THEME : MINI PROJET NLP

ANALYSE ET CLASSIFICATION DES SENTIMENTS TWEETER

Implémentation du modèle dans une application mobile Android

Deep Learning python et application android (Flutter)

Réalisé par :

AHMED-AMINE GUIIDAT

ZAHR-EDDINE EL BOUZIDI

Encadré par le professeur :

Mr.Abdelhak MAHMOUDI

Année universitaire : 2020-2021

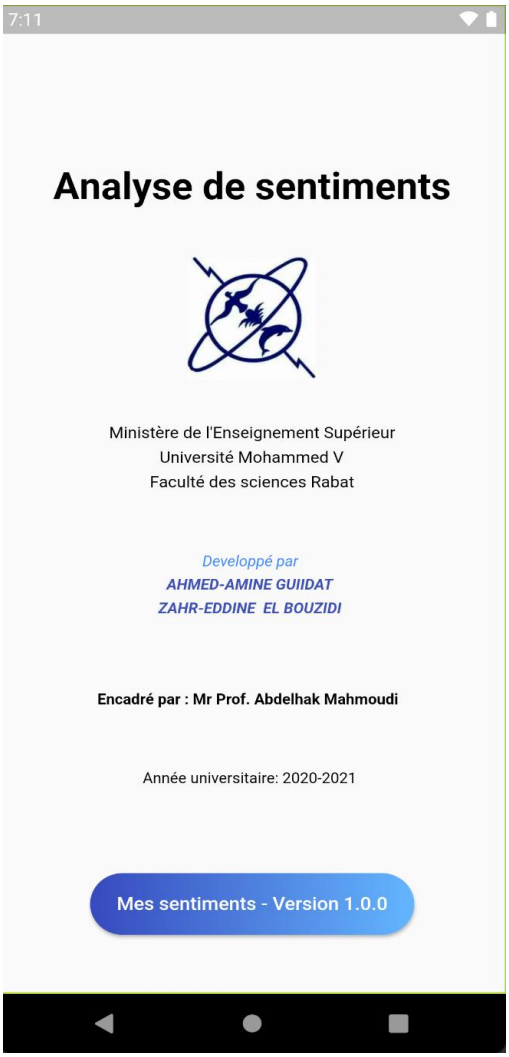


Table des matières

Introduction

L'objectif de notre projet est de réaliser une analyse exploratoire et visuelle des tweets en langage français présents dans notre jeu de données.

Dans un second temps, le but sera de parvenir à classifier à l'aide de modèles artificiels disponibles en Python, les sentiments des tweets selon qu'ils soient plutôt positifs ou négatifs. Autrement dit réunir **sentiment analysis** et NLP **en Deep neural network**.

Les prédictions de ce modèle NN sera appliquer dans une application mobile android afin de mélanger entre le deep learning et l'ingénierie des applications mobile

Nous allons commencer avec le nettoyage des tweets sous forme de texte a l'aide de preprocessing , les entrer dans un modèle **LSTM** classificateur et l'activation avec la fonction **sigmoid** .

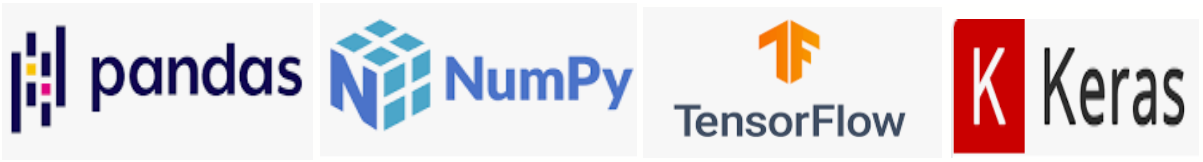
Ce classificateur produire des prédictions sur la classe (Positive ou Négative) à laquelle appartient le tweet.

Environnement et Librairies

Machine Learning

Google Colab Research https://colab.research.google.com/

- Langage Python 3
- NumPy
- Pandas
- Os
- String
- Re
- collections
- Struct
- Matplotlib
- Tensorflow
- Keras



Ingénierie – Application Mobile

- Android studio
- Flutter
- Langage Dart

1. Dataset – Jeu de données

Tout d'abord, dataset des sentiments en français est accessible via le site web de compétitions KAGGLE : **https://www.kaggle.com/hbaflast/french-twitter-sentiment-analysis**

2.Présentation du processus d'apprentissage automatique

2.1. Ingénierie des fonctionnalités

Avant de construire toute sorte de modèles prédictifs, nous avons non seulement besoin de données, mais nous avons besoin d'une représentation utilisable. Entrez dans l'ingénierie des fonctionnalités.

