

# Chinese Spam Message Classification

DS4CS Final Project

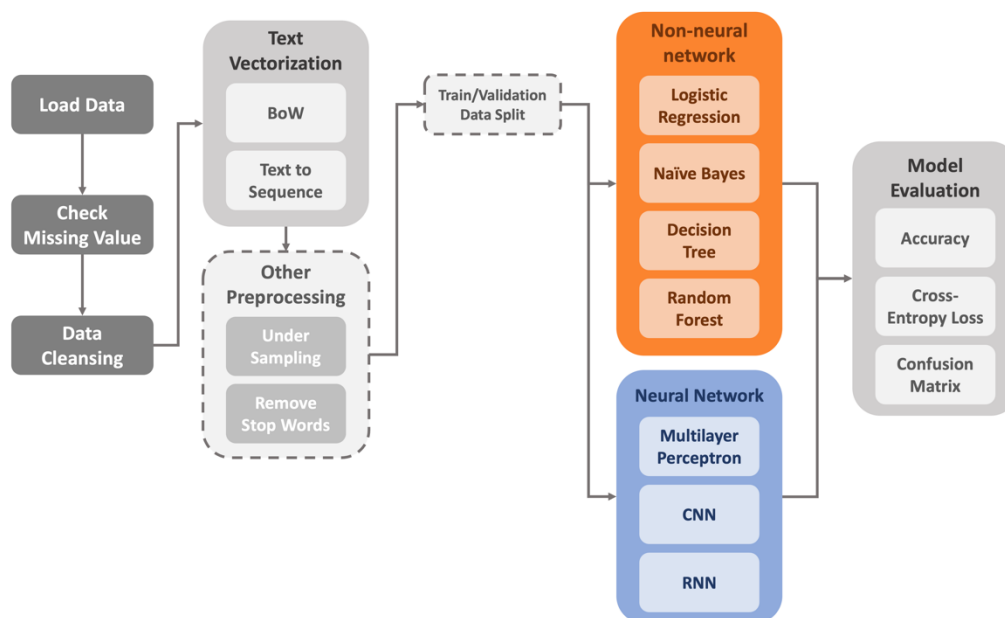
資管三甲 曹靜之 108306095

## Introduction

自從疫情開始後，許多人經常收到垃圾簡訊，如果要有效阻隔垃圾簡訊傳播，第一步便是使用分類模型判斷一條簡訊是否為垃圾簡訊。之前的作業中使用了比較經典的英文簡訊資料集來完成 SMS spam classification，因此期末報告選擇延伸此課題實作中文的垃圾簡訊分類模型，並比較不同模型間的分類成效。

## Methodology

### • Overall Procedure



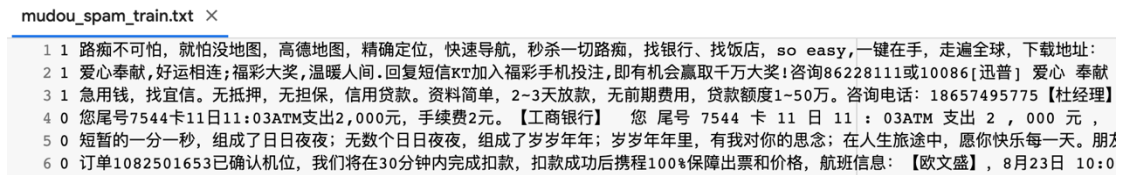
### • Dataset

在尋找資料集的過程中發現中文簡訊的公開資料集明顯相較英文少了許多，最終選擇了一個叫做 mudou\_spam 的中國垃圾簡訊資料集[1]，其資料使用文字檔(.txt)紀錄，每一筆資料共有三個欄位，分別是：垃圾簡訊標記結果(0, 1)、簡訊內容、斷詞結果，中間以 Tab 鍵("\t")隔開（圖 1）。原始資料訓練集共有 10872 筆，測試集共有 1382 筆資料，經過檢查後發現部分缺失值，但數量不多，因此直接將其移除，清理後資料分佈如表 1，垃圾簡訊與非垃圾簡訊的比例約為 16:9，有些許不平衡。

