# Rapport – UE : Extraction et programmation statistique de l'information

GUO Xuewen, NADO Ndieme

April 2, 2023

## 1 Introduction

Ce rapport a pour objectif de présenter le travail que nous avons réalisé dans le cadre du projet de l'UE 11. Le but du projet est de détecter la langue maternelle à partir de phrases écrites en anglais. Dans ce rapport, nous vous présenterons d'abord les technologies que nous avons utilisées, puis le prétraitement des données et les modèles de machine learning que nous avons réalisés.

## 2 Les technologies utilisées

Nous avons choisi d'implémenter ce projet en utilisant le langage de programmation Python, en raison de sa popularité et de ses nombreuses bibliothèques dédiées à la science des données. Nous avons notamment exploité les capacités de Scikit-Learn, TensorFlow et Keras pour le traitement des données et la création de modèles de machine learning. Afin de trouver le modèle optimal pour notre tâche de détection de la langue maternelle à partir de phrases en anglais, nous avons testé plusieurs autres modèles, tels que SVM, la régression logistique, etc. Nous avons expliqué chacun de ces modèles dans notre rapport, bien que nous n'ayons pas sélectionné tous ces modèles pour notre modèle final.

## 3 Preprocessing

Le preprocessing est une étape clé dans la programmation statistique. En effet, nous devons préparer et nettoyer les données afin de permettre aux modèles de les analyser.

L'ensemble de données comprend deux parties: l'étiquette et le contenu. L'étiquette contient la langue maternelle de la personne ayant écrit le texte (ARA, CHI, FRE, GER, HIN, ITA, JPN, KOR, SPA, TEL TUR). Le contenu correspond au texte écrit par la personne. Pour uniformiser les données et extraire des informations utiles pour entraîner les modèles, nous avons effectué plusieurs étapes de nettoyage, notamment :

- Convertir tous les caractères en minuscule : pour uniformiser les caractères et considérer a = A
- Supprimer les "stop words" : les mots inutiles pour de l'analyse de texte, Pour cela, nous avons utilisé la bibliothèque nltk
- Supprimer les liens : très rarement exploitable
- Supprimer tous les caractères spéciaux : cela n'est pas utile pour determiner la langue maternelle d'une personne
- Ignorer les tokens contenant uniquement un caractère (car cela ne nous permet pas d'avoir une analyse efficace)

  Cependant, lors de nos tests, nous avons remarqué que le score obtenu avec les données nettoyées était inférieur à celui obtenu avec les données brutes. Après étude de nos données, nous avons considéré que les textes étaient suffisamment propres et avons décidé de ne pas convertir tous les caractères en minuscules, car cela peut aussi représenter la langue maternelle de l'auteur.

## 3.1 Tokenisation

Le principe de la tokenisation est de rendre un texte en une série (ou une liste) de token individuel. Ce token réprensent un mot.

## 3.2 Vectorisation

Nous avons ensuite vectorisé nos textes afin de pouvoir entraîner les modèles. L'objectif est de vectoriser le texte pour représenter celui-ci sous forme de série de vecteur qui peuvent exprimer la sémantique du texte. Nous allons voir les deux techniques de vectorisation que nous avons utilisé : le Bag of Words et le Tf-Idf. Finalement, nous avons choisi Tf-Idf.

### 3.2.1 Bag of Words : CountVectorizer

Le Bag of Words ("Sac de mots" en français) est une méthode qui permet de compter le nombre d'occurrence de mots dans un texte. Cette technique ne permet pas de garder l'ordre des mots dans le texte. Elle permet principalement de savoir si deux textes sont simmilaires par leur contenu similaire. Avec CountVectorizer et SVM, nous avons obtenue :

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
vectorizer = CountVectorizer()
X_counts = vectorizer.fit_transform(df.doc)
X_train, X_test, y_train, y_test = train_test_split(X_counts, new_y, test_size=0.2,
svm = SVC(kernel='linear', random_state = 29)
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

✓ 1m 41.1s

```
0.6116161616161616
```

Avec CountVectorizer et logistic regression, nous avons obtenue :