Avant l'ingénierie des fonctionnalités, nous nettoyons toujours nos données - ce qui peut consister à supprimer les valeurs symboles (ponctuations), à supprimer les données d'entrée non pertinentes selon la logique métier, à supprimer les urls, à supprimer les stops words.

2.2. Chargement de données -Sentiments Tweets

Voici un premier aperçu du jeu de données obtenu à l'aide de la commande **df.sample(5 , random_state = 0).**

```
[5]  df.sample(5 , random_state= 0)
```

	label	text
1222334	1	Une autre belle journée
656853	0	Hana c'était une bonne nuit. ça me manque
160403	0	Oui, malheureusement, j'ai seulement 10 minute...
913120	1	C'est comme, 1 am et amp; J'ai envie de manger...
695768	0	Ma cheville me reproche encore une fois

Figure 1:jeu de données aléatoires

Comme nous pouvons l'apercevoir, ce jeu de données recensant au total **1.5 millions** de tweets en langue français, contient une colonne intitulée **label** attribuant un 1 si le sentiment du tweet est positif et 0 dans le cas contraire.

On a presque de **755120** de tweets sont négative et **771604** de tweets avec un sentiment positive.

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:4
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7fda6fb08e90>
```

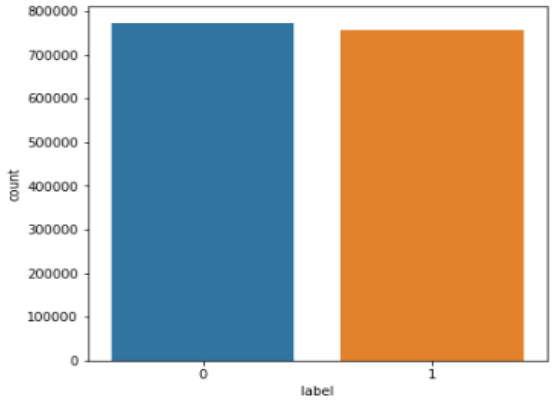


Figure 2: Nombre de tweets par sentiment

2.3. Preprocessing de données

2.3.1. Nettoyage de données (Punctuations, Urls, Stopwords,)

Comme j'ai dit dans le chapitre d'ingénierie des fonctionnalités, Pour analyser le sentiment d'un tweet, nous pouvons alors supprimer ce qui nous semble être inutile comme par exemple : les ponctuations, les urls, les stops words etc…

Class Preprocessing data

```
[ ]  #my preprocesing class
     class Preprocessing_data:

       def __init__(self , df):
         self.df = df
         self.all_punctuations = '''`÷×¬“”!|+¦~{}',.؟":/←][%^&*()_<>¦''' +string.punctuation

       #remove punctuations
       def remove_punctuations(self , text):
         text = text.lower()
         my_clean_sentence = ''.join([item for item in text if item not in self.all_punctuations])
         return my_clean_sentence

       #remove urls
       def remove_urls(self , text):
         text = text.lower()
         url_clean = re.compile(r"https?://(\S+|www)\.\S+")
         return url_clean.sub(r"", text)

       # define all french stopwords
       def define_all_french_stopwords(self):
         all_french_stop_words=set(fr_stop)
         deselect_stop_words = ['n\'', 'ne','pas','plus','personne','aucun','ni','aucune','rien']
         for w in deselect_stop_words:
           if w in all_french_stop_words:
             all_french_stop_words.remove(w)
           else:
             continue
         return all_french_stop_words
```

Figure 3: class de preprocessing partie 1