表 1: 資料分佈

	Ham (0)	Spam (1)	Total
Train	3933	6899	10832
Test	491	888	1379

圖 1: 資料集部分截圖



mudou\_spam\_train.txt ×

```
1 1 路痴不可怕,就怕没地图,高德地图,精确定位,快速导航,秒杀一切路痴,找银行、找饭店, so easy,一键在手,走遍全球,下载地址:
2 1 爱心奉献,好运相连;福彩大奖,温暖人间.回复短信KT加入福彩手机投注,即有机会赢取千万大奖!咨询86228111或10086[迅普] 爱心 奉献
3 1 急用钱,找宜信。无抵押,无担保,信用贷款。资料简单, 2~3天放款,无前期费用,贷款额度1~50万。咨询电话: 18657495775 【杜经理】
4 0 您尾号7544卡11日11:03ATM支出2,000元,手续费2元。【工商银行】 您 尾号 7544 卡 11 日 11 : 03ATM 支出 2 , 000 元 ,
5 0 短暂的一分一秒,组成了日日夜夜;无数个日日夜夜,组成了岁岁年年;岁岁年年里,有我对你的思念;在人生旅途中,愿你快乐每一天。朋友
6 0 订单1082501653已确认机位,我们将在30分钟内完成扣款,扣款成功后携程100%保障出票和价格,航班信息: 【欧文盛】, 8月23日 10:0
```

圖 2: 讀取資料與資料清理實作程式碼

```
def get_data(data_file_path):
    lines = [] #經斷詞處理的簡訊
    labels = [] #標籤結果
    for line in open(data_file_path, "r", encoding="utf-8"):
        arr = line.rstrip().split("\t")
        if len(arr) < 3: #資料集應有三欄, 若小於三欄判斷為缺失資料, 將其跳過
            continue
        if int(arr[0]) == 1:
            label = 1
        elif int(arr[0]) == 0 or int(arr[0]) == -1:
            label = 0
        else:
            continue
        labels.append(label)
        line = arr[2].split()
        lines.append(line)
    return lines, labels
```

- Preprocessing

本次報告中主要使用了兩種不同方法將文本轉為向量，

1. Bag-of-words: 將斷詞文字建立為字典，統計所有字詞出現的次數，刪除出現次數<5 的詞彙，可以得到含有 6012 個單詞的詞典，再將每則簡訊是否有出現某單詞分別寫成 6012 維的向量，以訓練集為例，最終得到 (10832, 6012) 的訓練集矩陣。

圖 3: Bag-of-words 實作程式碼

```
def create_vocab_dict(data_lines):
    vocab_dict = {}
    for data_line in data_lines:
        for word in data_line:
            if word in vocab_dict:
                vocab_dict[word] += 1 #如字典裡已經有該詞, 出現次數+1
            else:
                vocab_dict[word] = 1 #如字典裡沒有該詞, 將出現次數初始化設定為1
    return vocab_dict

def BOW_feature(vocab_list, input_line):
    return_vec = np.zeros(len(vocab_list), )
    for word in input_line:
        if word in vocab_list:
            return_vec[vocab_list.index(word)] += 1 #如果簡訊內包含字典內的詞彙, 根據詞彙id標記出現次數+1
    return return_vec
```

2. 因為後續測試會使用 RNN，因此也實作了考慮文字順序的方法，並且只統計垃圾簡訊字詞出現的次數，同樣刪除出現次數<5 的詞彙後，得到含有 4632 個單詞對應出現次數的垃圾簡訊詞典，再將每則簡訊依照斷詞順序查找是否有出現在垃圾簡訊詞典，寫入對應分數。使用時搭配 `sequence.pad_sequences(train_X, maxlen=50)`，將簡訊矩陣限制成同一長度的資料，最終得到 (10832, 50) 的訓練集矩陣。

