# U.S. Airline.

## Twitter評論分析

109034544 姜柏宏 109034545 蘇沛強
109034550 周宛昀 109034552 謝予欣

# TABLE OF CONTENTS

# 1

# SCENARIO

主題說明

# background

## CHOOSE

出國時選擇航空公司很重要，選擇安全、服務良好的公司。

## INFORMATION

不可能一篇評價都有時間觀看，而且文字過長的評論也不適閱讀。

## EVALUATION

直接幫使用者判斷評論是偏向正/負面或是中立的，讓使用者判斷航空公司的優劣。

# 5W1H

**WHAT**

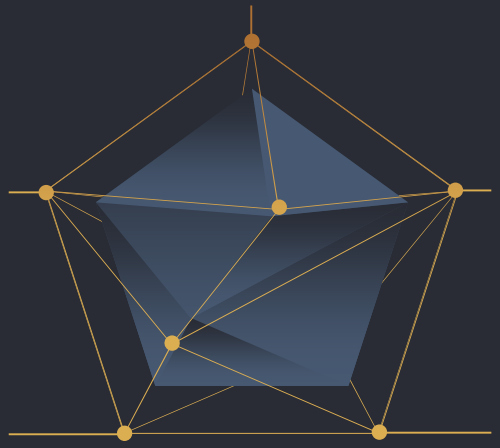
判斷評論是正/負面或中立評價

**WHEN**

想要選擇航空公司時

**WHERE**

所有的網路平台及論壇



**WHO**

想選擇航空公司的旅客

**WHY**

節省使用者自行判斷的時間

**HOW**

Word embedding 、
LSTM、SSO

# 2

# DATA PREPROCESSING

資料預處理

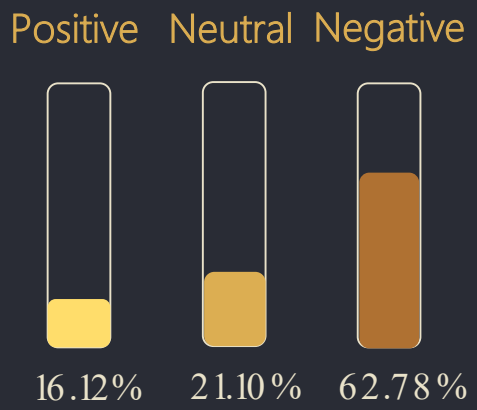
This data originally came from Crowdflower's Data for Everyone library .
As the original source says,
A sentiment analysis job about the problems of each major **U.S. airline. Twitter data** was scraped from February of 2015 and contributors were asked to first classify **positive, negative, and neutral** tweets, followed by **categorizing negative reasons** (such as "late flight" or "rude service").

Data from Kaggle

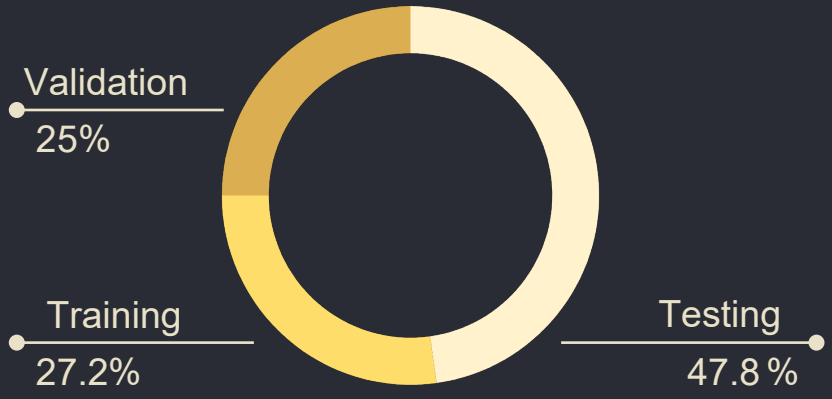
# Data Describe

14605筆資料

## 評價比例




## 資料分割



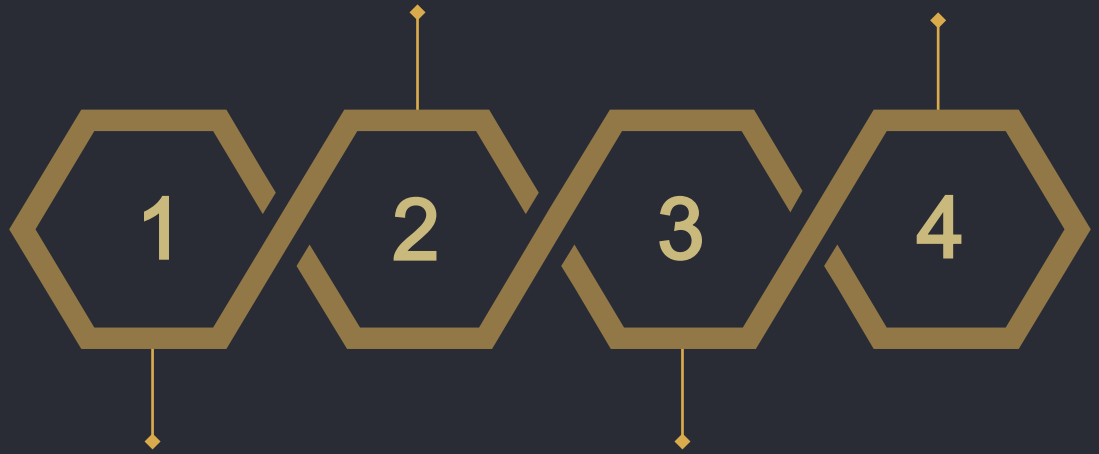

## Sample data

**Positive** : LTALJX, from DCA to OMA this morning. All of the staff that helped me fix my problem were so helpful. #goodthingscome #thanks