```
def preprocessing_data(self , df):
  my_stopwords = self.define_all_french_stopwords()
  #remove urls
  df['text'] = df['text'].apply(lambda x : self.remove_urls(x))
  #remove all punctuations
  df['text'] = df['text'].apply(lambda x : self.remove_punctuations(x))
  arrayOfWordsTweets=[]
  filter_text_array=[]
  for tweet in df["text"].apply(str):
    array_of_words = []
    array_of_words = [word for word in re.sub("\W"," ",tweet).split()]
    arrayOfWordsTweets.append(array_of_words)
  if 'words_tweet' in df.columns:
    del df['words_tweet']
  if 'filtered_tweet_text' in df.columns:
      del df['filtered_tweet_text']
  df['words_tweet'] = arrayOfWordsTweets
  for tweet in df["words_tweet"]:
    filteredText = [w for w in tweet if not ((w in my_stopwords) or (len(w) == 1))]
    filter_text_array.append(' '.join(filteredText))
  df['filtered_tweet_text'] = filter_text_array
  return df
```

Figure 4: classe de preprocessing partie 2

2.3.2. Générer un nouveau data frame nettoyé

Ci-dessous un nouveau data frame qui donne une idée sur le texte tweet après le nettoyage (Preprocessing data) **filtered_tweet_text**

```
final_df = pd.DataFrame(my_filtered_df , columns=['label','text', 'filtered_tweet_text'])
final_df.sample(5 , random_state = 1)
```

	label	text	filtered_tweet_text
156683	0	je pense que je devrais embaucher un de ces tr...	devrais embaucher tranlateurs personne ne jamais
730038	0	et la pauvre ruth vous voulez des tissus	pauvre ruth voulez tissus
985311	1	a dessiné une carte pour vous 6 baguettes c...	dessiné carte baguettes connu seigneur victoir...
403385	0	mcfly 7 juin 2009 au mexique la grippe porci...	mcfly juin 2009 mexique grippe porcine détruit...
1195641	1	bon lhummer sort des rues il faut que les voi...	bon lhummer sort rues faut voitures déplacent ...

Figure 5: jeu de données après le preprocessing

3. Préparation d'un model réseau de neurone artificiel

3.1. LSTM (Long Short-Term Memory) RNN amélioré

Ce type de RNN est très utilisé en traitement du langage naturel. L'idée derrière ce choix d'architecture de réseaux de neurones est de diviser le signal entre ce qui est important à court terme à travers le **hidden state** (analogue à la sortie d'une cellule de RNN simple), et ce qui l'est à long terme, à travers le **cell state**, qui sera explicité plus bas.

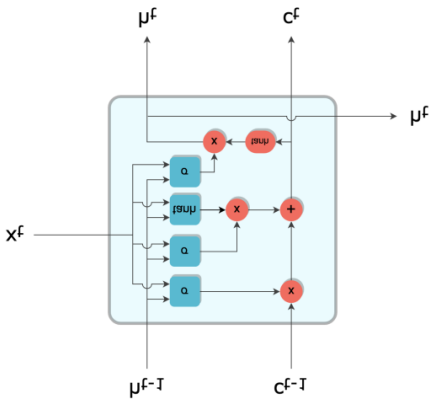


Figure 6: LSTM architecture

Comme le RNN, le LSTM définit donc une relation de récurrence, mais utilise une variable supplémentaire qui est le cell state c :

$$h_t , c_t = f(x_t, h_{t-1}, c_{t-1})$$

La **forget gate** est une couche dense de taille **R** avec une activation **sigmoïde**. À partir de **ht-1** et **xt**, cette forget gate produit un vecteur **ft** de taille **R**, dont les valeurs sont comprises entre **0** et **1.**

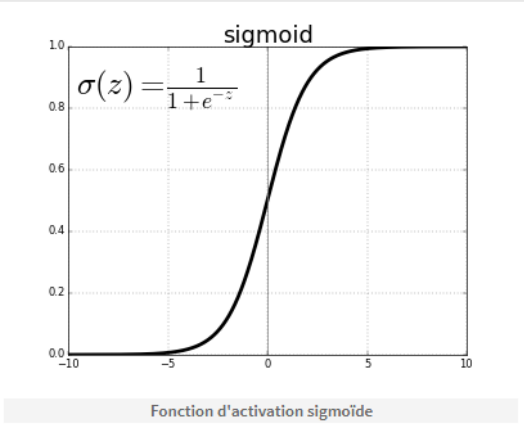


Figure 7: Fonction d'activation sigmoïde

3.2. Créer un model de classification (Tensorflow et Keras)

3.2.1. Préparer, Entraîner et Tester le modèle LSTM avec Tensorflow 2.5

La classe ci-dessus montre la procédure complète de création de notre réseau de neurone récurent améliorée avec LSTM.

1. En regroupant ce nouveau jeu de données en deux parties : la première partie pour l'entraînement de données (80%) et la deuxième partie est égale à 20%.
2. Tokenizing notre jeu de données afin d'extraire la séquence d'entraînement, la séquence de validation et les indices des mots uniques.
3. Créer notre modèle de classification LSTM.
4. Entraîner et tester notre modèle.

