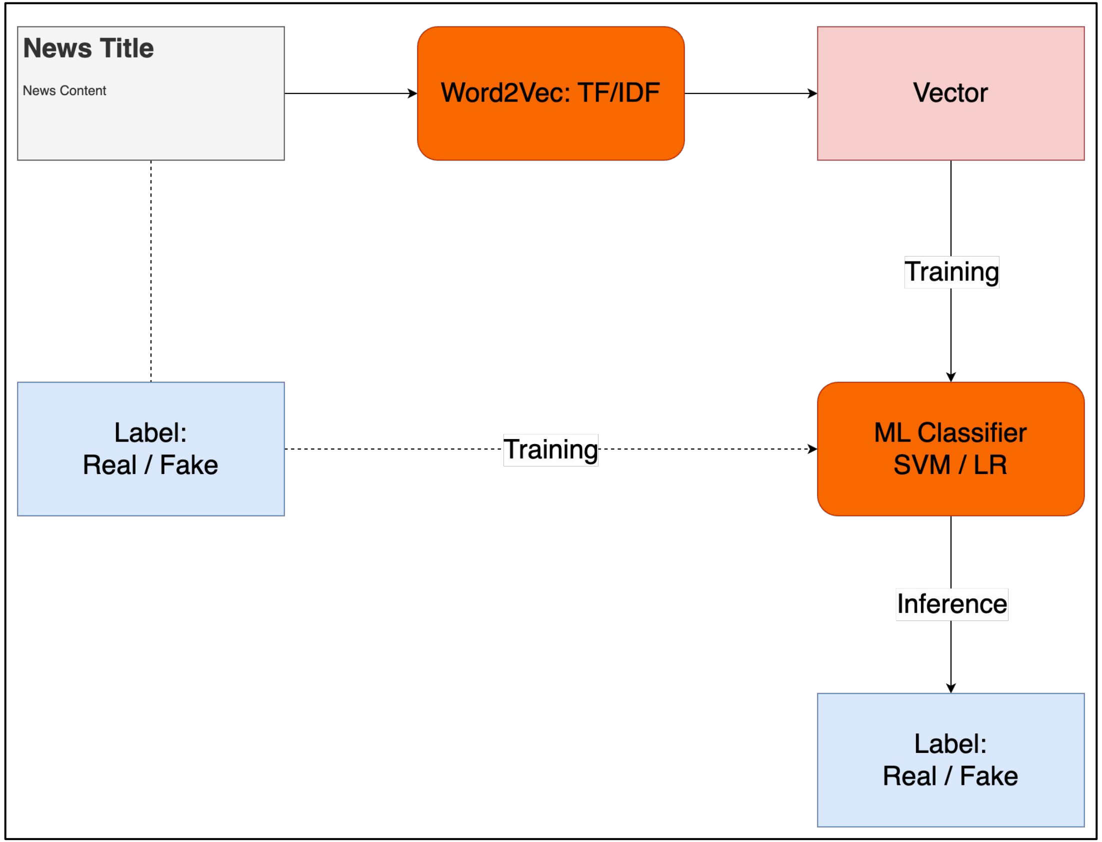
* ML Classifier: SVM, Logistic Regression

* Transformer: Bert based model as our baseline.