圖 4: Text to Sequence 實作程式碼

```
def create_vocab_dict(data_lines, label):
    vocab_dict = {}
    for data_line in data_lines:
        for word in data_line:
            if word not in vocab_dict:
                vocab_dict[word] = 0
            vocab_dict[word] += 1 * label[data_lines.index(data_line)] #只考慮詞彙在spam message的出現次數
    return vocab_dict

def sequence_feature(vocab_dict, input_line):
    return_vec = []
    for word in input_line:
        if word in vocab_dict:
            return_vec.append(vocab_dict[word])
        else:
            return_vec.append(0)
    return return_vec
```

除以上兩種文本轉向量的方法外，因為觀察到資料集有些許不平衡且原資料有將標點符號、數字納入詞典的狀況，在資料預處理上還做了 `under sampling` 與排除 `stop words` 的嘗試，搭配此兩種文本轉向量的方法使用，並比較其訓練成效，其中 `stop words` 採用 github 上 goto456 提供的中文停用詞表，共有 746 個停用詞。[2]

圖 5: Under Sampling 實作程式碼

```
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=42)
resampled_X, resampled_label = rus.fit_resample(train_X, train_label)
```

圖 6: 排除 Stop Words 實作程式碼

```
stopwords = []
for stopword in open('cn_stopwords.txt', 'r', encoding="utf-8"):
    stopwords.append(stopword.split('\n')[0])
vocab_list = [v for v in vocab_list if v not in stopwords]
```

- Algorithms

因為垃圾簡訊分類屬於二元分類問題，在本次報告中分別從 `non-neural network` 與 `neural network` 兩種大分類挑選了總共七種演算法進行比較。在 `non-neural network` 中挑選了四種常用演算法，分別是：資料集作者使用的 Logistic Regression、比較經典的分類問題演算法 Naïve Bayes、能看到判斷節點的 Decision Tree、以及決策樹的升級版 Random Forest。而在 `neural network` 中則挑選了三種神經網路模型，分別是 Multilayer Perceptron、CNN 以及常用於文本分析的 RNN。

Non-neural network 的模型使用 scikit-learn 建立，程式碼如下。

圖 7: Logistic Regression 實作程式碼

```
from sklearn.linear_model import LogisticRegression
reg_model = LogisticRegression(max_iter=1000)
reg_model.fit(train_X, train_label)
```

圖 8: Naïve Bayes 實作程式碼

```
from sklearn.naive_bayes import GaussianNB
nb_model = GaussianNB()
nb_model.fit(train_X, train_label)
```

在 Decision Tree 的 max\_depth 設定上，測試了 max\_depth = 6~15 的結果，使用驗證集驗證後得到 Best Max Depth = 10，因此使用 max\_depth = 10 建立 Decision Tree。

圖 9: Decision Tree 實作程式碼

```
from sklearn.tree import DecisionTreeClassifier
max_dep = 15
best_max_dep = 15
min_loss = 10
while max_dep > 5:
    dt_model = DecisionTreeClassifier(max_depth = max_dep)
    dt_model.fit(train_X, train_label_)
    pred_prob_y = dt_model.predict_proba(val_X)[: , 1]
    current_loss = metrics.log_loss(val_label, pred_prob_y)
    if current_loss < min_loss:
        min_loss = current_loss
        best_max_dep = max_dep
    max_dep -= 1
print("Best Max Depth: ", best_max_dep)
print("Min Cross Entropy Loss: ", min_loss)
```

```
Best Max Depth: 10
Min Cross Entropy Loss: 0.4293096367112812
```

```
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier(max_depth = 10)
dt_model.fit(train_X, train_label)
```

Random Forest 則測試了樹的數量(n\_estimators)設定，測試了 max\_depth = 100~1000 的結果，實測發現 n\_estimators 在 100 到 200 間效果有明顯差異，n\_estimators > 200 後，訓練成效成長趨緩但有一些波動，而在 n\_estimators = 200 時模型的 loss 剛好能夠達到與 n\_estimators = 1000 相差無幾的水準，綜合考量訓練時間與訓練效果，最終將 n\_estimators 設定為 200。

