Projet 07 :

Détectez les Bad Buzz grâce au Deep Learning

Mohamed A.

OPENCLASSROOMS

TEAM

Vous êtes ingénieur IA chez MIC (Marketing Intelligence Consulting), une entreprise de conseil spécialisée sur les problématiques de marketing digital.

CLIENT

Air Paradis

Air Paradis a missionné votre cabinet pour créer un produit IA permettant d’anticiper les bad buzz sur les réseaux sociaux. Il est vrai que “Air Paradis” n’a pas toujours bonne presse sur les réseaux…

Air Paradis veut un prototype d’un produit IA permettant de **prédire le sentiment associé à un tweet**.

Données : pas de données clients chez Air Paradis. Solution : utiliser des données Open Source

TO-DO :

- Préparer un prototype fonctionnel du modèle. Le modèle envoie un tweet et récupère la prédiction de sentiment.
- Préparer un support de présentation explicitant la méthodologie utilisée pour l’approche “modèle sur mesure avancé” (attention : audience non technique).

Après avoir reçu votre compte-rendu, Marc, votre manager, vous a contacté pour, selon ses mots, “faire d’une pierre deux coups”.

De : Marc

Envoyé : hier 17:14

À : vous

Objet : Air Paradis : complément

Salut

Merci pour ton récap du meeting avec Air Paradis. J'ai l'impression que ça s'est bien passé ?!

Je me disais... Puisque tu vas faire un proto pour ce client, j'ai l'intuition que ce produit pourrait se généraliser à d'autres cas d'usage.

Tu voudrais bien en profiter pour tester plusieurs approches ?

- approche "Modèle sur mesure simple", pour développer rapidement un modèle classique (ex : régression logistique) permettant de prédire le sentiment associé à un tweet.
- approche "Modèle sur mesure avancé" pour développer un modèle basé sur des réseaux de neurones profonds pour prédire le sentiment associé à un tweet. => C'est ce modèle que tu devras déployer et montrer à Air Paradis.

-- Pour cette approche, tu penseras bien à essayer au moins deux word embeddings différents et à garder celui qui permet d'obtenir les meilleures performances. En complément, pourrais-tu également regarder l'apport en performance d'un modèle BERT ? Cela nous permettra de voir si nous devons investir dans ce type de modèle.
Je suis sûr que ça ne te prendra pas beaucoup plus de temps...

Merci d'avance !

Marc
PS : Ah au fait, tant que tu y es, tu pourras rédiger un petit article pour le blog à partir de ton travail ?

Livrables

1-Le "Modèle sur mesure avancé", **exposé grâce à un déploiement d'une API sur un service Cloud** (par exemple Azure, Heroku, Pythonanywhere qui offrent des solutions gratuites, ou toute autre solution)**,** qui recevra en entrée un tweet et retournera le sentiment associé au tweet prédit par le modèle.

-Ce livrable permettra d'illustrer votre travail auprès du client.

2-L'ensemble des scripts pour réaliser les trois approches (classique, modèle sur mesure avancé, modèle avancé BERT).

Ce livrable vous servira à présenter les détails de votre travail à une audience technique (par exemple des lecteurs de votre post de blog qui voudraient en savoir plus).

3-Un **article de blog** de 800-1000 mots environ contenant une présentation et une comparaison des trois approches ("Modèle sur mesure simple" et "Modèle sur mesure avancé", "Modèle avancé BERT") :

Ce livrable vous servira à faire rayonner le cabinet en démontrant votre expertise technique, mais aussi à valoriser votre travail auprès de la communauté des data scientists en ligne. Et surtout, à répondre aux exigences de votre manager !

4-Un **support de présentation** (type PowerPoint) de votre démarche méthodologique, des résultats des différents modèles élaborés et de la mise en production d'un modèle avancé.