**Negative** : what's happening with 1217 Phl to LAX? Now 3 hr delay. Poor communication!

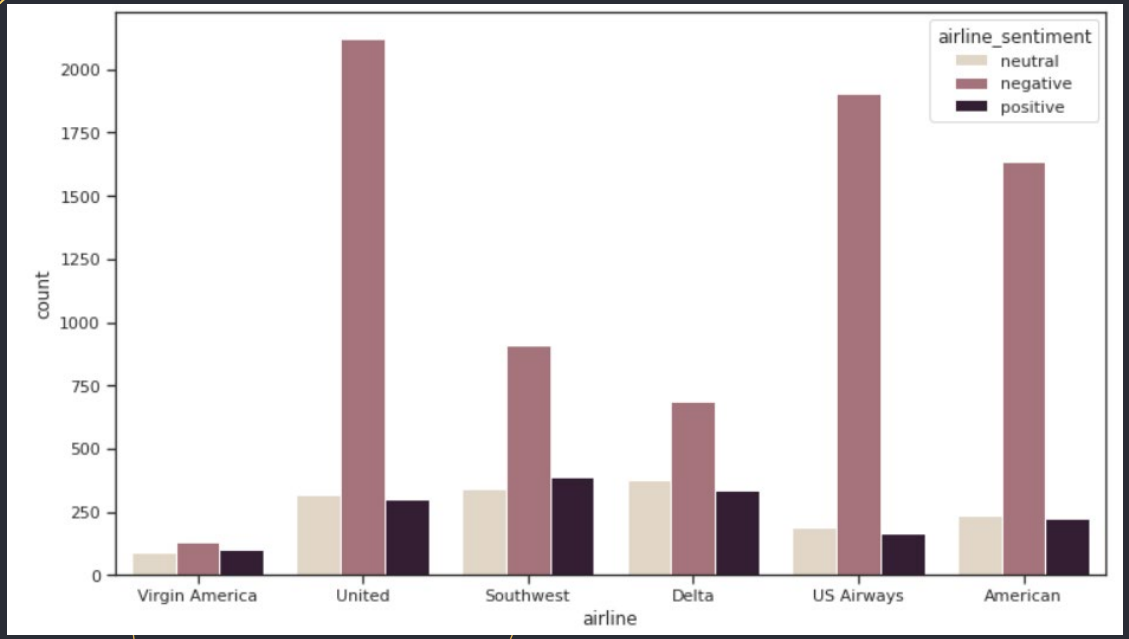
**Neutral :** yes, after this awful weather it appears I can get home

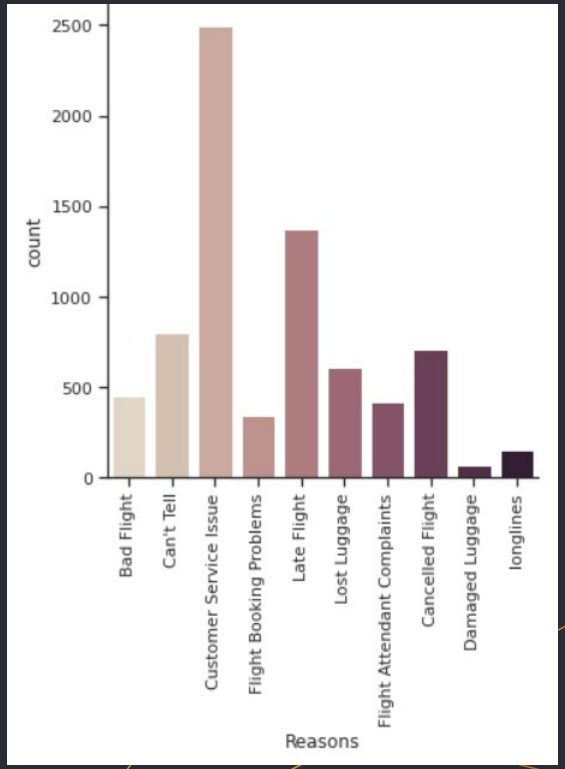
# Data Preprocessing Process




資料特徵模型內會自行尋找

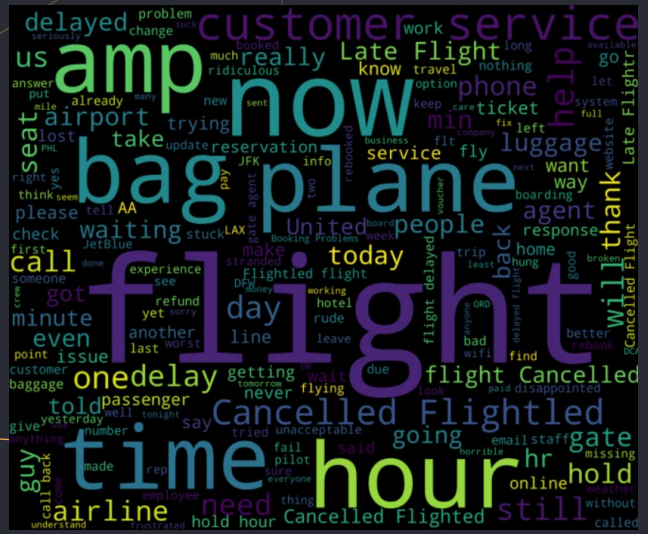
# Data Analysis



# Data Analysis

- 資料中發現大部份推特內容都與時間管理或個人物品有關，或者向社群尋求幫助
- 若”help”出現於推文中，則高機率為負面評論，表示航空公司某部分服務不周，造成顧客需要尋求幫助
- 若出現”thanks”或” gratitude”則對航空公司形象有助益，表示航空公司有提供了讓顧客滿意的服務




# Data Cleaning

- 分析出高缺失欄位並刪除

| | |
|---|---|
| tweet_id | 0.000000 |
| airline_sentiment | 0.000000 |
| airline_sentiment_confidence | 0.000000 |
| negativereason | 0.373087 |
| negativereason_confidence | 0.281284 |
| airline | 0.000000 |
| airline_sentiment_gold | 0.997268 |
| name | 0.000000 |
| negativereason_gold | 0.997814 |
| retweet_count | 0.000000 |
| text | 0.000000 |
| tweet_coord | 0.930396 |
| tweet_created | 0.000000 |
| tweet_location | 0.323292 |
| user_timezone | 0.329235 |