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
vectorizer = TfidfVectorizer()
X_counts = vectorizer.fit_transform(df.doc)
X_train, X_test, y_train, y_test = train_test_split(X_counts, new_y, test_size=0.2, random_state=42)
lr = LogisticRegression()
bow_lr = lr.fit(X_train, y_train)
y_pred = bow_lr.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

✓ 8.8s

```
0.6787878787878788
```

### 3.2.2 TF-IDF Vectorizer

Le TF-IDF ou Term Frequency Inverse Document Frequency, est une méthode de vectorisation. Elle permet de donner, à chaque mot, un poids en fonction de son importance dans le texte tout en comptant

le nombre d'occurence. Voici la formule :

$$TfIdf = \frac{f_{t,d}}{log\frac{N}{n_t}}$$

Pour notre cas, nous mettons la fenêtre du N-Gram à (1,2).

Avec TF-IDF Vectorizer et SVM, nous avons obtenue :

```
# tf -idf with svm
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
vectorizer = TfidfVectorizer()
X_counts = vectorizer.fit_transform(df.doc)
X_train, X_test, y_train, y_test = train_test_split(X_counts, new_y, test_size=0.2, random_state=42)
svm = SVC(kernel='linear', random_state = 29)
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

✓ 2m 28.9s

```
0.7055555555555556
```

Avec TF-IDF Vectorizer et logistic regression, nous avons obtenue :

```
# tf -idf with lr
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
vectorizer = TfidfVectorizer()
X_counts = vectorizer.fit_transform(df.doc)
X_train, X_test, y_train, y_test = train_test_split(X_counts, new_y, test_size=0.2, random_state=42)
lr = LogisticRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

✓ 8.6s

```
0.6787878787878788
```

# 4 Modèle de machine learning

Dans un premier temps, nous avons testé plusieurs modèles de classification, tels que le Random Forest et le Bayésien naïf. La plupart de ces modèles avaient une précision moyenne d'environ 70 %, mais chacun avait ses propres forces et faiblesses en termes de capacité à prédire certaines classes.

Par la suite, nous avons essayé SVM et logistic par régression logistique, mais l'Average accuracy est autour de 63,68%.

Enfin, nous avons décidé d'essayer les réseaux de neurones, même si le jeu de données n'était pas suffisamment volumineux pour entraîner un réseau de neurones classique. En paramétrant les

différentes couches du modèle, nous avons réussi à obtenir une précision de 72 %.

## 4.1 Cross-Validation avec kfold k=5 ou 10

Nous avons ensuite testé les SVM et la régression logistique en utilisant une méthode de validation croisée k-fold (k = 5 ou 10) pour évaluer la performance de chaque modèle. Cependant, nous avons observé que cette technique de validation croisée réduit la précision de nos modèles. Sans l'utilisation de la cross-validation, nous avions un score moyen de 70 %, mais une précision moyenne de 63,68%. Après avoir utilisé cette technique, nous avons observé un résultat intéressant, étant donné la complexité de notre jeu de données. Par conséquent, nous avons commencé à spéculer sur les raisons de ce phénomène.

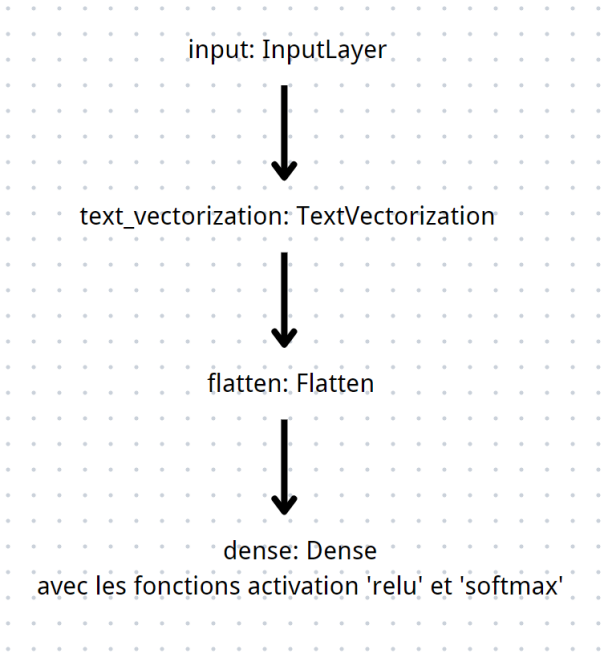
La validation croisée en k-folds consiste à diviser les données en k sous-ensembles, et à utiliser chaque sous-ensemble une fois pour la validation, tandis que les k-1 autres sous-ensembles sont utilisés pour l'entraînement. Cela permet d'obtenir k mesures de la performance du modèle, qui peuvent être agrégées pour obtenir une estimation de la performance moyenne.