Approach N 01 :	Machine Learning models

1. Text cleaning
2. Text stemming and Lemmatization
3. Text vectorizing
4. Data Splitting for test and train
5. Define and Run ML models :
   5-1 LogisticRegression
   5-2 SVM
6. Prediction on other data

Approach N 01 :

1. Text Cleaning
2. Text stemming and Lemmatization

Load Modules

Load Dataframe :

- Remove null tweets from dataset
- Splitting text into train & test sets
- Transforming Text into Numerical Feature Vectors
- Training the Model method :
  - 1 - Logistic regression
  - 2 - SVM
- Evaluation of the Model :
- Predictions on New Data

```
corpus_stem=[]

for i in range(0,16000):
     #clean Usertags '@user'
    review = re.sub('\B@\w+',"",df['tweet_'].iloc[i]) # \B to specify non-word boundary
    review = re.sub('(http|https):\/\/\S+','',review)#twisk slashes is mandatory ,\S capt
    review = re.sub('RT\s+',"",review) # \s : token character for white space / '+' is ad
    #clean emoji
    review = emoji.demojize(review) # replace emoji by it's name
    review = re.sub('[^a-zA-Z]',' ',review)

    review = review.lower()
    review = review.split()
    review = [ps.stem(word) for word in review if not word in set(all_stopwords)]
    review = ' '.join(review)
    corpus_stem.append(review)
```

```
# Define Lemmatizer
lemmatizer = WordNetLemmatizer()
```

```
# Define stemmer
ps = PorterStemmer()
all_stopwords = stopwords.words('english')
all_stopwords.remove('not')
```

<u>Results of text cleaning</u> : we can discuss some examples of stemming and Lemmatization

	rank	tweet_	tw_stem	tw_lemm
0	0	@Superpaperlink @treesmurf11 Oh that's just an...	oh annoy guess use dsi specif stuff like app shop	oh annoying guess used dsi specific stuff like...
1	0	I miss you guys soo much	miss guy soo much	miss guy soo much
2	0	@emilyphillips i use to leave them in my trunk...	use leav trunk time	use leave trunk time
3	0	Fuck, an old man just coughed in my hair!	fuck old man cough hair	fuck old man coughed hair
4	0	@bethoneil that sucks you should close the of...	suck close offic earli	suck close office early
5	0	Dealing with Vodafone is a nightmare	deal vodafon nightmar	dealing vodafone nightmare
6	0	It's hard to cry when the tears won't fall down	hard cri tear fall	hard cry tear fall
7	0	@honeybfly215 your missing all the festivities...	miss festiv sad not	missing festivity sad not
8	0	Weekends over - Back To Schoooolio for Anoth...	weekend back schoooolio anoth week	weekend back schoooolio another week

Approach N 01 :

3. Text Vectorizing :
4. Data Splitting for test and train
5. Define ML models :

Method used for vectorization :

We have chosen Lemmatized text for the rest of analysis according to its better quality of text's treatment.

```
from sklearn.feature_extraction.text import CountVectorizer
#By default, the vectorizer might be created as follows:
vectorizer = CountVectorizer()
#vectorizer = CountVectorizer(tokenizer = spacy_tokenizer, ngram
vectorizer.fit(reviews_train)

CountVectorizer()

X_train = vectorizer.transform(reviews_train)
X_test = vectorizer.transform(reviews_test)
```

Split dataset using sklearn : Ratio of test size = 0.2

```
from sklearn.model_selection import train_test_split
reviews = df['tw_lemm'].values
labels = df['rank'].values
reviews_train, reviews_test, y_train, y_test = train_test_split(reviews, labels, test_size=0.2, random_state=1000)
```

Define ML models :

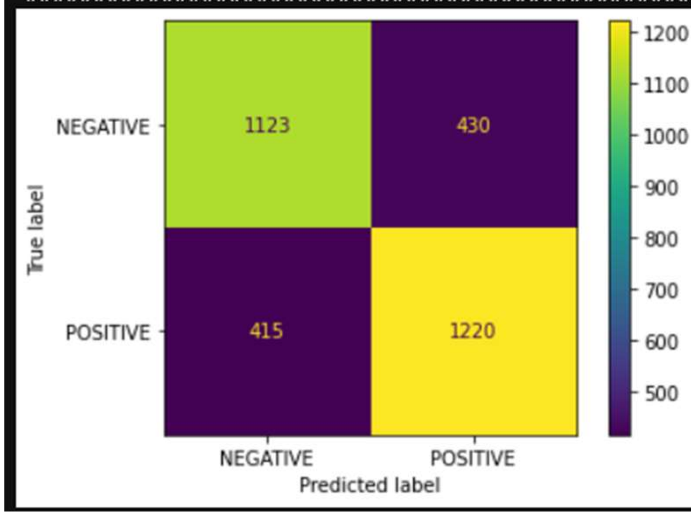
```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import classification_report

lr = LogisticRegression(max_iter=1000)
sv = SVC()
```