- 將評價分類信心度不足80%的資料刪除, 刪除了資料集中的29%資料

```
Best_confidence = Tweet[Tweet['airline_sentiment_confidence'] > .80]
print('you have dropped :'+ str((round((1 - Best_confidence.shape[0]/Tweet.shape[0]) * 100 )))+ '% of your data which has less that 80% confidence')
Tweet = Best_confidence
```

```
you have dropped :29% of your data which has less that 80% confidence
```

# Standardization

1. 將推文轉換為一個一個單詞，並刪除 stopword

*stopword example : “the”, “a”, “an”, “in”

```
def tweet_to_words(raw_tweet):
    letters_only = re.sub("[^a-zA-Z]", " ",raw_tweet)
    words = letters_only.lower().split()
    stops = set(stopwords.words("english"))
    meaningful_words = [w for w in words if not w in stops]
    return( " ".join( meaningful_words ))
```

2. 將推文內容轉換全英文小寫

```
def clean_tweet_length(raw_tweet):
    letters_only = re.sub("[^a-zA-Z]", " ",raw_tweet)
    words = letters_only.lower().split()
    stops = set(stopwords.words("english"))
    meaningful_words = [w for w in words if not w in stops]
    return(len(meaningful_words))
```

# Standardization

3. 將推文性質轉換為數字('negative' = 0, 'neutral' = 1, 'positive' = 2 )

```
train_df = Tweet[['text', 'airline_sentiment']]
train_df.text = Tweet.text.apply(remove_mentions)
train_df.loc[:,'sentiment'] = train_df.airline_sentiment.map({'negative':0,'neutral':1,'positive':2})
train_df = train_df.drop(['airline_sentiment'], axis=1)
```

# 3

# MODEL ARCHITECTURE

模型架構

# Onehot Encoding

使用N位狀態暫存器來對N個狀態進行編碼，每個狀態都由他獨立的暫存器位，並且在任意時候，其中只有一位有效。

這樣做的好處主要有：

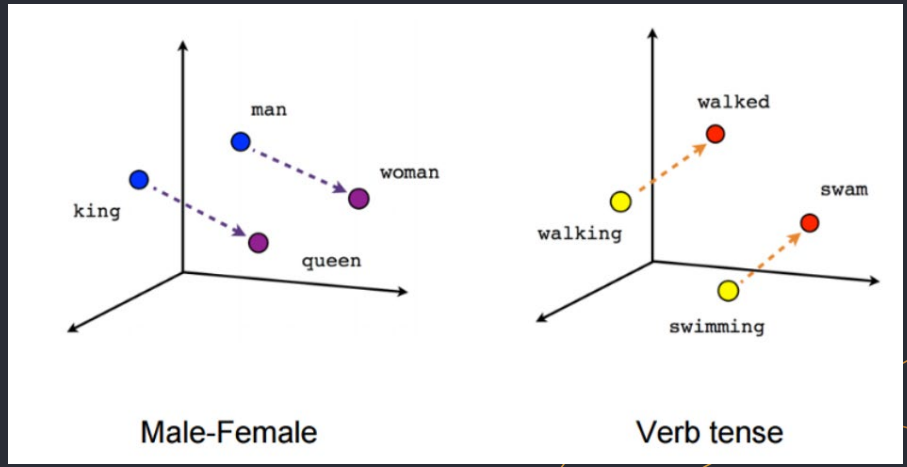
1. 解決了分類器不好處理屬性資料的問題
2. 在一定程度上也起到了擴充特徵的作用

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| I | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ate | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| an | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| apple | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| and | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| played | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| the | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| piano | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

# Word Embedding

- 概念：

如果將word看作文本的最小單元，可以將Word Embedding理解為一種映射，其過程是：將文本空間中的某個word，通過一定的方法，映射或者說嵌入到另一個數值向量空間，而通常這個過程維度會隨之降低。



# Word Embedding

- Word Embedding的輸入：

Word Embedding的輸入是將原始文本中的一組不重疊的詞彙，將這些詞彙放置到一個dictionary裡，這個dictionary就可以看作是Word Embedding的一個輸入

假設有句子：apple on a apple tree => [“apple”，“on”，“a”，“tree”]

- Word Embedding的輸出：

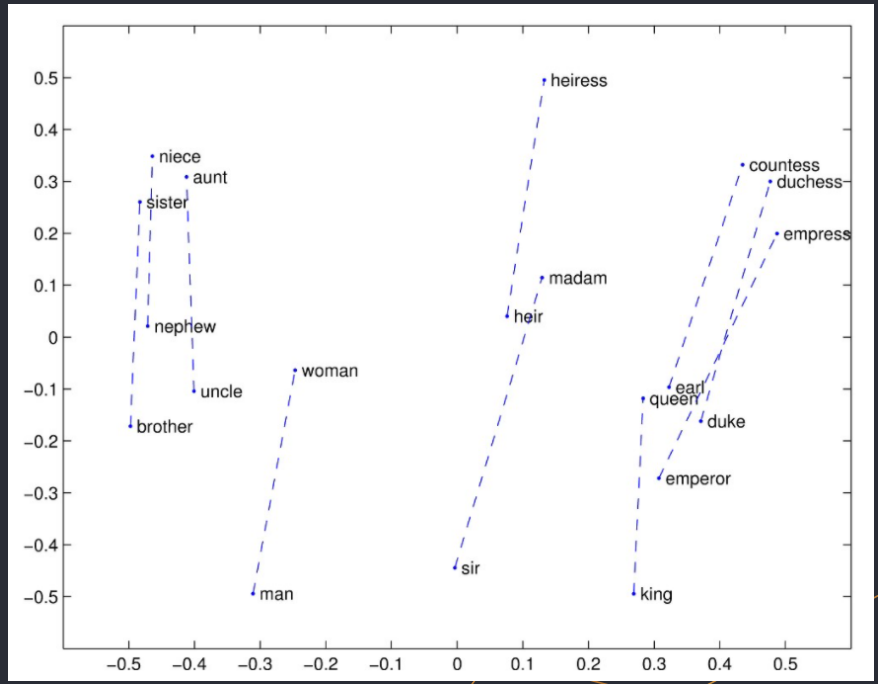
Word Embedding的輸出就是每個word的向量表示。對於上文中的原始輸入，每個word都對應了一種數值表示。例如，apple對應的vector就是[1, 0, 0, 0]，a對應的vector就是[0, 0, 1, 0]，各種機器學習應用可以基於這種word的數值表示來構建各自的模型。