```python
class BuildModel:

  def __init__(self , df):
    self.df = df

    #counts unique words
  def counts_words(self , df):
    counter = Counter()
    for text in df['filtered_tweet_text'].values:
      for word in text.split():
        counter[word]+=1
    return len(counter)

  def split_dataset(self , df):

    training_size = int(df.shape[0] * 0.80) #80% training set
    training_df = df[:training_size] # same shape 80%
    validation_df = df[training_size:] #rest of training df
    #split label and text into numpy
    training_texts = training_df.filtered_tweet_text.to_numpy()
    training_labels = training_df.label.to_numpy()
    validation_texts = validation_df.filtered_tweet_text.to_numpy()
    validation_labels = validation_df.label.to_numpy()
    return training_df,validation_df,training_texts,training_labels,validation_texts,validation_labels
```

Figure 8: Préparation de modèle

```python
def tokenizing_data(self , df):

  training_texts = self.split_dataset(df)[2]
  validation_texts = self.split_dataset(df)[4]
  num_unique_words_of_tweet = self.counts_words(df)
  tokenizer = Tokenizer(num_words = num_unique_words_of_tweet)
  tokenizer.fit_on_texts(training_texts)
  word_indexes = tokenizer.word_index
  training_sequences = tokenizer.texts_to_sequences(training_texts)
  validation_sequences = tokenizer.texts_to_sequences(validation_texts)
  return word_indexes , training_sequences , validation_sequences

def padded_vector(self , df):

  training_sequences = self.tokenizing_data(df)[1]
  validation_sequences = self.tokenizing_data(df)[2]
  #specify the max length of words in one sequence
  max_length = 20
  training_pad = pad_sequences(training_sequences , maxlen = max_length , padding = "post" , truncating="post")
  validation_pad = pad_sequences(validation_sequences , maxlen = max_length , padding = "post" , truncating = "post")
  return training_pad , validation_pad


def buildNeuralNetworkLSTMModel(self, df):
  #get unique words from the dataframe
  num_unique_words_of_tweet = self.counts_words(df)
  max_length = 20
  #create LSTM Model
  model = keras.models.Sequential()
  model.add(layers.Embedding(num_unique_words_of_tweet, 32 , input_length=max_length ))
  model.add(layers.LSTM(64 , dropout= 0.1))
  model.add(layers.Dense(1 , activation="sigmoid"))
  model.summary()
  return model
```

Figure 9: Suite de prép modèle

```python
def trainingNeuralNetworkLSTMModel(self ,model , df , epochs_numbers = 10):
  #get Model
  #model = self.buildNeuralNetworkLSTMModel(df)
  #get training pad vector
  training_pad = self.padded_vector(df)[0]
  #get validation pad vector
  validation_pad = self.padded_vector(df)[1]
  #get validation labels
  validation_labels = self.split_dataset(df)[5]
  #get training labels
  training_labels = self.split_dataset(df)[3]
  #begin training model with 10 epochs by default
  loss = keras.losses.BinaryCrossentropy(from_logits=False)
  myoptimizer = keras.optimizers.Adam(learning_rate=0.001)
  metrics = ['accuracy']
  model.compile(optimizer=myoptimizer , loss = loss,metrics= metrics)
  model.fit(training_pad , training_labels , epochs= epochs_numbers,validation_data=(validation_pad , validation_labels) , verbose= 2)
  return model
```

Figure 10:Suite de pré modèle

Entraînement de modèle :

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 20, 32)            9365472
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                24832
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 9,390,369
Trainable params: 9,390,369
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
38169/38169 - 486s - loss: 0.4394 - accuracy: 0.7965 - val_loss: 0.6688 - val_accuracy: 0.6774
Epoch 2/10
38169/38169 - 482s - loss: 0.3852 - accuracy: 0.8291 - val_loss: 0.6259 - val_accuracy: 0.6919
Epoch 3/10
38169/38169 - 482s - loss: 0.3407 - accuracy: 0.8530 - val_loss: 0.7358 - val_accuracy: 0.6534
Epoch 4/10
38169/38169 - 482s - loss: 0.3172 - accuracy: 0.8630 - val_loss: 0.6996 - val_accuracy: 0.6686
Epoch 5/10
38169/38169 - 482s - loss: 0.3009 - accuracy: 0.8704 - val_loss: 0.7087 - val_accuracy: 0.7020
Epoch 6/10
38169/38169 - 482s - loss: 0.2888 - accuracy: 0.8762 - val_loss: 0.7664 - val_accuracy: 0.6766
Epoch 7/10
38169/38169 - 482s - loss: 0.2790 - accuracy: 0.8811 - val_loss: 0.8242 - val_accuracy: 0.6561
Epoch 8/10
38169/38169 - 483s - loss: 0.2707 - accuracy: 0.8849 - val_loss: 0.7666 - val_accuracy: 0.6702
Epoch 9/10
38169/38169 - 482s - loss: 0.2639 - accuracy: 0.8882 - val_loss: 0.8229 - val_accuracy: 0.6680
Epoch 10/10
38169/38169 - 482s - loss: 0.2587 - accuracy: 0.8903 - val_loss: 0.8306 - val_accuracy: 0.6717
<keras.engine.sequential.Sequential at 0x7fcfcfa5f450>
```