Approach N 01 :

Results of :
5-1 LogisticRegression
5-2 SVM

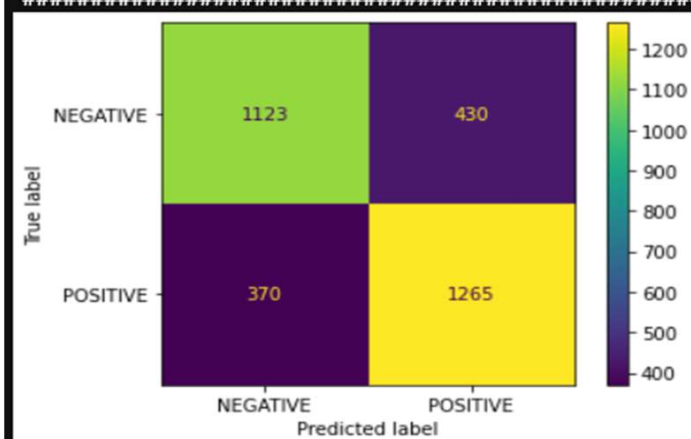
```
################################################
####### Logistic Regression ####################
################################################
Classification on train  :              precision    recall  f1-score   support

          0       0.90      0.90      0.90      6412
          1       0.90      0.90      0.90      6336

    accuracy                          0.90     12748
   macro avg       0.90      0.90      0.90     12748
weighted avg       0.90      0.90      0.90     12748

************************************************************************
Classification on test   :              precision    recall  f1-score   support

          0       0.73      0.72      0.73      1553
          1       0.74      0.75      0.74      1635

    accuracy                          0.73      3188
   macro avg       0.73      0.73      0.73      3188
weighted avg       0.73      0.73      0.73      3188
```

```
confusion matrix   :
#####################################################
```




```
################################################
####### SVM                ####################
################################################
              precision    recall  f1-score   support

           0       0.75      0.72      0.74      1553
           1       0.75      0.77      0.76      1635

    accuracy                           0.75      3188
   macro avg       0.75      0.75      0.75      3188
weighted avg       0.75      0.75      0.75      3188
```

```
confusion matrix   :
#####################################################
```



Approach N 01 :

Hyperparameter Tuning :
6-1 LogisticRegression
6-2 SVM

Logistic_regression

```
model = LogisticRegression()
solvers = ['newton-cg', 'lbfgs', 'liblinear']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]
# define grid search
grid = dict(solver=solvers,penalty=penalty,C=c_values)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
```

```
accuracy score _ on train   : 0.8079698776278632

accuracy score _ on test   : 0.7415307402760352
```

SVM

```
kernel = ['rbf', 'sigmoid']
C = [50,10, 1.0, 0.1]
#kernel = ['rbf', 'sigmoid']
#C = [10,0.01]
gamma = ['scale']
# define grid search
grid = dict(kernel=kernel,C=C,gamma=gamma)
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)
```

```
accuracy score _ on train   : 0.9390492626294321

accuracy score _ on test   : 0.7490589711417817
```

Approach N 01 : 6. Prediction on other data

Predictions on New Data

```python
models = [lr , sv]
```

```python
new_reviews = ['i am happy','it is a pain in the butt', 'Very good effort, but not five stars',
               'not Clear and not concise','i am good for now','never ever come back again']
new_reviews_class = [1,0,0,0,1,0]

print('Real classification of new inputs   :',new_reviews_class)
print('-----------------------------------------------------------')
for model in models:
    X_new = vectorizer.transform(new_reviews)
    print('Predictions for model',model,'  :',model.predict(X_new))
    print('-----------------------------------------------------------')
```

```
Real classification of new inputs   : [1, 0, 0, 0, 1, 0]
-----------------------------------------------------------
Predictions for model LogisticRegression(max_iter=1000)   : [1 0 1 0 1 0]
-----------------------------------------------------------
Predictions for model SVC()   : [1 0 0 0 1 0]
-----------------------------------------------------------
```

Approach N 02 : Deep Learning models

1. Define Embedding Layer
   1.1 Glove Layer
   1.2 Word2Vec Layer
2. RNN model with Glove
   2.1 RNN model with added Dropout layers (overfitting)
2. RNN model with Word2Vec
3. CNN model with Glove
4. CNN model with Word2Vec
5. LSTM model with Glove
6. LSTM model with Word2Vec