# Word Embedding

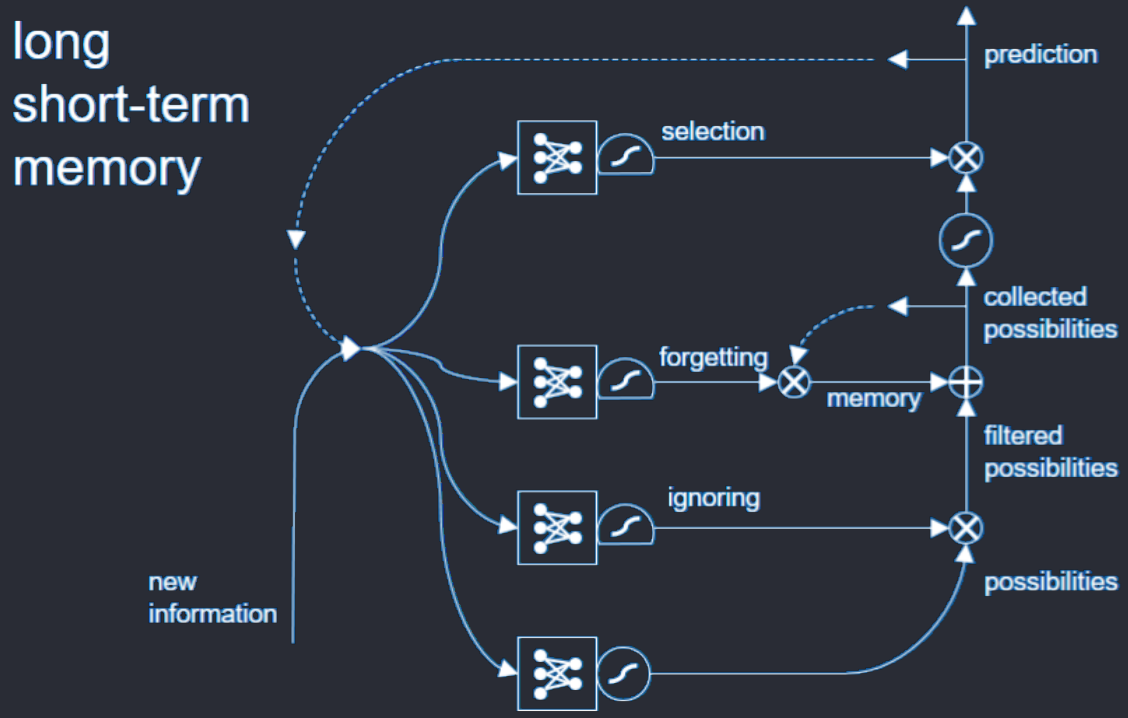
- Frequency based embedding
  - Count Vector
  - TF-IDF Vector
- Predicted Based Embedding
  - Word2vec
  - Doc2Vec
  - **GloVe**

# GloVe(Global Vectors for Word Representation)

GloVe 的全稱叫Global Vectors for Word Representation，它是一個基於**全局詞頻統計**（count-based &overall statistics）的詞表徵（word representation）工具，它可以把一個單詞表達成一個由實數組成的向量，這些向量捕捉到了單詞之間一些語義特性，比如相似性（sim ilarity）、類比性（analogy）等。我們通過對向量的運算，比如歐幾里得距離或者cosine相似度，可以計算出兩個單詞之間的語義相似性。



# LSTM (long short-term memory)



# Simplified Swarm Optimization (SSO)

參數$C_g$、$C_p$和$C_w$是預先設定好的，這當中，$C_g$、$C_p$和$C_w$的值分別由$c_g$、$c_g+c_p$和$c_g+c_p+c_w$累加而成，$X_{ij}^t$代表在第 $t$ 迭代中的粒子 $i$，$\rho_{ij}^t$是介於 [0,1] 之間均勻分佈的隨機數，$x$ 則是變數 $j$ 的隨機值。

如果$0 \le \rho_{ij}^t < C_g$成立，那麼該欄位的值將會被 *gbest* 相對應欄位的值所取代。
如果$C_g \le \rho_{ij}^t < C_p$成立，那麼該欄位的值將會被 *pbest* 相對應欄位的值所取代。
如果$C_p \le \rho_{ij}^t < C_w$成立，那麼該欄位的值將會被保留下來。
如果$C_w \le \rho_{ij}^t < 1$成立，那麼該欄位的值將會被隨機值x所取代，其中這一隨機值 x可以幫助解在更新時，增加粒子的變異性，避免解困在區域解裡。

$$x_{ij}^{\ t} = \begin{cases} x_{ij}^t & \text{if } \rho_{ij}^t \in [0, C_w) \\ p_{ij} & \text{if } \rho_{ij}^t \in [C_w, C_p) \\ g_i & \text{if } \rho_{ij}^t \in [C_p, C_g) \\ x & \text{if } \rho_{ij}^t \in [C_g, 1) \end{cases}$$

《W. Yeh, "Study on quickest path networks with dependent components and apply to RAP. Report, "NSC 97-2221-E-007-099-MY3, 2008-20112011》

# 4

# MODEL SETTING

模型設定

# Define SSO

```
def SSO_Init():
    sol_list = []
    sol_list.append(np.random.randint(1, 4))
    sol_list.append(np.random.randint(64, 256))
    sol_list.append(np.random.randint(1, 4))
    sol_list.append(np.random.randint(64, 256))
    sol_list.append(np.random.uniform(0.4,0.8))
    sol_list.append("softmax")
    sol_list.append(np.random.randint(16, 128))

    return sol_list
init_best_dict = SSO_Init()
```

初始化SSO，利用SSO來選擇參數配置，初始化:

1. LSTM層數
2. LSTM層節點數
3. 全連接層層數
4. 全連接層節點數
5. dropout 比例
6. 激勵函數選擇
7. 訓練批量

# Define SSO

```
# 建立best_dict 紀錄gbest, pbest 與其fitness值
best_dict = {
        'gbest_sol': init_best_dict,
        'gbest_fitness': 0,

        'pbest_LSTM_layers': [0, 0],
        'pbest_LSTM_layers_fitness': [0, 0],

        'pbest_LSTM_units': [0, 0],
        'pbest_LSTM_units_fitness': [0, 0],

        'pbest_Dense_layers': [0, 0],
        'pbest_Dense_layers_fitness': [0, 0],

        'pbest_Dense_nurons': [0, 0],
        'pbest_Dense_nurons_fitness': [0, 0],

        'pbest_Drop_rate': [0, 0],
        'pbest_Drop_rate_fitness': [0, 0],

        'pbest_batch': [0, 0],
        'pbest_batch_fitness': [0, 0],
```

```
# Parameter for SSO
Ngen = 10
Nsol = 2
```

本模型用了:
10個generation
每個世代中有2個solution

gbest:全域最佳解
pbest:該solution最佳解

# Define SSO update

```python
def SSO_UPDATE(sol, sol_list, best_dict):

    # LSTM_layers
    rnd = np.random.rand(1)
    if rnd < 0.3:
        sol_list[0] = best_dict['gbest_sol'][0]
    elif rnd < 0.5:
        sol_list[0] = best_dict['pbest_LSTM_layers'][sol]
    else:
        sol_list[0] = np.random.randint(1, 4)

    # LSTM_units
    rnd = np.random.rand(1)
    if rnd < 0.3:
        sol_list[1] = best_dict['gbest_sol'][1]
    elif rnd < 0.5:
        sol_list[1] = best_dict['pbest_LSTM_units'][sol]
    else:
        sol_list[1] = np.random.randint(64, 256)

    # Dense_layers
    rnd = np.random.rand(1)
    if rnd < 0.3:
        sol_list[2] = best_dict['gbest_sol'][2]
    elif rnd < 0.5:
        sol_list[2] = best_dict['pbest_Dense_layers'][sol]
    else:
        sol_list[2] = np.random.randint(1, 4)
```

```python
    # Dense_nurons
    rnd = np.random.rand(1)
    if rnd < 0.3:
        sol_list[3] = best_dict['gbest_sol'][3]
    elif rnd < 0.5:
        sol_list[3] = best_dict['pbest_Dense_nurons'][sol]
    else:
        sol_list[3] = np.random.randint(64, 256)

    # DropOut_rate
    rnd = np.random.rand(1)
    if rnd < 0.3:
        sol_list[4] = best_dict['gbest_sol'][4]
    elif rnd < 0.5:
        sol_list[4] = best_dict['pbest_Drop_rate'][sol]
    else:
        sol_list[4] = np.random.uniform(0.4,0.8)

    #  Batch_size
    rnd = np.random.rand(1)
    if rnd < 0.3:
        sol_list[6] = best_dict['gbest_sol'][6]
    elif rnd < 0.5:
        sol_list[6] = best_dict['pbest_batch'][sol]
    else:
        sol_list[6] = np.random.randint(16, 128)

    return sol_list
```

更新SSO，透過亂數選擇下一個解打算用的參數，若之前使用過較好的參數(準確度較高)，有較高機率會選擇到該參數來使用。

# Define SSO Update(gbest & pbest)

```
def best_UPDATE(sol, sol_fitness, sol_list, best_dict):

        if sol_fitness >= best_dict['gbest_fitness']:
                best_dict['gbest_sol'] = sol_list
                best_dict['gbest_fitness'] = sol_fitness

        if sol_fitness >= best_dict['pbest_LSTM_layers_fitness'][sol]:
                best_dict['pbest_LSTM_layers_fitness'][sol] = sol_fitness
                best_dict['pbest_LSTM_layers'][sol] = sol_list[0]

        if sol_fitness >= best_dict['pbest_LSTM_units_fitness'][sol]:
                best_dict['pbest_LSTM_units_fitness'][sol] = sol_fitness
                best_dict['pbest_LSTM_units'][sol] = sol_list[1]

        if sol_fitness >= best_dict['pbest_Dense_layers_fitness'][sol]:
                best_dict['pbest_Dense_layers_fitness'][sol] = sol_fitness
                best_dict['pbest_Dense_layers'][sol] = sol_list[2]
```

```
        if sol_fitness >= best_dict['pbest_Dense_nurons_fitness'][sol]:
                best_dict['pbest_Dense_nurons_fitness'][sol] = sol_fitness
                best_dict['pbest_Dense_nurons'][sol] = sol_list[3]

        if sol_fitness >= best_dict['pbest_Drop_rate_fitness'][sol]:
                best_dict['pbest_Drop_rate_fitness'][sol] = sol_fitness
                best_dict['pbest_Drop_rate'][sol] = sol_list[4]

        if sol_fitness >= best_dict['pbest_batch_fitness'][sol]:
                best_dict['pbest_batch_fitness'][sol] = sol_fitness
                best_dict['pbest_batch'][sol] = sol_list[6]

        return best_dict
```

# Define LSTM Model

```
def ltsm_model(input_shape, word_to_vec_map, word_to_index, sol):

    sentence_indices = Input(shape=input_shape, dtype='int32')
    embedding_layer = pretrained_embedding_layer(word_to_vec_map,
                                                 word_to_index)
    embeddings = embedding_layer(sentence_indices)

    for layer in range(sol[0]):
        X = LSTM(sol[1], return_sequences=True)(embeddings)

    X = LSTM(sol[1], return_sequences=False)(X)
    X = Dropout(sol[4])(X)

    for layer in range(sol[2]):
        X = Dense(sol[3], activation=None)(X)

    X = Dropout(sol[4])(X)
    X = Dense(3, activation=None)(X)
    X = Activation('softmax')(X)

    model = Model(inputs=[sentence_indices], outputs=X)

return model
```