Cependant, l'utilisation de la validation croisée en k-folds peut entraîner une baisse de précision pour plusieurs raisons :

- Perte de données d'entraînement : En divisant les données en k sous-ensembles, la taille de l'ensemble de données d'entraînement pour chaque pli est réduite, ce qui peut entraîner une perte importante de données d'entraînement. Une taille de pli trop petite peut entraîner une instabilité de l'estimation de la performance du modèle.
- Variabilité des performances : En utilisant des sous-ensembles différents pour l'entraînement et la validation à chaque itération, la performance estimée pour chaque pli peut être très variable. En conséquence, l'estimation de la performance moyenne peut être moins précise.
- Sensibilité aux paramètres : La performance du modèle peut être sensible aux paramètres utilisés pour la validation croisée, tels que le nombre de folds k ou la stratégie de répartition des données entre les folds. Une mauvaise sélection de ces paramètres peut entraîner une baisse de précision.

## 4.2 Réseaux de neurones

Nous avons construit un modèle simple de réseau de neurones en utilisant keras. Le schéma ci-dessous montre les différentes couches du modèle :



et avec notre code:

```python
def nlp_tfidf_classification(X,y_encoded):
    tfidf_vectorizer = TfidfVectorizer(max_features=10000)
    X_tfidf = tfidf_vectorizer.fit_transform(X)
    X_tfidf = X_tfidf.toarray()

    X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y_encoded, test_size=0.2, random_state=42)

    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(X_tfidf.shape[1],)),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(11, activation='softmax')
    ])

    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.1)

    y_pred = model.predict(X_test).argmax(axis=1)

    accuracy = accuracy_score(y_test, y_pred)
    print(f"Test accuracy: {accuracy:.4f}")

    return model,y_pred,y_test

model,y_pred,y_test = nlp_tfidf_classification(df["doc"],y)
```

Voici quelques explications sur les parametres choisis pour notre modele:

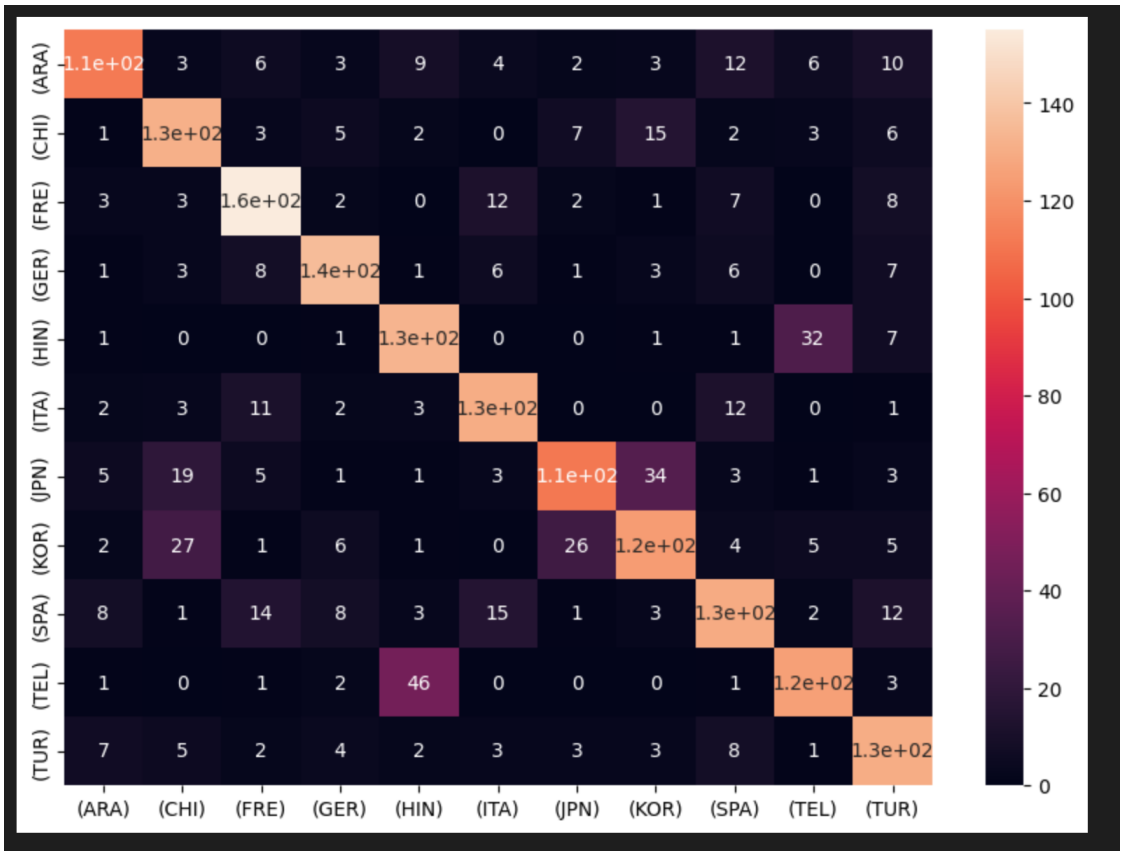
- La fonction d'activation 'relu' (Rectified Linear Unit) est utilisée dans les couches intermédiaires pour ajouter de la non-linéarité au modèle. Elle permet de transformer les valeurs négatives en 0 et de laisser les valeurs positives inchangées. Cette fonction d'activation est souvent utilisée dans les réseaux de neurones pour ses bonnes performances en termes d'apprentissage et de vitesse de calcul.
- La fonction d'activation 'softmax' est utilisée dans la dernière couche du modèle pour calculer la probabilité d'appartenance à chaque classe (langue maternelle). Elle permet de normaliser les sorties du modèle pour qu'elles soient comprises entre 0 et 1, et de garantir que leur somme soit égale à 1.
- L'optimiseur 'Adam' est utilisé pour minimiser la fonction de coût (loss). Cet optimiseur est très populaire dans l'apprentissage profond car il est efficace pour la convergence rapide du modèle et la gestion des gradients.
- La fonction de coût 'sparse categorical crossentropy' est utilisée pour calculer l'erreur entre les sorties du modèle et les labels de sortie (langue maternelle). Elle est adaptée aux problèmes de classification multiclasse et est particulièrement utile lorsque les classes ne sont pas équilibrées (c'est-à-dire lorsque les exemples ne sont pas répartis de manière égale entre toutes les classes). L'utilisation de 'sparse' signifie que les labels ne sont pas encodés en one-hot encoding, mais simplement en tant que labels entiers compris entre 0 et le nombre de classes - 1.

Nous avons obtenue une score:

```
Epoch 1/10
223/223 [==============================] - 4s 14ms/step - loss: 2.3175 - accuracy: 0.1667 - val_loss: 2.0225 - v
Epoch 2/10
223/223 [==============================] - 5s 23ms/step - loss: 1.7279 - accuracy: 0.3803 - val_loss: 1.3796 - v
Epoch 3/10
223/223 [==============================] - 2s 9ms/step - loss: 1.1922 - accuracy: 0.5787 - val_loss: 1.0634 - va
Epoch 4/10
223/223 [==============================] - 2s 9ms/step - loss: 0.8293 - accuracy: 0.7210 - val_loss: 0.9131 - va
Epoch 5/10
223/223 [==============================] - 2s 9ms/step - loss: 0.5999 - accuracy: 0.8084 - val_loss: 0.8523 - va
Epoch 6/10
223/223 [==============================] - 2s 9ms/step - loss: 0.4461 - accuracy: 0.8685 - val_loss: 0.8306 - va
Epoch 7/10
223/223 [==============================] - 2s 9ms/step - loss: 0.3323 - accuracy: 0.9039 - val_loss: 0.8257 - va
Epoch 8/10
223/223 [==============================] - 2s 9ms/step - loss: 0.2559 - accuracy: 0.9272 - val_loss: 0.8319 - va
Epoch 9/10
223/223 [==============================] - 2s 9ms/step - loss: 0.1981 - accuracy: 0.9508 - val_loss: 0.8517 - va
Epoch 10/10
223/223 [==============================] - 2s 9ms/step - loss: 0.1607 - accuracy: 0.9575 - val_loss: 0.8965 - va
62/62 [==============================] - 0s 1ms/step
Test accuracy: 0.7273
```