Approach N 02 : Deep Learning models

1.1 Glove Layer :

GloVe, coinced from Global Vectors , is a model for distributed word representation. The model is an unsupervised learning algorithm for obtaining vector representations for words. This is achieved by mapping words into a meaningful space where the distance between words is related to semantic similarity.
Training is performed on aggregated global word-word co-occurrence statistics from a corpus , and the resulting representations showcase interesting linear substructures of the word vector space. It is developed as an open-source project at stanford and launched in 2014.

https://nlp.stanford.edu/projects/glove/

```
embeddings_dictionnary = dict()
glove_file = open('/content/drive/MyDrive/P_7/glove.6B.300d.txt',encoding='utf8')

for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = asarray(records[1:],dtype='float32')
    embeddings_dictionnary[word] = vector_dimensions
glove_file.close()
```

```
# Create Embedding Matrix having 100 columns
# Containing 100-Dimensionnal Glove word embeddings for all words in our corpus
embedding_dim = 300  # Dimension provided by Glove 300 uploaded
skipped_words_g =0

embedding_matrix_glove = zeros((vocab_length,embedding_dim))
print('Embedding Matrix shape :',embedding_matrix_glove.shape)

for word, index in word_tokenizer.word_index.items():
    try:
        embedding_vector_glove = embeddings_dictionnary.get(word)
    except:
        skipped_words_g =skipped_words_g+1
        pass
    if embedding_vector_glove is not None :
        embedding_matrix_glove[index]=embedding_vector_glove

print('Count of skipped words  :',skipped_words_g)
```

```
Embedding Matrix shape : (13955, 300)
Count of skipped words  : 0
```

Approach N 02 : Deep Learning models

1.1 Word2Vec Layer :

Word2Vec is a technique for natural language processing which uses a neural network model to learn word associations from a large corpus of text. Once trained , such a model can detect synonymous words or suggest additional words for a partial sentence. As the name implies, word2Vec represents each distict word with a particular list of numbers called a vector chosen carefully such that a simple mathematical function(cosine similarity)indicates the level of similarity between the words represented by those vectors.

```
# Import pre-trained word2vec google file from drive
wordembeddings = gensim.models.KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin')
```

The model contains 300-dimensionnal vectors for 3 million words and phrases.

```
embedding_dim = 300  # Dimension provided by word2vec
skipped_words_w =0
word_skipped=[]

embedding_matrix_2vec = zeros((vocab_length,embedding_dim))
print('Embedding Matrix shape :',embedding_matrix_2vec.shape)

for word, index in word_tokenizer.word_index.items():

    try:
        embedding_vector_2vec = wordembeddings[word]
        word_skipped.append[word]
    except:
        skipped_words_w=skipped_words_w+1
        pass
    if embedding_vector_2vec is not None :
        embedding_matrix_2vec[index]=embedding_vector_2vec

print('Count of skipped words  :',skipped_words_w)
```

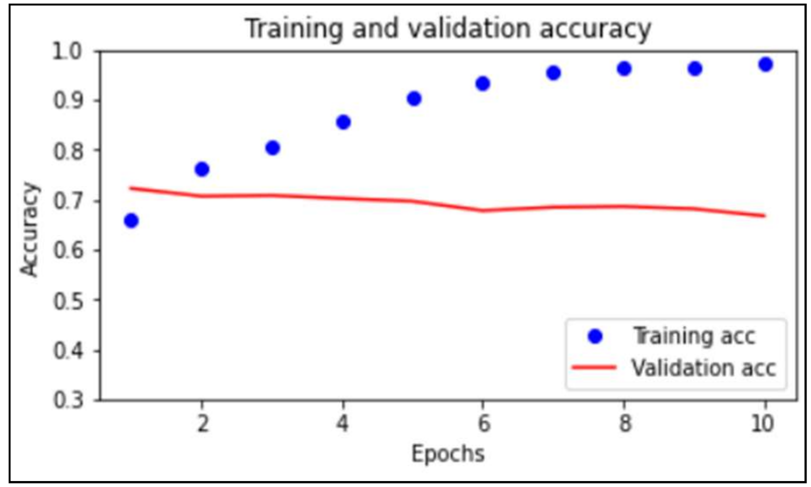
```
Embedding Matrix shape : (13955, 300)
Count of skipped words  : 13954
```