04

Model Details

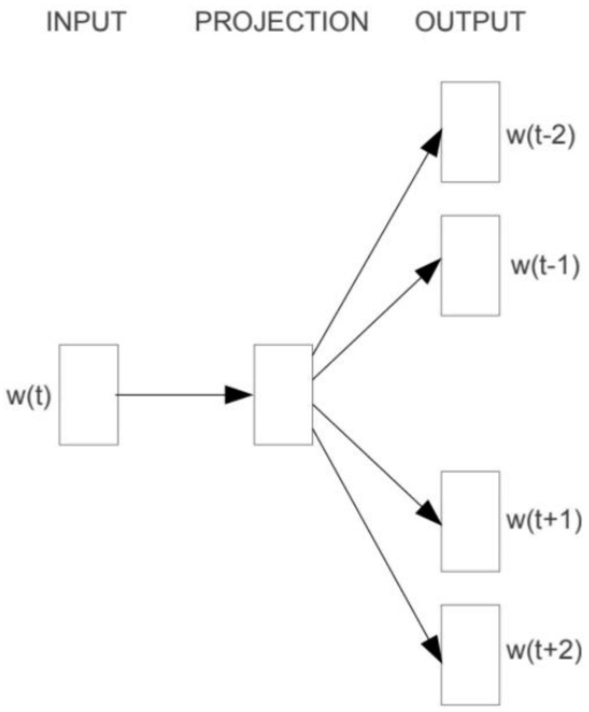
1. TF-IDF + ML Classifier



1. TF-IDF + ML Classifier

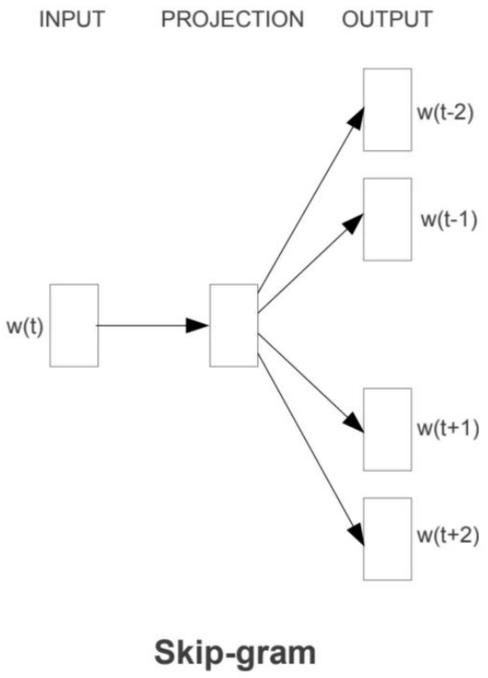
Word2Vec Model

- Skip-gram:
  - inputs -> a word
  - outputs -> a vector
  - hyperparameters:
    - vector_size = 100
    - window = 5



Skip-gram

1. TF-IDF + ML Classifier



Classification Results

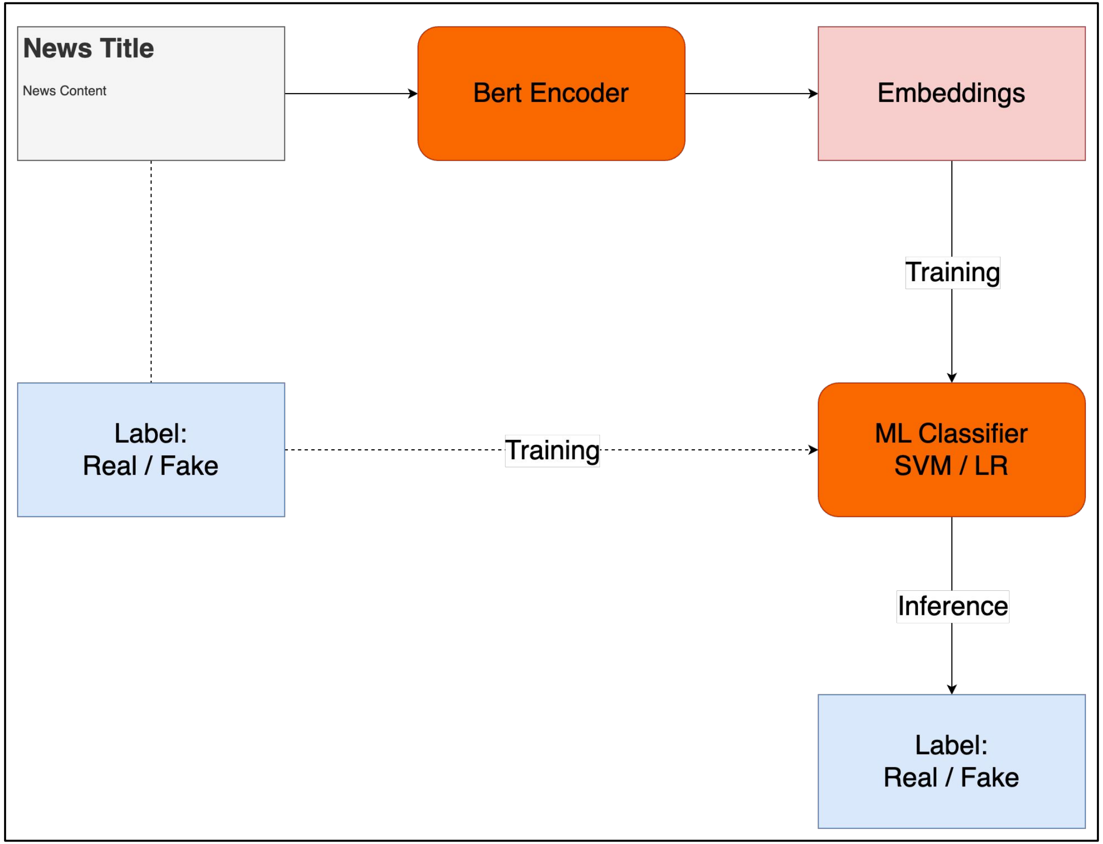
SVM :