### 4.3 Modèle en cascade

Pour améliorer notre modele, nous avons affiché la matrice de confusion:




À l'aide de la matrice de confusion, nous avons constaté que les deux langues HIN et TEL étaient souvent confondues. Après avoir recherché des informations sur ces deux langues, nous avons découvert

qu'elles appartenaient à la même branche de langues, ce qui explique leur similitude.
Nous avons donc décidé de créer un deuxième modèle pour distinguer ces deux langues, et de l'utiliser en cascade avec le premier modèle de réseaux de neurones.
Après avoir testé plusieurs modèles tels que SVM et RNN, nous avons finalement choisi d'utiliser la vectorisation TF-IDF associée à une régression logistique, qui a obtenu de meilleurs résultats parmi les trois modèles testés.

```
    tfV = TfidfVectorizer(ngram_range=(2,2))
    tf_X = tfV.fit_transform(X)
    X_trainTF, X_testTF, y_trainTF, y_testTF = train_test_split(tf_X, y_HINTEL, random_state=4)

    from sklearn.linear_model import LogisticRegression
    lr = LogisticRegression()
    tf_lr = lr.fit(X_trainTF, y_trainTF)
    y_pred = tf_lr.predict(X_testTF)
    accuracy = accuracy_score(y_testTF, y_pred)
    print(accuracy)
]
```

```
0.7888888888888889
```

# 5 Résultat

Selon la modele en cascade, nous avons obtenue une score 0.78

# 6 Problèmes rencontrés

## 6.1 SVM avec kernel rbf

Nous avions essayé d'utiliser le SVM avec un noyau gaussien. Le résultat obtenu était de 0.08%. Cela peut peut-être s'expliquer par le fait qu'en utilisant le noyau RBF, le SVM devient un "approximateur universel" et ce cas n'est pas adapté à notre problème.

```
    from sklearn import svm

    modelSvm = svm.SVC(C=1.0, kernel='rbf',gamma='auto',
                       decision_function_shape='ovr',
                       cache_size=500)

    modelSvm.fit(tf_train,y_train)
68]
```

```
SVC
SVC(cache_size=500, gamma='auto')
```

```
    svmPre=modelSvm.predict(tf_test)
    accuracy(y_test,svmPre)
27]
```

```
Accuracy: 0.08606060606060606
```

### 6.2 Connexion des deux modèles

Un second problème rencontré est sur la cascade des modèles. En effet, nous avons pas pu connecter les outputs de premier modèle aux inputs du second. Ce qui nous n'a pas permit d'avoir le score total de ce modèle.

## 7 Conclusion

Ce projet a été une expérience très bénéfique pour notre équipe, car il nous a permis de mettre en pratique nos compétences en NLP et en machine learning dans un contexte réel. Nous avons appris de nombreuses méthodologies sur la mise en place de projets de machine learning, telles que l'exploration des données, le preprocessing et la création de modèles, ainsi que sur les technologies utilisées pour réaliser ces phases.

Au cours de ce projet, nous avons testé plusieurs modèles et pipelines, amélioré les résultats de nos prédictions et mis en place une approche en cascade pour une prédiction plus précise. Nous sommes également conscients que ces résultats pourraient être améliorés en utilisant des données plus volumineuses ou en explorant davantage de modèles.

En conclusion, ce projet nous a permis d'acquérir de nouvelles connaissances et compétences en machine learning et en NLP, tout en réalisant une tâche pratique intéressante. Nous sommes fiers des résultats que nous avons obtenus et sommes impatients d'appliquer ces compétences à des projets futurs.