利用前面SSO選擇的參數建立LSTM模型，使用到sol[0](LSTM層數)、sol[4](斷開比例)、sol[2](全連接層層數)、sol[3]全連接層節點數，並將其組合及準確度記錄下來，選出一組最好的組合進行正式訓練。

# Define Train Evaluate

```
from keras.optimizers import Adam,Adagrad,RMSprop,Adadelta
adadelta = Adadelta(lr=0.1, rho=0.95 ,epsilon=None, decay=0.0)

def train_evaluate(model, sol_list):
    model.compile(loss='categorical_crossentropy', optimizer=adadelta,
                  metrics=['accuracy'])
    model.fit(X_train_indices, y=Y_oh_train, batch_size=sol_list[6],
              epochs=100, verbose=1,
              validation_data=(X_vaild_indices, Y_oh_vaild))

    # Plot train/test loss and accuracy
    acc = hist_1.history['accuracy']
    val_acc = hist_1.history['val_accuracy']
    loss = hist_1.history['loss']
    val_loss = hist_1.history['val_loss']
```

# 5

# TRAINING & TESTING

訓練及測試

# Start SSO

```
# Initial solution list with random solution
print("Initialization")
for i in range(Nsol):
        sol_list = SSO_Init()
        sso_info.append(sol_list)
        print(f"Initialize sol_list{i}:{sol_list}")
        model = ltsm_model((maxLen,), word_to_vec_map, word_to_index, sol_list)
        sol_fitness = train_evaluate(model, sol_list)
        best_dict = best_UPDATE(i, sol_fitness, sol_list, best_dict)
        print(f'best_dict of initialization: {best_dict}')

# Start SSO
for gen in range(Ngen):
        print(f'=============== Generation_{gen} ===============')
        for j in range(Nsol):
                sol_list = SSO_Init()
                sol_list = SSO_UPDATE(j, sol_list, best_dict)
                sso_info.append(sol_list)
                print(f"sol_{j} sol_list: {sol_list}")
                model = ltsm_model((maxLen,), word_to_vec_map, word_to_index, sol_list)
                sol_fitness = train_evaluate(model, sol_list)
                best_dict = best_UPDATE(j, sol_fitness, sol_list, best_dict)
                fitness_info = [sol_fitness]
                plot_info.append(fitness_info)
                print(plot_info)
        print('Global best fitness: {}'.format(best_dict['gbest_fitness'])
        print('Global best solution: {}'.format(best_dict['gbest_sol']))
```

# Select Hyperparameter

| | |
|---|---|
| LSTM層數 | 2 |
| LSTM層節點數 | 107 |
| 全連接層層數 | 1 |
| 全連接層節點數 | 204 |
| dropout比例 | 0.4554 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 117 |
| accuracy | 80.30% |

| | |
|---|---|
| LSTM層數 | 2 |
| LSTM層節點數 | 129 |
| 全連接層層數 | 3 |
| 全連接層節點數 | 233 |
| dropout比例 | 0.7961 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 101 |
| accuracy | 80.15% |

| | |
|---|---|
| LSTM層數 | 2 |
| LSTM層節點數 | 171 |
| 全連接層層數 | 2 |
| 全連接層節點數 | 150 |
| dropout比例 | 0.5772 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 97 |
| accuracy | 80.38% |

| | |
|---|---|
| LSTM層數 | 2 |
| LSTM層節點數 | 129 |
| 全連接層層數 | 1 |
| 全連接層節點數 | 187 |
| dropout比例 | 0.5455 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 83 |
| accuracy | 80.92% |

| | |
|---|---|
| LSTM層數 | 1 |
| LSTM層節點數 | 89 |
| 全連接層層數 | 2 |
| 全連接層節點數 | 118 |
| dropout比例 | 0.4696 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 86 |
| accuracy | 80.80% |

| | |
|---|---|
| LSTM層數 | 2 |
| LSTM層節點數 | 129 |
| 全連接層層數 | 3 |
| 全連接層節點數 | 250 |
| dropout比例 | 0.4184 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 83 |
| accuracy | 80.19% |

# Select Hyperparameter

| | |
|---|---|
| LSTM層數 | 2 |
| LSTM層節點數 | 226 |
| 全連接層層數 | 2 |
| 全連接層節點數 | 204 |
| dropout比例 | 0.6982 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 83 |
| accuracy | 79.85% |

| | |
|---|---|
| LSTM層數 | 2 |
| LSTM層節點數 | 191 |
| 全連接層層數 | 3 |
| 全連接層節點數 | 198 |
| dropout比例 | 0.7053 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 34 |
| accuracy | 79.96% |

| | |
|---|---|
| LSTM層數 | ***2*** |
| LSTM層節點數 | ***129*** |
| 全連接層層數 | ***1*** |
| 全連接層節點數 | ***68*** |
| dropout比例 | ***0.4184*** |
| 激勵函數選擇 | ***softmax*** |
| 訓練批量 | ***71*** |
| accuracy | ***82.83%*** |

| | |
|---|---|
| LSTM層數 | 3 |
| LSTM層節點數 | 238 |
| 全連接層層數 | 1 |
| 全連接層節點數 | 80 |
| dropout比例 | 0.4184 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 48 |
| accuracy | 82.72% |

| | |
|---|---|
| LSTM層數 | 2 |
| LSTM層節點數 | 129 |
| 全連接層層數 | 1 |
| 全連接層節點數 | 127 |
| dropout比例 | 0.4353 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 71 |
| accuracy | 80.76% |

| | |
|---|---|
| LSTM層數 | 1 |
| LSTM層節點數 | 76 |
| 全連接層層數 | 1 |
| 全連接層節點數 | 80 |
| dropout比例 | 0.4184 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 48 |
| accuracy | 81.11% |