test_acc = **72%**

Logistic Regression :

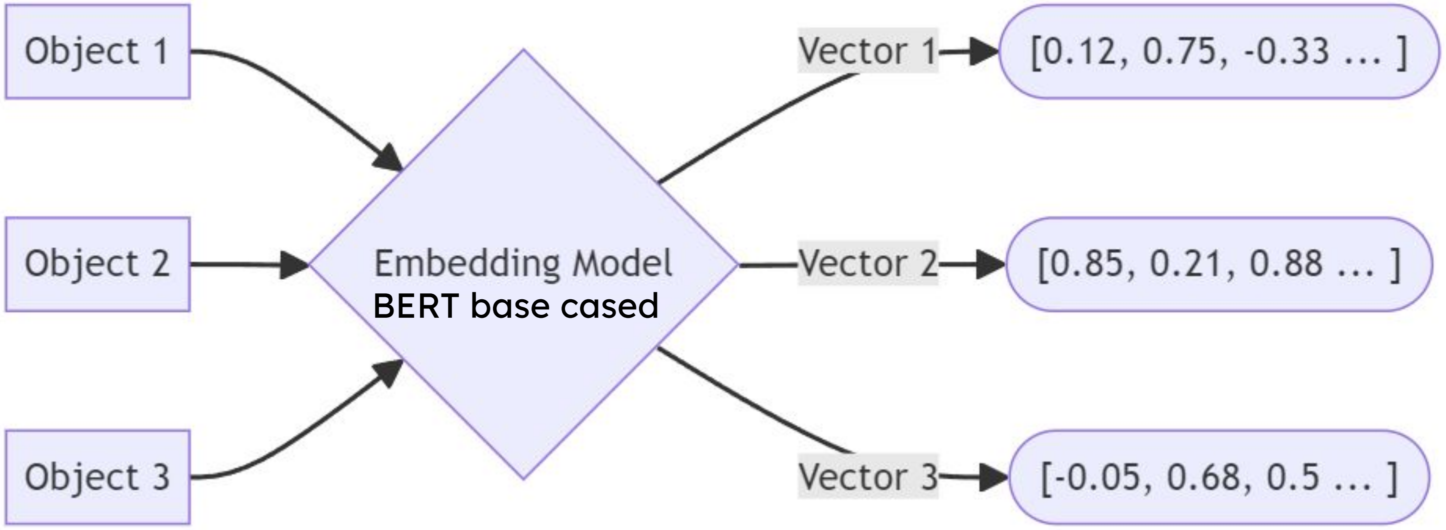
test_acc = **73%**

2. Embedding + ML Classifier



2. Embedding + ML Classifier

Embedding



2. Embedding + ML Classifier

Embedding

[0.12, 0.75, -0.33 ...]