Approach N 02 : Deep Learning models

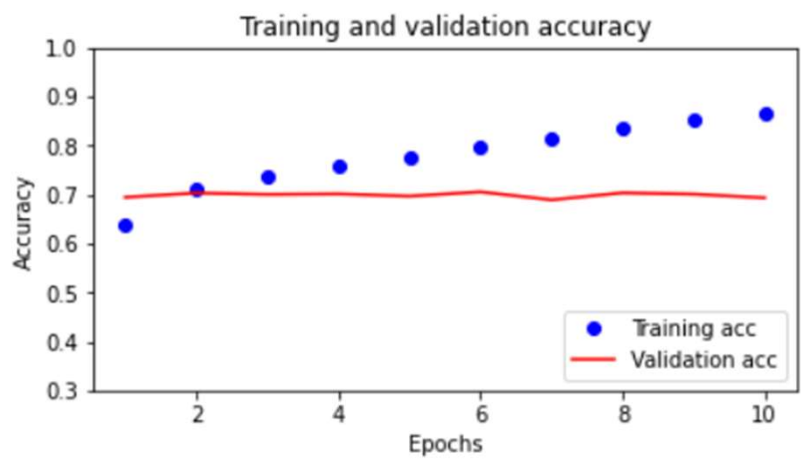
1.1 RNN model :

RNN sequential model with 'GloVe' embedding Layer :

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 23, 300)           4186500

 simple_rnn (SimpleRNN)      (None, 23, 128)           54912

 simple_rnn_1 (SimpleRNN)    (None, 23, 256)           98560

 flatten (Flatten)           (None, 5888)              0

 dense (Dense)               (None, 1)                 5889

=================================================================
Total params: 4,345,861
Trainable params: 159,361
Non-trainable params: 4,186,500
_________________________________________________________________
```



RNN sequential model with 'GloVe' embedding Layer and Dropout layers to adress 'overfitting' :

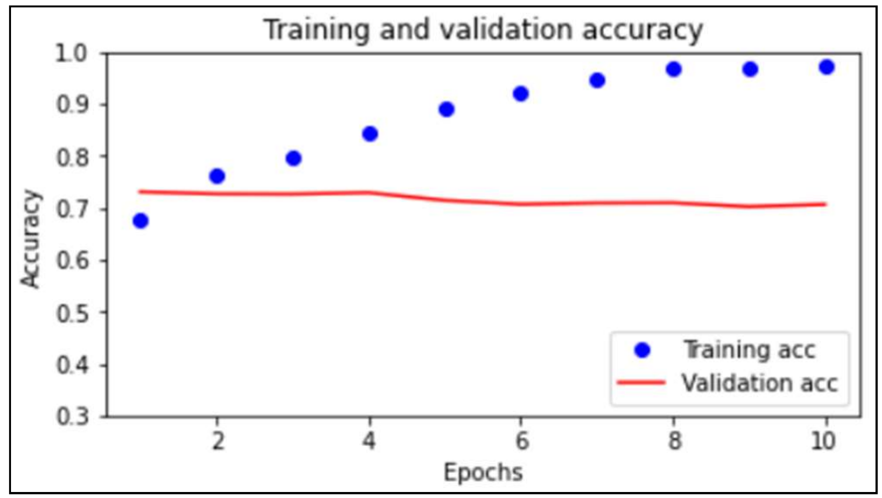


Approach N 02 : Deep Learning models

1.1 RNN model :

RNN sequential model with ‘Wor2Vec’ embedding Layer :

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 23, 300)           4186500

 simple_rnn (SimpleRNN)      (None, 23, 128)           54912

 simple_rnn_1 (SimpleRNN)    (None, 23, 256)           98560

 flatten (Flatten)           (None, 5888)              0

 dense (Dense)               (None, 1)                 5889

=================================================================
Total params: 4,345,861
Trainable params: 159,361
Non-trainable params: 4,186,500
_________________________________________________________________
```



RNN sequential model with both ‘GloVe’ or Word2Vec seems to overfit during training phase.

We can add some dropout layers or modify some hyperparameters as ‘kernel’ to reduce the overfitting state.