# Select Hyperparameter

| | |
|---|---|
| LSTM層數 | 1 |
| LSTM層節點數 | 129 |
| 全連接層層數 | 2 |
| 全連接層節點數 | 189 |
| dropout比例 | 0.4184 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 34 |
| accuracy | 80.27% |

| | |
|---|---|
| LSTM層數 | 2 |
| LSTM層節點數 | 94 |
| 全連接層層數 | 1 |
| 全連接層節點數 | 80 |
| dropout比例 | 0.6025 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 32 |
| accuracy | 81.41% |

| | |
|---|---|
| LSTM層數 | 1 |
| LSTM層節點數 | 190 |
| 全連接層層數 | 1 |
| 全連接層節點數 | 68 |
| dropout比例 | 0.4184 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 71 |
| accuracy | 80.23% |

| | |
|---|---|
| LSTM層數 | 3 |
| LSTM層節點數 | 216 |
| 全連接層層數 | 1 |
| 全連接層節點數 | 80 |
| dropout比例 | 0.7510 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 48 |
| accuracy | 81.41% |

| | |
|---|---|
| LSTM層數 | 1 |
| LSTM層節點數 | 129 |
| 全連接層層數 | 3 |
| 全連接層節點數 | 200 |
| dropout比例 | 0.4184 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 71 |
| accuracy | 81.76% |

| | |
|---|---|
| LSTM層數 | 2 |
| LSTM層節點數 | 221 |
| 全連接層層數 | 1 |
| 全連接層節點數 | 213 |
| dropout比例 | 0.4184 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 71 |
| accuracy | 80.19% |

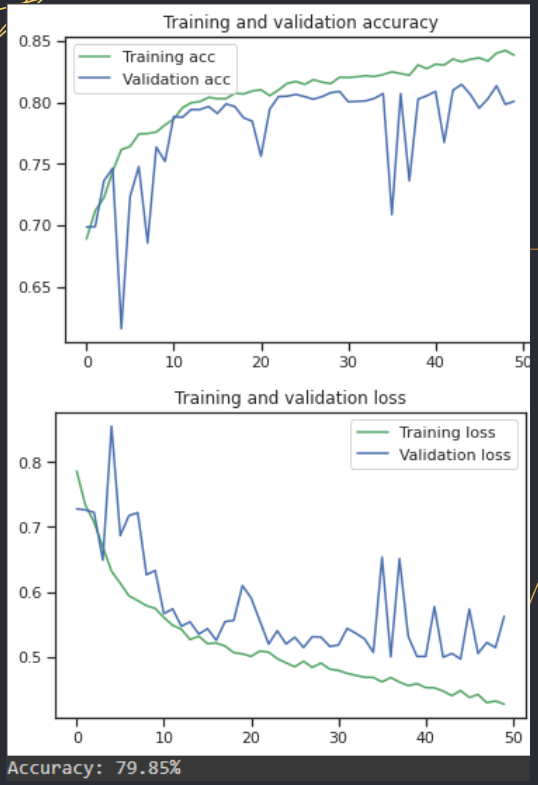
# Select Hyperparameter

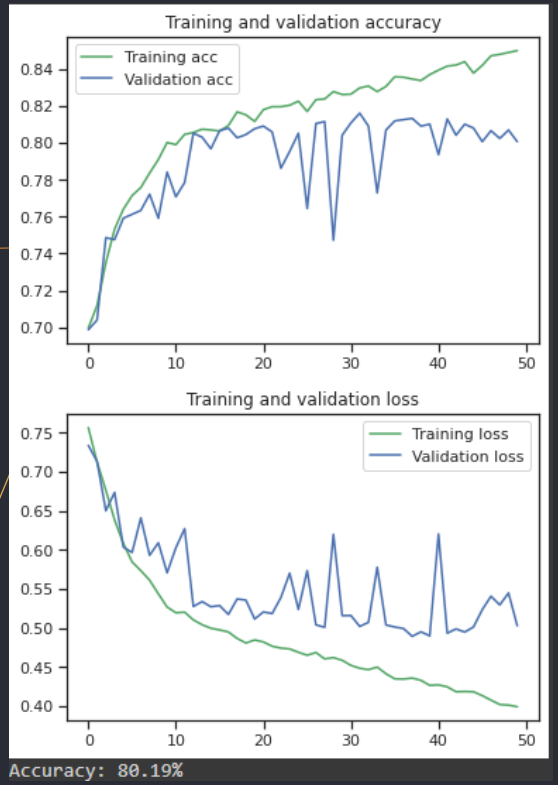
| | |
|---|---|
| LSTM層數 | 2 |
| LSTM層節點數 | 129 |
| 全連接層層數 | 1 |
| 全連接層節點數 | 241 |
| dropout比例 | 0.7973 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 71 |
| accuracy | 79.92% |

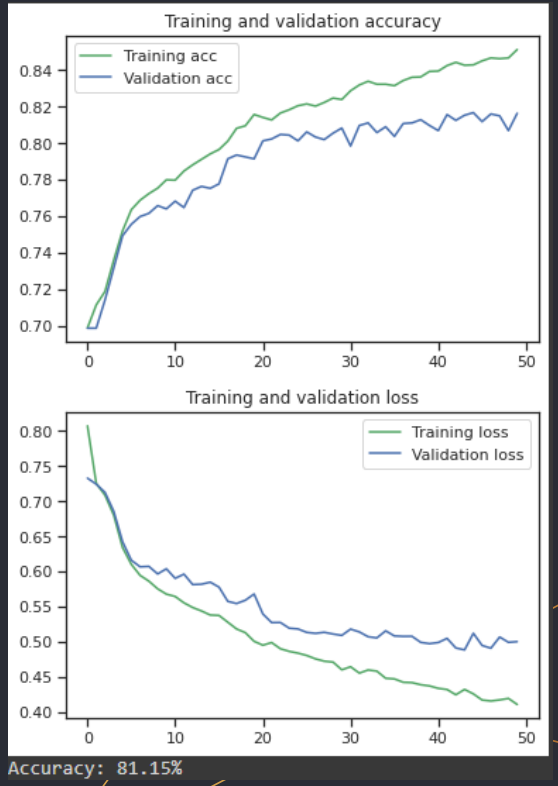
| | |
|---|---|
| LSTM層數 | 2 |
| LSTM層節點數 | 64 |
| 全連接層層數 | 1 |
| 全連接層節點數 | 213 |
| dropout比例 | 0.4184 |
| 激勵函數選擇 | softmax |
| 訓練批量 | 71 |
| accuracy | 81.15% |

| | |
|---|---|
| LSTM層數 | ***2*** |
| LSTM層節點數 | ***129*** |
| 全連接層層數 | ***1*** |
| 全連接層節點數 | ***68*** |
| dropout比例 | ***0.4184*** |
| 激勵函數選擇 | ***softmax*** |
| 訓練批量 | ***71*** |
| accuracy | ***82.83%*** |