[0.85, 0.21, 0.88 ...]

[-0.05, 0.68, 0.5 ...]

Classification Results

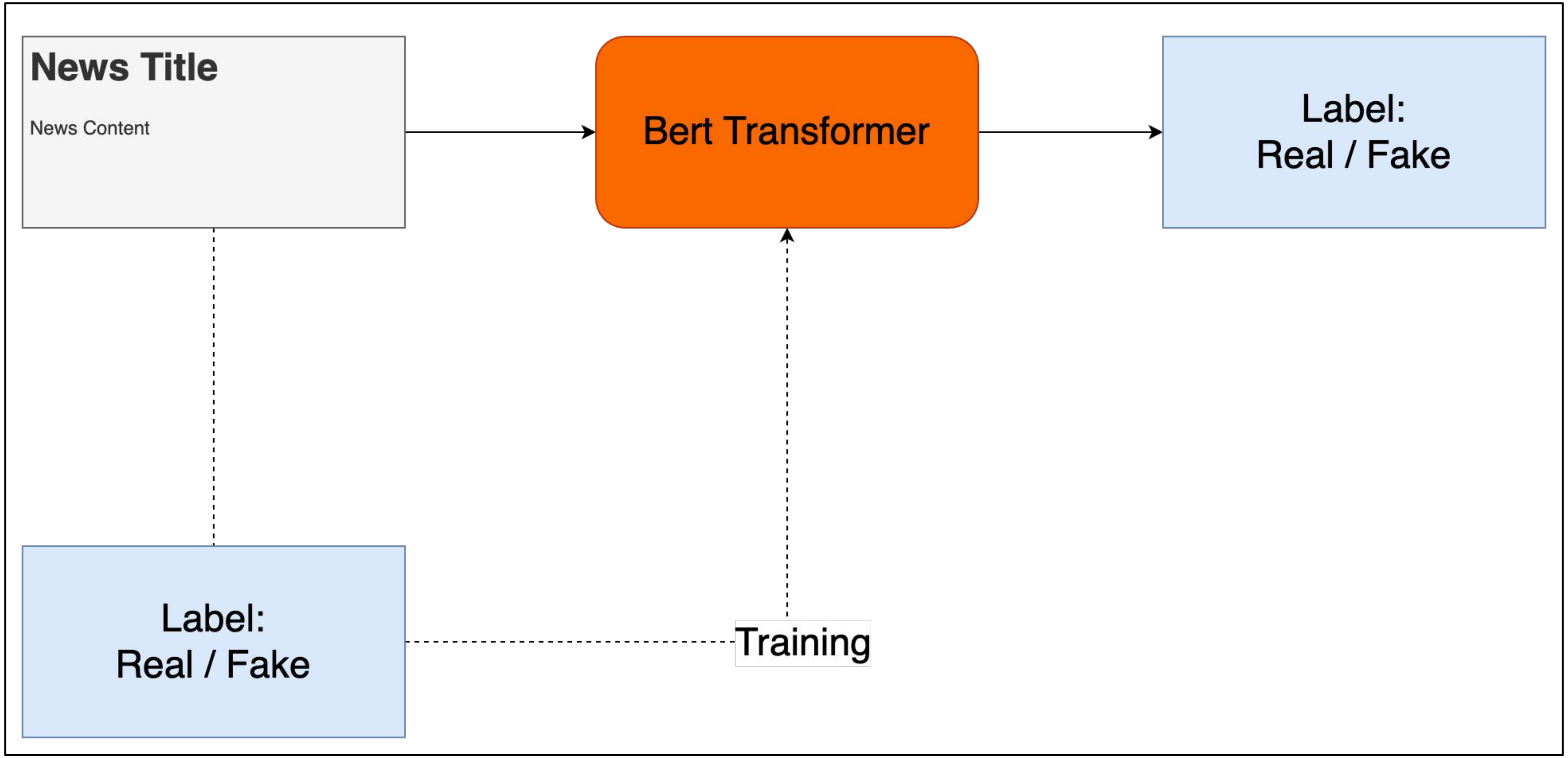
SVM :