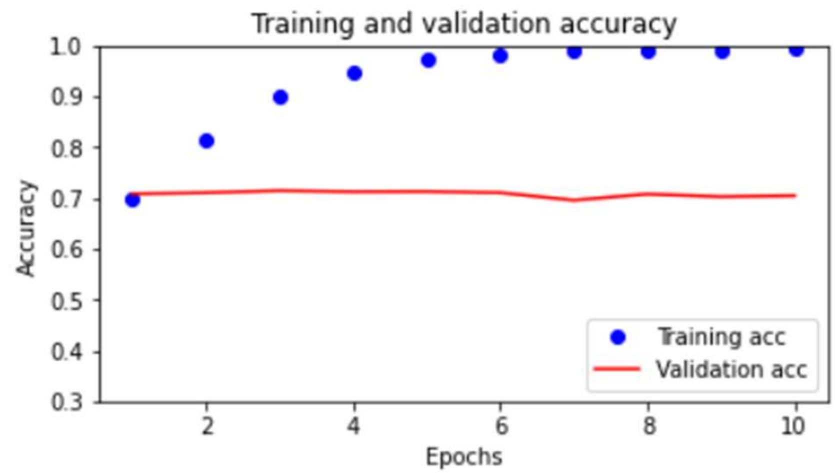
All accuracies for validation text seems to be similar and don not go beyond 0.8 value.

Approach N 02 : Deep Learning models

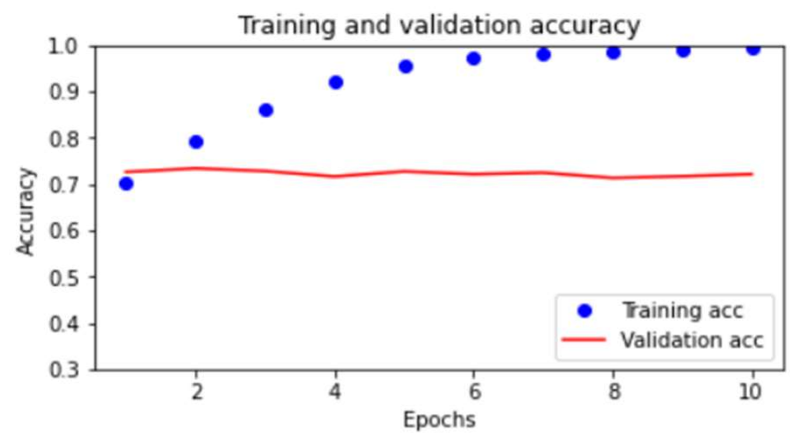
1.1 CNN model :

CNN Convolutional model with ‘GloVe’ embedding Layer :

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 23, 300)           4186500

conv1d (Conv1D)              (None, 19, 128)           192128

global_max_pooling1d (Globa  (None, 128)               0
lMaxPooling1D)