# Selected Process

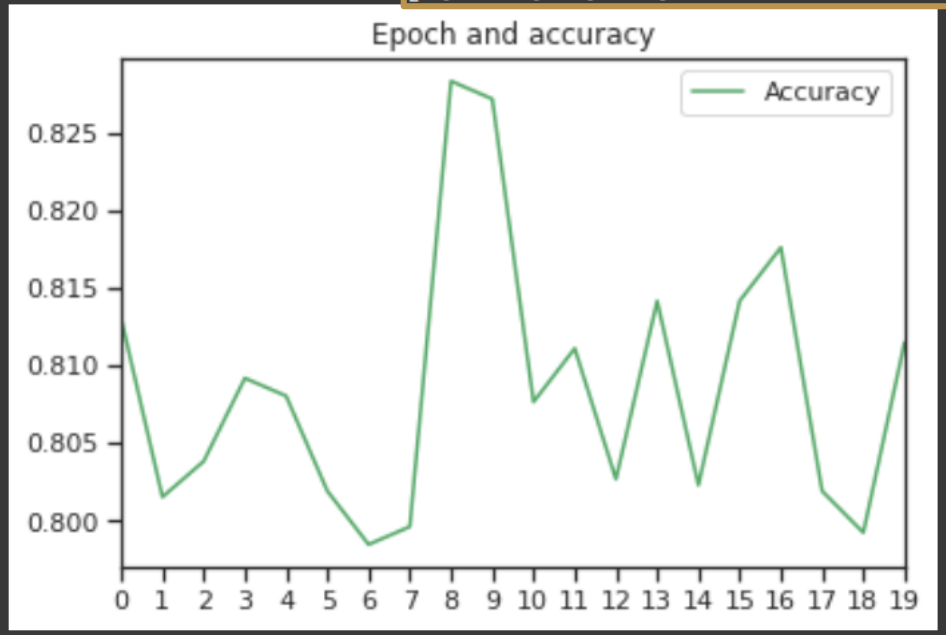


# Final Hyperparameter

```
============================== Final_Result ==============================
Global best fitness: 82.83%
Global best solution: [2, 129, 1, 68, 0.4183799784755163, 'softmax', 71]
```



# Training Model

```
# model = LSTM_model(best_dict['gbest_sol'])
model = ltsm_model((maxLen,), word_to_vec_map, word_to_index, best_dict['gbest_sol'])
model.compile(loss='categorical_crossentropy', optimizer=adadelta , metrics=['accuracy'])
model.fit(X_train_indices, y=Y_oh_train, batch_size=best_dict['gbest_sol'][6],
                    epochs=100, verbose=1, validation_data=(X_vaild_indices, Y_oh_vaild))
# Final evaluation of the model
scores = model.evaluate(X_test_indices, Y_oh_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Epoch 97/100
71/71 [==============================] - 1s 9ms/step - loss: 0.2797 - accuracy: 0.9002 - val_loss: 0.6347 - val_accuracy: 0.7887
Epoch 98/100
71/71 [==============================] - 1s 9ms/step - loss: 0.2816 - accuracy: 0.8984 - val_loss: 0.5315 - val_accuracy: 0.8143
Epoch 99/100
71/71 [==============================] - 1s 9ms/step - loss: 0.2748 - accuracy: 0.8962 - val_loss: 0.5056 - val_accuracy: 0.8235
Epoch 100/100
71/71 [==============================] - 1s 10ms/step - loss: 0.2730 - accuracy: 0.9056 - val_loss: 0.5341 - val_accuracy: 0.8231
Accuracy: 82.98%
```

# 6

# RESULT & CONCLUSION

結論與未來展望

# Define Inference Model

```
def TestModel(stringList):
        classess = ['negative','neutral','positive']
        results=[]
        if len(stringList) == 1:
                x_test = np.array(stringList)
                X_test_indices = sentences_to_indices(x_test, word_to_index, maxLen)
                index = np.argmax(model.predict(X_test_indices))
                return x_test[0] +' is a '+ classess[index]+' word'
        for i,x in enumerate(stringList):
                x = [x]
                x_test = np.array(x)
                X_test_indices = sentences_to_indices(x_test, word_to_index, maxLen)
                index = np.argmax(model.predict(X_test_indices))
                results.append(x_test[0] +' is a '+ classess[index]+' word')
        return results
```

# Inference Result

```
TestModel(['Nice  AirPlance  Service  I  love  Them  ',  #  Positive
                    'bad  AirPlance',  #  negative
                    'What  The  Fuckk  Plane'  ,  #  negative
                    'my  Flight  is  so  late  and  its  about  1  hour  why  ?',  #  negative
                    'okay  its  fine  flight  ,  thank  you  ',#  Postive
                    'I  lost  my  Bag  and  and  They  Back  it  to  me  So  Thanks  ',  #  Negative
                    'okay'  #  neutral
                  ])
```

```
['Nice AirPlance Service I love Them  is a positive word',
 'bad AirPlance is a negative word',
 'What The Fuckk Plane is a neutral word',
 'my Flight is so late and its about 1 hour why ? is a negative word',
 'okay its fine flight , thank you  is a positive word',
 'I lost my Bag and and They Back it to me So Thanks  is a negative word',
 'okay is a negative word']
```

# Future Direction

## 新增資料

添加更多複雜度較高的文句，以提升模型彈性。

## 增加變數

希望未來可以增加啟發式演算法的變數來彈性調整更多超參數之組合。

## 擴展應用

加入至機器人，使機器人在辨識完正/負面評價後可以做出相對的反應。

# THANKS!

DO YOU HAVE ANY QUESTION?