test_acc = **73%**

Logistic Regression :

test_acc = **69%**

3. Embedding + Encoder(Finetuned) + MLP Classifer

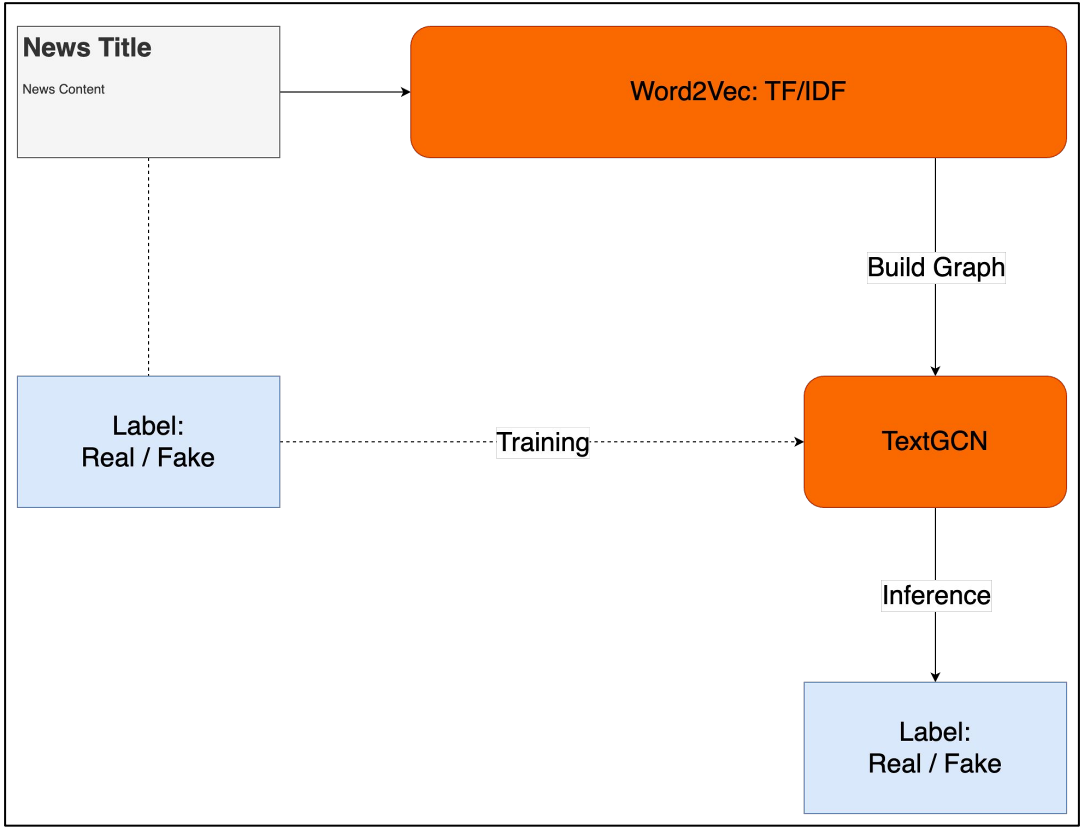


3. Embedding + Encoder(Finetuned) + MLP Classifer

- Pretrained-Tokenizer: distilbert/distilbert-base-uncased

- Training Epoch: 20 (5 is enough)
- Validation Accuracy: 0.81

4. TF-IDF+PMI+GCN Classifier



4. TF-IDF+PMI+GCN Classifier

會建立一個包含**document node**與**word node**的**異質圖**，Document-word edge的建邊依據是透過**TF-IDF**，而word-word edge的建邊依據則是透過**PMI**

$$A_{ij} = \begin{cases} \mathrm{PMI}(i,j) & i, j \text{ are words, } \mathrm{PMI}(i,j) > 0 \\ \text{TF-IDF}_{ij} & i \text{ is document, } j \text{ is word} \\ 1 & i = j \\ 0 & \text{otherwise} \end{cases}$$