Figure 11: Entraînement de modèle

3.2.2. Sauvegarder le modèle en H5

Save NN Model as H5 file (my_classification_model.h5)

```
[ ]  mymodel.save('my_classification_model.h5')
```

Figure 12: Sauvegarde de modèle H5

3.2.3. Prédiction du modèle en H5

```
# load model
model = load_model('/content/drive/MyDrive/deep_learning_french_tweet_dataset/my_classification_model.h5')
# summarize model.
model.summary()
tokenizer = Tokenizer(num_words = 292671)
sequence = tokenizer.texts_to_sequences(['je suis très content !'])
print(sequence)
test = pad_sequences(sequence, maxlen=20)
print(test)
resp = model.predict(test)
print(resp)
res = ["Positive" if p > 0.5 else "Négative" for p in resp]
print(res)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 20, 32)            9365472
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                24832
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 9,390,369
Trainable params: 9,390,369
Non-trainable params: 0
_________________________________________________________________
[[]]
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
[[0.62746894]]
['Positive']
```

Figure 13: Prediction

3.2.4. Sauvegarder un dictionnaire des mots

Save dictionary

```
[ ]  reverse_words_index = dict([(i,v) for (i , v) in word_indexes.items()])
     reverse_words_index
     with open('converted_labels_classifier.txt', 'w') as f:
         print(reverse_words_index, file=f)
```

Figure 14: Dictionnaire de mots

3.2.5. Convertir le modèle en Tensorflow Lite

Convert Tensorflow model to the TFLite Model for Mobile App

```
[ ]  converter = tf.lite.TFLiteConverter.from_keras_model(model)
     tflite_model = converter.convert()
     open("converted_classifier_model.tflite", "wb").write(tflite_model)
```

```
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_ca
INFO:tensorflow:Assets written to: /tmp/tmpi_t50ypw/assets
INFO:tensorflow:Assets written to: /tmp/tmpi_t50ypw/assets
37578348
```

Figure 15: Conversion H5 en TfLite

Pour la partie d'ingénierie l'implémentation de ce modèle dans une application mobile android, nous avons besoin de deux fichiers importants :

- Le premier c'est un fichier qui contient un ensemble de mots avec indexation sous forme de dictionnaire.
- Et le deuxième on a besoin de convertir le modèle **my_classification_model.h5** en TFLITE alors il faut faire une conversion de ce modèle NN.

