StackOverflow Auto Tagging

Introduction

L'objectif du modèle est de prédire des tags associés à une question et un post afin d'automatiser la création de ceux-ci.

Ici le modèle reçoit en entrée un titre et un corp de texte et ressort les tags associés.

Architecture

Stockage

Les données sont récupérées via un CSV extrait de Stack overflow
Un tri est effectué afin de récupérer uniquement les colonne titre, post, et tags et le plus de ligne possible , ici 17 600 lignes.
Les données sont récupérées en CSV afin de les stocker en base de données relationnel à l'aide de Supabase pour mettre la base de données en ligne.
https://supabase.com/dashboard/project/bosbinvsnempbohyiwjy

La base de données est récupérée dans un notebook python afin d'appliquer les traitements suivants sur celui-ci.

Nettoyage

Les données sont nettoyées à l'aide de BeautifulSoup et StopWord pour retirer balises et stopword (he, him, she, etc…) puis nous utilisons une concaténation la colonne titre et post afin de récupérer une seule colonne contenant toute nos données.

Puis nous stemmatison la colonne afin de racinisé celle ci et éviter le surplus de données à cause des doublons pour mieux “groupé” les sens des mots entre eux et pouvoir créer des “relation”

```
def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext

def cleanpunc(sentence): #function to clean the word of any punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned

def stem(text: str):
    sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer which is developed in recent years
    stop=set(stopwords.words('english'))

    str1=' '
    final_string=[]
    s=''

    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(text) # remove HTMl tags
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                else:
                    continue
            else:
                continue
    #print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("**************************************************************")

    final_string.append(str1)

    return pd.Series(final_string[0].decode('utf8'))
```

Entrainement

Ensuite celui-ci est vectorisé à l'aide de la librairie SKLearn en important le vectorizer qui permet de transformer les mots en vecteur pour que notre modèle puisse le traiter.
Le jeu de données est splité à l'aide de Train Test Split (Train = 70% et Test = 30%)

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state = 42)
```

Après cela, la données est envoyée dans Transformer (TfidfTransformer).

Enfin est utilisé le modèle MultinomialNB (Naive Bayes Multinomial) afin de pouvoir prédire la probabilité qu’un tag apparaisse ou non et de classer le bon tag au bon post.

```
nb = Pipeline([('vect', CountVectorizer()),
               ('tfidf', TfidfTransformer()),
               ('clf', MultinomialNB()),
              ])
```

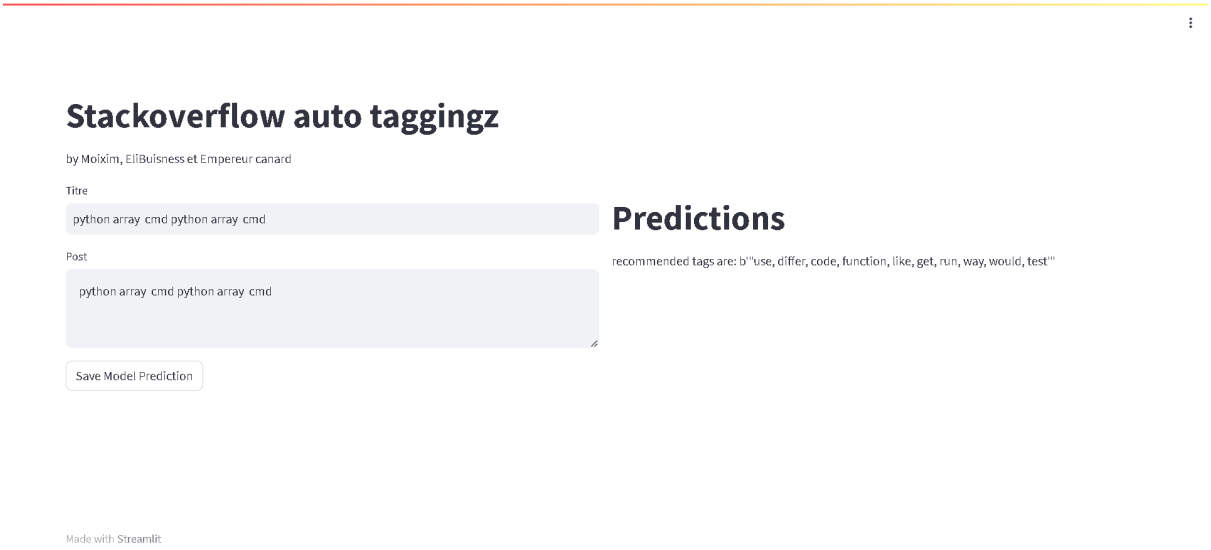
Les résultats observé sont une accuracy du F1 score de 47%

```
accuracy 0.46944549226706905
              precision    recall  f1-score   support

           0       0.95      0.26      0.40       696
           1       0.00      0.00      0.00       198
           2       0.91      0.03      0.05       369
           3       1.00      0.01      0.01       147
           4       0.00      0.00      0.00       211
           5       0.00      0.00      0.00       359
           6       0.00      0.00      0.00       109
           7       1.00      0.01      0.02       538
           8       1.00      0.02      0.03       381
           9       0.45      1.00      0.62      2294

    accuracy                           0.47      5302
   macro avg       0.53      0.13      0.11      5302
weighted avg       0.58      0.47      0.33      5302
```

Production

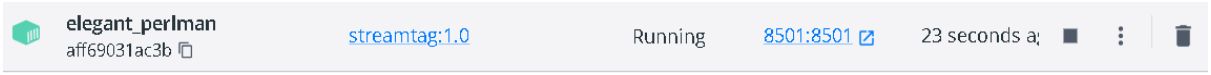
Interface

Pour l'interface utilisateur , un Streamlit a été déployé pour que l'utilisateur puissent interagir avec le modèle et faire des prédiction sur ces propre post



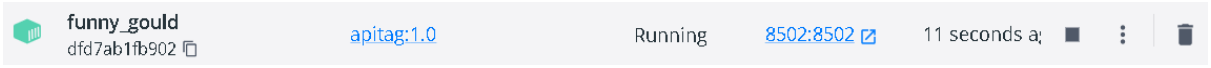
Déploiement

Afin de facilité de déploiement et de rendre notre modeles interoperable , la solution d'utiliser Docker a été retenu afin de conteneurisé le tout et qu'il puisse être lancé à partir de n'importe quelle machine



Monitoring

Pour pouvoir monitorer notre modèle la solution retenue est d'utiliser ML Flow afin de pouvoir comparer toute nos itération



Conclusion

Le modèle utilisé est un MultinomailNB avec une Accuracy de 47%.
La base de données est déployée à l'aide de superbase, l'interface et le modèle sont déployés à l'aide de l'hébergeur natif de Streamlit.

Le tout a été conteneurisé a l'aide de Docker, et le monitoring du modèle sera suivi à l'aide de ML Flow déployé sur FastAPI qui lui même est déployé à l'aide de ngrok.

Mode d’emploi

Afin d'exécuter correctement le modèle veuillez suivre les indications suivante:

- Déterminer les variables d'environnement sur la plateforme choisie :
  API_URL= ''

- Construire les images docker
  - docker build -f Dockerfile -t streamtag:1.0 .
  - docker build -f Dockerfile_api -t apitag:1.0 .

- Envoyer les images à la plateforme souhaité