The PMI value of a word pair i, j is computed as

$$\mathrm{PMI}(i,j) = \log \frac{p(i,j)}{p(i)p(j)}$$

$$p(i,j) = \frac{\#W(i,j)}{\#W}$$

$$p(i) = \frac{\#W(i)}{\#W}$$

4. TF-IDF+PMI+GCN Classifier

以TF-IDF來當作graph construction的document node與word node之間的建邊依據:

一句話解釋TF-IDF:**用來從一段文字 /一個語料庫中, 給越重要的字詞 /文檔, 越高的加權分數** 。

字詞的重要性隨著 在文本出現的頻率越高則越高;在不同文本檔案間出現的次數越高則反而降低 。

$$Score_{t,d} = tf_{t,d} \times idf_t$$

TF

(Term Frequency)

每個詞在每個文件出現的比率

IDF

(Inverse Document Frequency)

詞在所有文件的頻率

頻率越高表該詞越不具代表性，IDF值越小

4. TF-IDF+PMI+GCN Classifier

以PMI(Pointwise mutual information)來當作graph construction的word node與word node之間的建邊依據:

由於文本長度太長, 因此我們透過建立一個大小固定的滑動窗口, 計算兩兩word組成的word-pair出現在同一窗口的次數

```
# word co-occurence with context windows
# store words in the same window
window_size = 15
windows = []
for doc_words in doc_content_list:
    words = doc_words.split()
    length = len(words)
    if length <= window_size:
        windows.append(words)
    else:
        for j in range(length - window_size + 1):
            window = words[j: j + window_size]
            windows.append(window)
```