圖 10: Random Forest 實作程式碼

```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=200, random_state=42)
rf_model.fit(train_X, train_label)
```

Neural network 的模型使用 scikit-learn 與 tensorflow 建立，程式碼如下。在模型設計上，CNN 使用 Conv1D 搭配全連接層，RNN 使用 LSTM 搭配 Dropout 防止 Overfitting，激發函數混合使用 Sigmoid 和 ReLU，optimizer 則是選擇 Adam。訓練時則搭配使用 Early Stopping 與 Model Checkpoint，防止過度訓練耗時又容易造成過擬和。

圖 11: MLP 實作程式碼

```
from sklearn.neural_network import MLPClassifier
mlp_model = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=42)
mlp_model.fit(train_X, train_label)
```

圖 12: CNN 實作程式碼

```
from tensorflow.keras.layers import Conv1D
from tensorflow.keras.layers import MaxPooling1D
from tensorflow.keras.layers import Flatten
input_shape=(677,16,6012)
cnn_model = Sequential()
cnn_model.add(Conv1D(32,1, activation='relu'))
cnn_model.add(Dense(128, activation='relu'))
cnn_model.add(Dense(1, activation='sigmoid'))
cnn_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
cnn_model.build(input_shape)
cnn_model.summary()
```

Model: "sequential\_3"

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(677, 16, 32)	192416
dense_4 (Dense)	(677, 16, 128)	4224
dense_5 (Dense)	(677, 16, 1)	129
Total params: 196,769		
Trainable params: 196,769		
Non-trainable params: 0		

```
train_X=np.reshape(train_X,(677,16,6012))
train_label=np.reshape(train_label,(677,16,1))
```

圖 13: RNN (LSTM) 實作程式碼

```
rnn_model = Sequential()
rnn_model.add(Embedding(10832, 128))
rnn_model.add(LSTM(128))
rnn_model.add(Dropout(0.2)) # 增加Dropout層, 防止過擬和
rnn_model.add(Dense(1, activation='sigmoid'))
rnn_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
rnn_model.summary()
```

Model: "sequential\_11"

Layer (type)	Output Shape	Param #
embedding_9 (Embedding)	(None, None, 128)	1386496
lstm_9 (LSTM)	(None, 128)	131584
dropout_10 (Dropout)	(None, 128)	0
dense_15 (Dense)	(None, 1)	129
Total params: 1,518,209		
Trainable params: 1,518,209		
Non-trainable params: 0		

- Evaluation

本報告使用 Cross Entropy Loss 與 Test Accuracy 評估模型，搭配 Confusion Matrix 查看不同模型間判斷結果的差異。

Result

• Best Model

在演算法分類效果的比較中，因為模型參數設定與使用的訓練資料處理方式不同會導致模型相關數據不同，所以首先比較各演算法成效最好的模型。在非神經網路的模型中，表現較好的是 Logistic Regression 與 Random Forest（表 2），而在神經網路的模型中，RNN 的 Cross-Entropy Loss 較低，而 MLP 則有比較好的準確度表現。（表 3）

表 2: Non-neural Network Best Model 分類效果比較

	Logistic Regression	Decision Tree	Random Forest	Naïve Bayes
Accuracy	0.9637	0.9007	0.9681	0.9275
Cross-Entropy Loss	0.1137	0.3371	0.1200	2.5046