dense_3 (Dense)              (None, 1)                 129

=================================================================
Total params: 4,378,757
Trainable params: 192,257
Non-trainable params: 4,186,500
```



CNN Convolutional model with Word2vec embedding Layer :

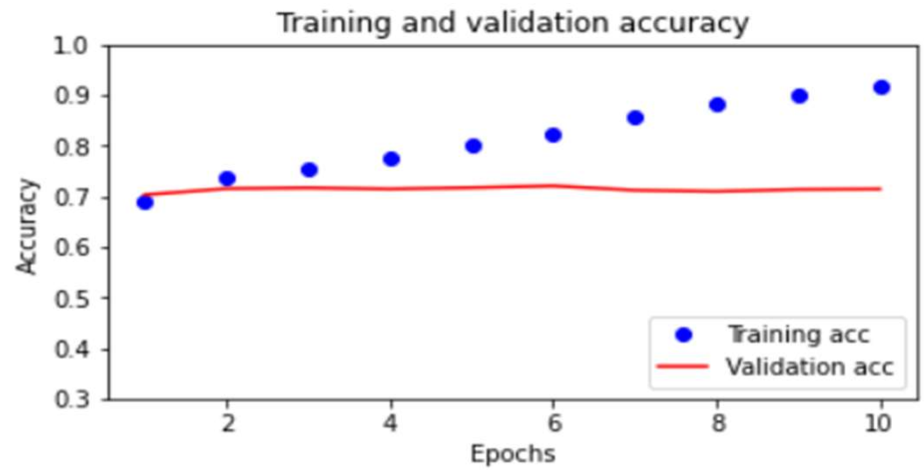


Approach N 02 : Deep Learning models

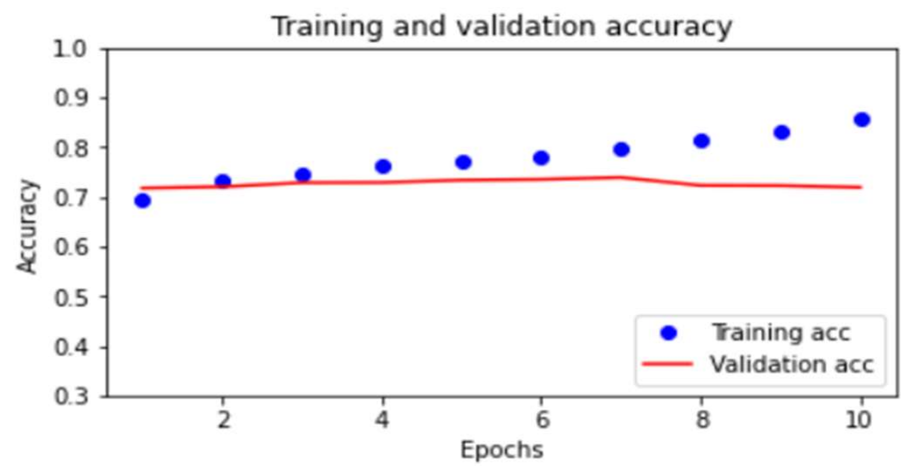
6 LSTM model :

LSTM model with ‘GloVe’ embedding Layer :

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding (Embedding)        (None, 23, 300)           4186500

 lstm (LSTM)                  (None, 128)               219648

 dense_5 (Dense)              (None, 1)                 129

=================================================================
Total params: 4,406,277
Trainable params: 219,777
Non-trainable params: 4,186,500
_________________________________________________________________
```



LSTM model with ‘Word2vec’ embedding Layer :



Approach N 03: Transformers

Bert Finetuning :

Model and Tokenizer used for Bert :

```
# Name of the BERT model to use
model_name = 'bert-base-uncased'

# Max length of tokens
max_length = 45

# Load transformers config and set output_hidden_states to False
config = BertConfig.from_pretrained(model_name)
config.output_hidden_states = False

# Load BERT tokenizer
tokenizer = BertTokenizerFast.from_pretrained(pretrained_model_name_or_path = model_name, config = config)

# Load the Transformers BERT model
transformer_bert_model = TFBertModel.from_pretrained(model_name, config = config)
```

Use of Tokenizer :

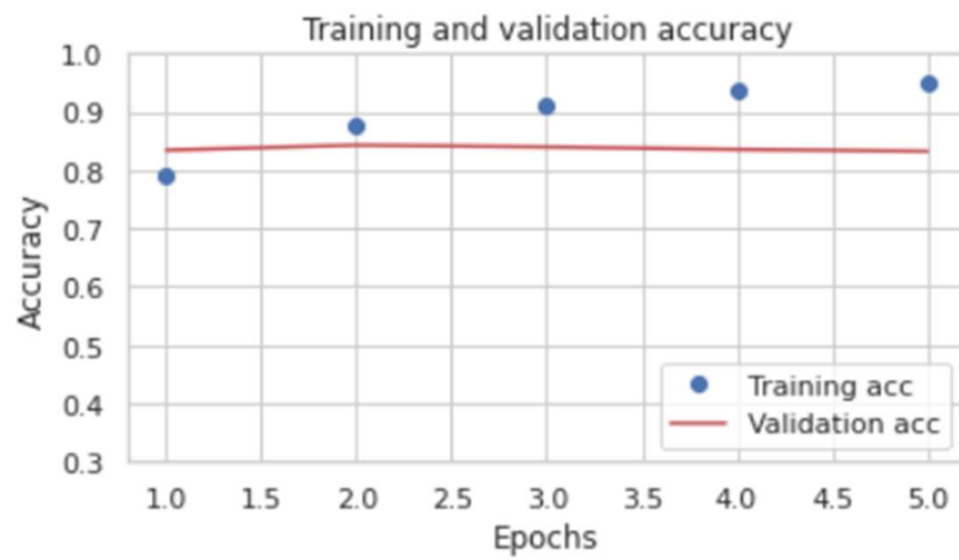
```
x_train = tokenizer(
          text=data_train['Phrase'].to_list(),
          add_special_tokens=True,
          max_length=max_length,
          truncation=True,
          padding=True,
          return_tensors='tf',
          return_token_type_ids = False,
          return_attention_mask = True,
          verbose = True)
```