$$PMI(a,b) = \log(\frac{P(a,b)}{P(\mathrm{a})\mathrm{P}(\mathrm{b})})$$

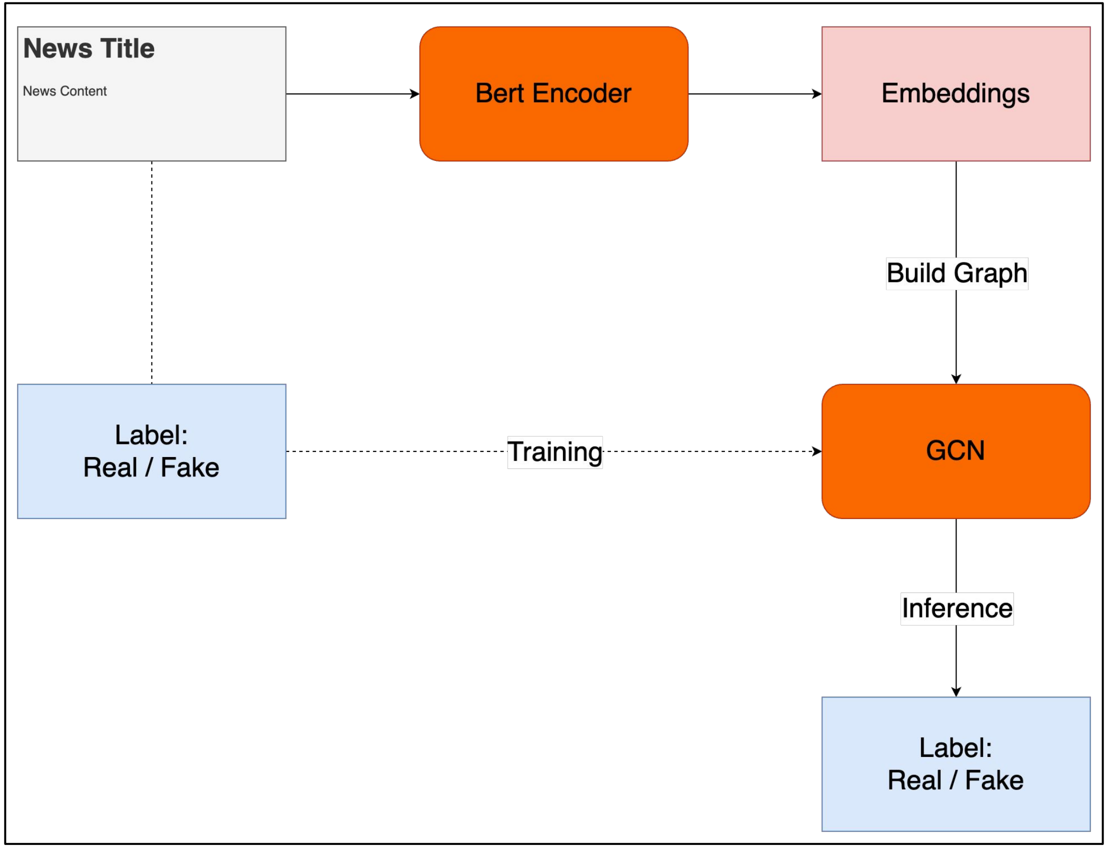
4. TF-IDF+PMI+GCN Classifier

建完graph後，直接將graph的鄰接矩陣當成data丟進去GCN模型訓練，並透過train_mask,test_mask的方式來完成對graph node的遮罩效果

```
# masks for training, testing
train_masks = torch.zeros(node_size).bool()
ids_train=np.array(ids_train)
ids_val=np.array(ids_val)
ids_test=np.array(ids_test)
train_masks[np.concatenate((ids_train,ids_val))] = 1
test_masks = torch.zeros(node_size).bool()
test_masks[ids_test+vocab_size] = 1

data.train_mask = train_masks
data.test_mask = test_masks
```

5. Word Embedding + GCN Classifier



5. Word Embedding + GCN Classifier

先透過BertTokenizer對文本進行tokenize，產生對應的encoding id vector，再將這些vector放入BertModel中，產生對應的embedding，最終將這些embedding透過不同的graph construction策略進行建圖，最後丟入GCN進行訓練。

tokenizer : bert-base-uncased

BertModel : bert-base-uncased

4. TF-IDF+GCN Classifier

Graph Construction策略1: **Cosine Similarity**

將embedding們互相進行cosine_similarity的計算，由於樣本數較多，若用一般的cos similarity算法會導致產生的相似度矩陣太大, RAM塞不下，因此我們採用了sparse matrix的方式來儲存這些cos相似度的值，並設定threshold為0.5，此時的graph density大約為0.4左右

```python
def calculate_similarity_blockwise(embeddings, block_size=100):
    num_embeddings = embeddings.size(0)
    rows, cols, vals = [], [], []

    for i in range(0, num_embeddings, block_size):
        for j in range(i, num_embeddings, block_size):
            block1 = embeddings[i:i+block_size]
            block2 = embeddings[j:j+block_size]
            similarities = F.cosine_similarity(block1.unsqueeze(1), block2.unsqueeze(0), dim=-1)

            # Apply threshold to keep only significant similarities
            threshold = 0.5
            mask = similarities > threshold

            rows.extend((mask.nonzero()[:, 0] + i).tolist())
            cols.extend((mask.nonzero()[:, 1] + j).tolist())
            vals.extend(similarities[mask].tolist())

    # Create sparse matrix
    sparse_matrix = coo_matrix((vals, (rows, cols)), shape=(num_embeddings, num_embeddings))
    return sparse_matrix
```

