

Feature Extraction and Embeddings

Data preparation

We load and preprocess the dataset that we got from the previous tp, encoding the *author* column using *One-hot Encoding* and get new column named *author_encoded*

```
...
```

```
label_encoder = LabelEncoder()
df['author_encoded'] = label_encoder.fit_transform(df['author'])
...
```

Construction of training/test set

We split the dataset into training and test sets (30% for the test) using *train_test_split* and since the dataset is imbalanced, we use the *stratify* parameter to ensure that the distribution of classes in training/test is similar

```
...
```

```
x_train, x_test, y_train, y_test = train_test_split(df['text'], df['author_encoded'], test_size=0.3,
random_state=0, stratify = df['author_encoded'].values)
...
```

we get

```
...
```

	train set	test set
0	5529	2369
2	4230	1813
1	3944	1691

```
...
```

```
train_set = test_set * 2.33
```

Vectorization methods

we use two vectorization methods: *frequency-based* from *CountVectorizer* and *TF-IDF* (*Term Frequency-Inverse Document Frequency*) in order to convert raw text data into numerical resresentation that machine learning models can process, *frequency-based* will split text into individual words/tokens, count the frequency of each token in the document and generate a matrix where each row refers to a document, each column refers to unique token in the corpus and the values represented are the frequency of each token in that document, for *TF-IDF* used to evaluate the importance of token is the document relative to the curpos with

$TF(token, document) = (how\ many\ time\ this\ token\ appears\ in\ the\ document) / (total\ tokens\ in\ the\ deocument)$

```

...
count_vectorizer = CountVectorizer(binary=False, analyzer= 'word', stop_words='english')
x_train_cv = count_vectorizer.fit_transform(x_train)
x_test_cv = count_vectorizer.transform(x_test)
...

```

```

...
tfidf_vectorizer = TfidfVectorizer()
x_train_tfidf = tfidf_vectorizer.fit_transform(x_train)
x_test_tfidf = tfidf_vectorizer.transform(x_test)
...

```

Train the models and predict

We train 3 *MLP classifier* models (*multilayer perception models*) with only one hidden layer which has 100 neurons and 100 max iterations, each model will be trained on the count vectorized training set, tf-idf training set and one-hot encoding training set in order to compare the results we will get, after creating and training the three models, we will make prediction on the same training set and after displaying the classification report for the three models we get

```

...
mlp_count = MLPClassifier(hidden_layer_sizes=(100,), max_iter=100, solver='adam',
random_state=1)
mlp_count.fit(x_train_cv, y_train)
y_train_pred_count = mlp_count.predict(x_train_cv)
...

```

```

...
mlp_tfidf = MLPClassifier(hidden_layer_sizes=(100,), max_iter=100, solver='adam',
random_state=1)
mlp_tfidf.fit(x_train_tfidf, y_train)
y_train_pred_tfidf = mlp_tfidf.predict(x_train_tfidf)
...

```

```

...
mlp_onehot = MLPClassifier(hidden_layer_sizes=(100,), max_iter=100, solver='adam',
random_state=1)
mlp_onehot.fit(x_train_cv, y_train)
y_train_pred_onehot = mlp_onehot.predict(x_train_cv)
...

```

after make prediction on test set we get that the *accuracy* of *TF-IDF* is the highest with 0.8 while the two other models give 0.74 for each one, for the execution time that each models need to make prediction on the test set we get

```

...

count vectorizer: 1.506387710571289 seconds
tfidf vectorizer: 1.01865553855896 seconds

```

One-Hot Encoding: 1.5098521709442139 seconds

...

so, we get that tf-idf is the faster