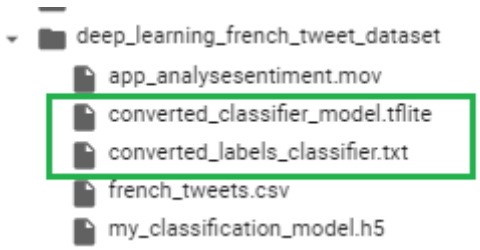


Figure 16: fichiers Dictionnaire et Modèle TfLite

5.Présentation après l'implémentation du modèle LSTM dans une application mobile

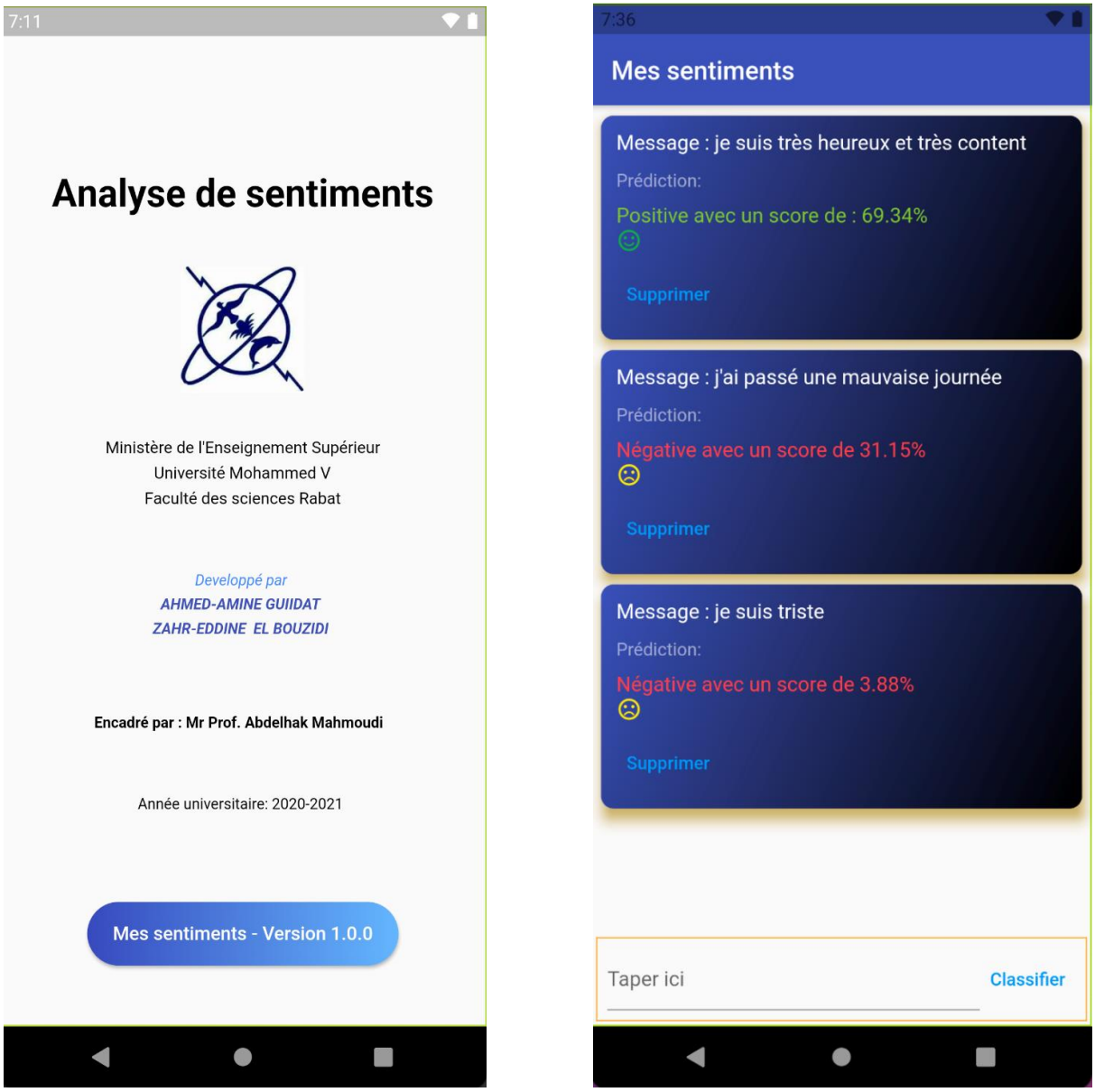


Figure 17: Page d'accueil application android

Test Sentiment Positive et Négative :