圖 14: Non-neural Network Confusion Matrix

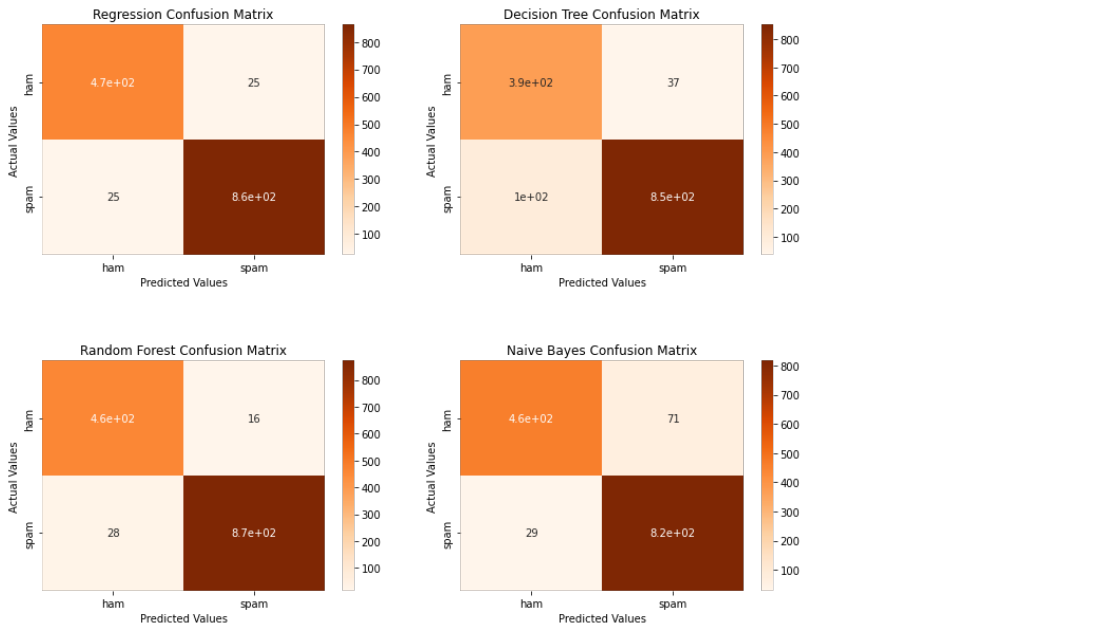
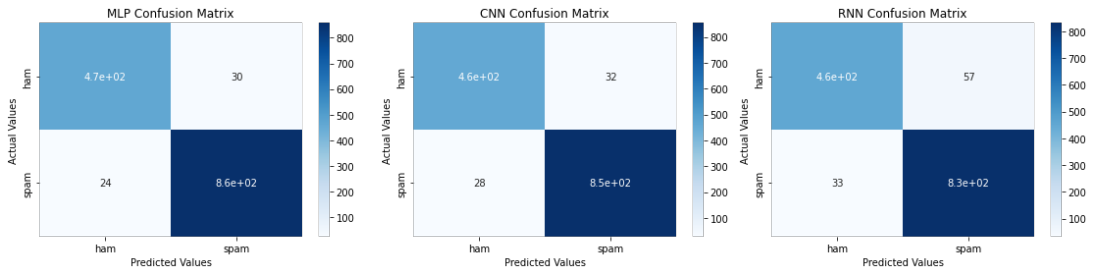


表 3: Neural Network Best Model 分類效果比較

	MLP	CNN	RNN
Accuracy	0.9608	0.9564	0.9347
Cross-Entropy Loss	0.6117	0.3267	0.1671

圖 15: Neural Network Confusion Matrix



值得注意的是，在這個資料集上，神經網路模型相較於非神經網路模型並沒有特別突出的表現，雖然這有可能是因為神經網路參數並沒有調整到最佳狀態，但也說明了非神經網路在模型表現上並不一定總是遜於神經網路，還是要視資料與模型設計而定。

接著比較文本轉向量的方法，所有報告中的模型，只有 RNN 在使用 Text-to-Sequence 的時候表現比較好，其他模型皆是使用 Bag-of-Words 進行文本向量轉換效果較好。（表 4）推測可能是因為兩種方法的資料維度相差太多，而 non-neural network 又比較仰賴大量資料變數，因此較不適合 Text-to-Sequence 的處理方式。在四種 non-neural network 的模型中，只有 Random Forest 表現較不受影響，可能與其本身模型結構較不依賴資料變數（樹的深度）有關。