$$Density = \frac{R}{N(N-1)/2}$$

4. TF-IDF+GCN Classifier

Graph Construction策略2: **KNN**

將embedding們互相進行距離計算，並選出前K個最近的鄰居embedding進行建邊

```
def calculate_similarity_knn(embeddings, k=5, block_size=100):
    num_embeddings = embeddings.size(0)
    rows, cols, vals = [], [], []

    for i in range(0, num_embeddings, block_size):
        for j in range(0, num_embeddings, block_size):
            block1 = embeddings[i:i+block_size]
            block2 = embeddings[j:j+block_size]
            similarities = F.cosine_similarity(block1.unsqueeze(1), block2.unsqueeze(0), dim=-1)

            if i == j:
                topk_similarities, topk_indices = similarities.topk(k + 1, dim=1, largest=True)
                topk_similarities = topk_similarities[:, 1:]  # Remove self-loops
                topk_indices = topk_indices[:, 1:]  # Remove self-loops
            else:
                topk_similarities, topk_indices = similarities.topk(k, dim=1, largest=True)

            rows.extend((torch.arange(i, min(i + block_size, num_embeddings)).unsqueeze(1).repeat(1, k).flatten()).tolist())
            cols.extend((topk_indices + j).flatten().tolist())
            vals.extend(topk_similarities.flatten().tolist())

    # Create sparse matrix
    sparse_matrix = coo_matrix((vals, (rows, cols)), shape=(num_embeddings, num_embeddings))
    return sparse_matrix
```

$$Density = \frac{R}{N(N-1)/2}$$

pipeline 4 & 5的GCN架構

```python
class GCN(torch.nn.Module):
    def __init__(self, num_features, num_classes):
        super(GCN, self).__init__()
        self.conv1 = GCNConv(num_features, 16)
        self.conv2 = GCNConv(16, num_classes)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)
```

05

Results

Results

Model	Validation Accuracy	Model	Validation Accuracy
TF-IDF + LR	0.71	Embeddings + BERT + MLP	0.81
TF-IDF + SVM	0.72	TF-IDF + TextGCN	0.66
Embeddings + LR	0.69	Embeddings + GCN (cos_sim)	0.59
Embeddings + SVM	0.73	Embeddings+GCN (KNN)	0.61

Conclusion

1. 比較了不同的method在fake news detection任務上的應用與表現(此次資料集為純content-based), 並嘗試將NLP method(bert,TF-IDF)、傳統機器學習演算法、GNN在上下游任務階段中混合運用
2. Embeddings + BERT 表現最好
3. 傳統 ML classifier 表現普遍比 GCN 好
4. 使用 TF-IDF & PMI 的GCN method 比使用 Word Embedding+cos_sim or KNN的GCN method還要好

Future Work

1. **改進GNN模型:**

   GNN 表現並沒有比其他模型好, 推測原因可能是我們的GNN模型結構與設計策略較為簡單, 沒有充分地利用與挖掘資料中的高階連通性。

   a. 在建邊階段, 我們可以比較 cosine 相似度不同 threshold 之表現, 以及KNN可以調整不同的K值。

   b. 嘗試 learning-based 的方式 (註一)、或是能針對每個節點去個別學習生成的節點圖譜濾波器來進一步強化異常節點資訊的利用 (註二), 也可以採用不同的GNN model, 例如GAT等。

2. **加入社群網路資訊:**

   由於此次的資料只有新聞文本與label, 若加入新聞發布與用戶互動的情況等額外資訊, 或許就有更多特徵可以利用, 也更能發揮出GNN-based model捕捉複雜交互關係的能力, 以達到更好的模型表現。

註一: 參考 Towards Unsupervised Deep Graph Structure Learning
註二: 參考 Partitioning Message Passing for Graph Fraud Detection

Reference

- https://www.kaggle.com/c/fakenewskdd2020/overview
- https://huggingface.co/docs/transformers/tasks/sequence_classification
- Vosoughi S, Roy D, Aral S. The spread of true and false news online. Science. 2018;359(6380):1146–1151. doi: 10.1126/science.aap9559.

GitHub

https://github.com/LittleFish-Coder/fake-news-detection-model-comparison

Thanks