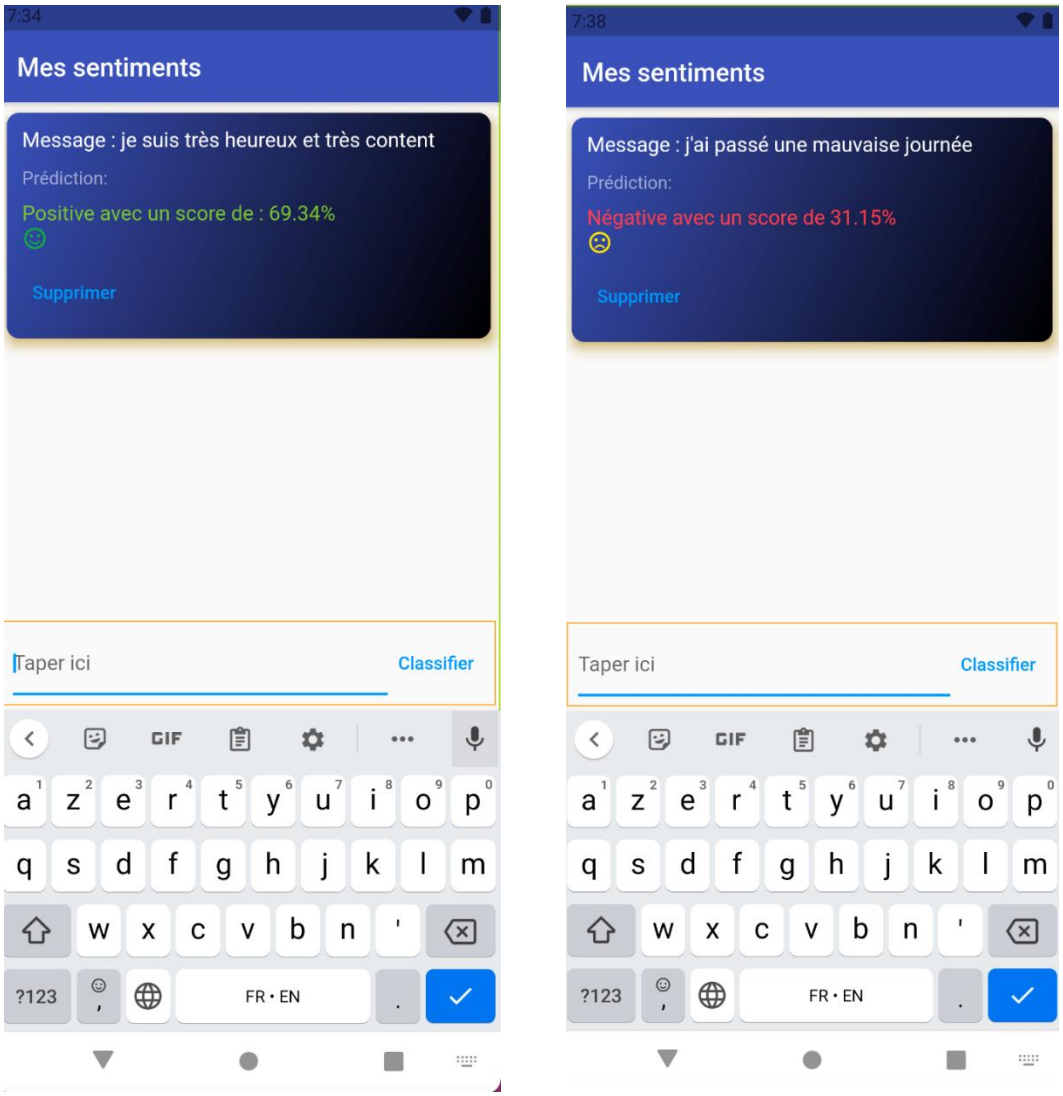


Figure 18: Sentiments positives et négatives

Merci de votre attention

Références :

- Support du cours de Mr Professeur Abdelhak MAHMOUDI - UM5- Faculté des sciences Rabat.
- https://ml4a.github.io/ml4a/fr/neural_networks/
- https://www.kaggle.com/hbaflast/french-twitter-sentiment-analysis
- https://www.tensorflow.org/text/guide/word_embeddings
- https://www.tensorflow.org/tutorials/keras/text_classification
- https://www.tensorflow.org/tutorials/keras/text_classification_with_hub
- https://www.tensorflow.org/tutorials/text/word2vec
- https://www.tensorflow.org/lite/tutorials/model_maker_text_classification
- https://www.tensorflow.org/lite/convert
- https://tel.archives-ouvertes.fr/tel-01771685/document
- https://pub.dev/packages/tflite_flutter
- https://flutter.dev/docs/development/ui/widgets-intro
- https://dart.dev/