表 4: Non-neural Network 文本轉向量方法比較

		<b>Logistic Regression</b>	<b>Decision Tree</b>	<b>Random Forest</b>	<b>Naïve Bayes</b>
Accuracy	BoW	0.9637	0.9007	0.9681	0.9275
	Text2Sequence	0.7194	0.8593	0.9108	0.6940
Cross-Entropy Loss	BoW	0.1137	0.3371	0.1200	2.5046
	Text2Sequence	0.5522	1.2981	0.2168	2.0788

最後比較兩種資料預處理的方式，分別是 Under Sampling 與 Remove Stop Words，其中 Under Sampling 會使資料筆數縮減到 7866 筆，得到(7866, 6012)的訓練集矩陣，而 Remove Stop Words 則會使詞袋中詞數減少 5759，得到(10832, 5759)的訓練集矩陣。可以發現不管是 Under Sampling 或是 Stop Words Removal，對模型效果的提升並沒有太大幫助（表 5, 表 6），這可能是因為這兩種預處理方式都會導致資料維度縮減而損失一定的原始資料。在 Logistic Regression 的測試中，Stop Words Removal 能夠小幅提升模型效果（表 6），顯示資料集中確實存在應移除的停用詞，但此次只測試了一種停用詞表，且並不是專為垃圾簡訊設計，或許在未來可以多測試幾個停用詞表的效果找尋更適合的停用詞表。

表 5: Non-neural Network 原始資料與經過 Under Sampling 比較 Stop Words Removal

		<b>Logistic Regression</b>	<b>Decision Tree</b>	<b>Random Forest</b>	<b>Naïve Bayes</b>
Accuracy	Original	0.9637	0.9007	0.9681	0.9275
	Under Sampled	0.9616	0.8963	0.9608	0.9275
Cross-Entropy Loss	Original	0.1137	0.3371	0.1200	2.5046
	Under Sampled	0.1286	0.5747	0.1455	2.5046



表 6: Non-neural Network 原始資料與經過 Stop Words Removal 比較

		<b>Logistic Regression</b>	<b>Decision Tree</b>	<b>Random Forest</b>	<b>Naïve Bayes</b>
Accuracy	Original	0.9637	0.9007	0.9681	0.9275
	Stop Words Removed	0.9652	0.8339	0.9645	0.9275
Cross- Entropy Loss	Original	0.1137	0.3371	0.1200	2.5046
	Stop Words Removed	0.1177	0.4562	0.1277	2.5046

## Discussion

本次報告主要比較了不同模型在處理中文垃圾簡訊分類問題時的效果，同時也比較了不同資料預處理方式對模型成效的影響，報告中發現在中文垃圾簡訊分類問題上，非神經網路可能是一個效率不錯的選擇，其中可以考慮使用 Logistic Regression 或 Random Forest 來達到比較好的分類效果。會有這樣的想法是因為在神經網路的模型訓練與反覆測試的過程中，由於使用的是 Google Colab 的線上運算資源，發生了數次 RAM 與 GPU 運算資源不足的情況，導致訓練一直中斷，後來升級 Colab Pro 才得以解決。但如果不考慮運算資源的情況下，RNN 依舊是一個很好的選擇。

我認為中文與英文垃圾簡訊分類問題的差異主要還是在於斷詞、停用詞等資料預處理的步驟。而此次報告中使用的是已經有提供建議斷詞的資料集，因此比較沒有斷詞方面的問題，但在觀察資料集時有發現大部分的垃圾簡訊帶有公司名稱等特殊名詞，這樣的名詞可能比較會造成大量斷詞的困難。而停用詞的部分，我認為有些普遍的停用詞可能會是垃圾簡訊的判斷變數（例如：特殊符號，因為有些廣告簡訊會有特定排版或是亂碼），如果直接移除可能會導致模型效果不升反降，因此尋找或自己建立適合垃圾簡訊的停用詞表可能會有更好的效果。

## Reference

- [1] 資料集來源 <https://github.com/shaonianruntu/Spam-Message-Classification/>
- [2] 中文停用詞表 <https://github.com/goto456/stopwords>