Approach N 03: Transformers

Bert Finetuning :

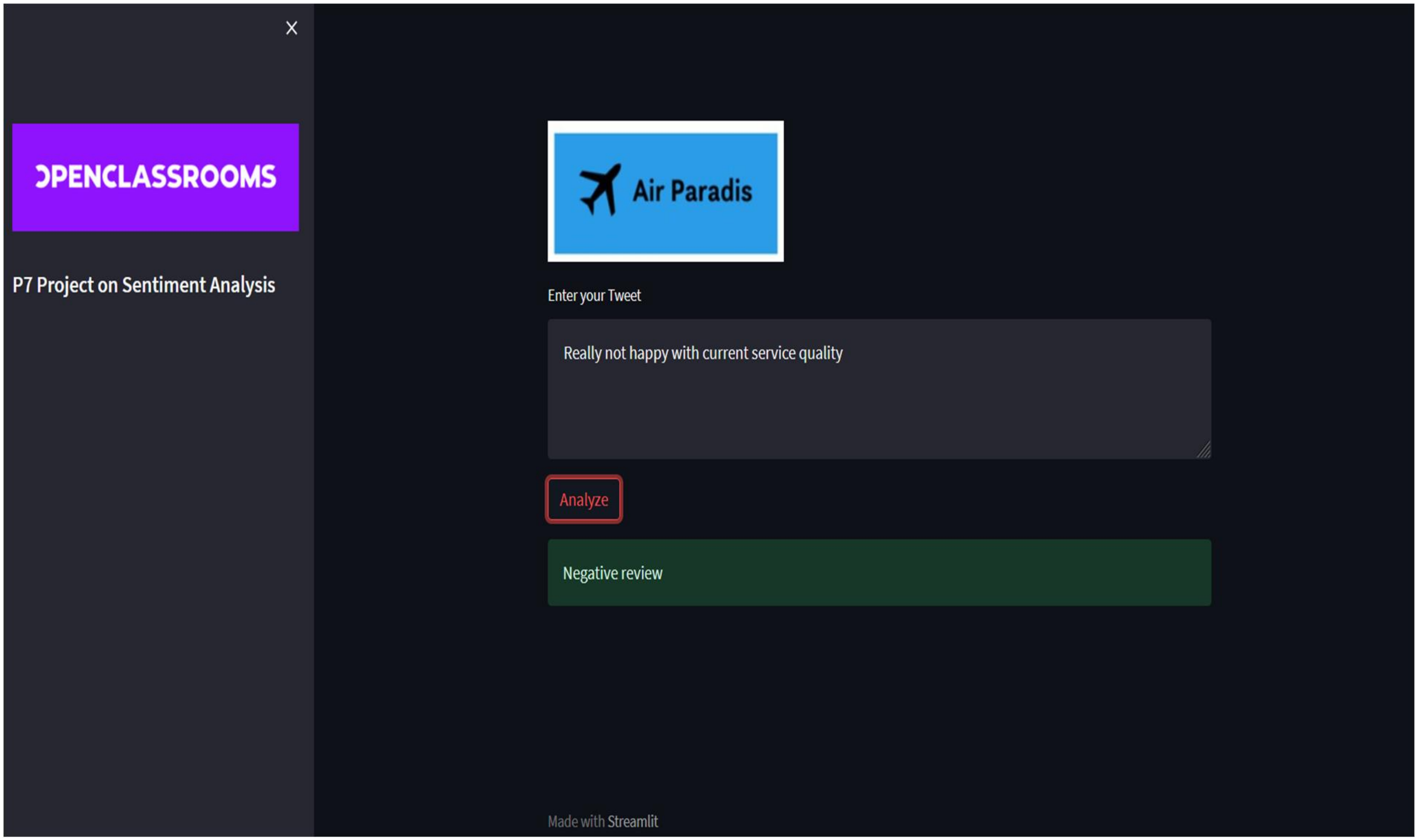
Results of Bert Fine tuning :

The bert model seems to be the best model according to the accuracy scores for both training , validation and test sets.



Application Deployed on streamlit :

We can see result on personnal tweet put in the app



Application Deployed on streamlit :