Final Project

Group HOMEWORK. This final project can be collaborative. The maximum members of a group is 2. You can also work by yourself. Please respect the academic integrity. **Remember: if you get caught on cheating, you get F.**

A Introduction to the competition

Sexism is a growing problem online. It can inflict harm on women who are targeted, make online spaces inaccessible and unwelcoming, and perpetuate social asymmetries and injustices. Automated tools are now widely deployed to find, and assess sexist content at scale but most only give classifications for generic, high-level categories, with no further explanation. Flagging what is sexist content and also explaining why it is sexist improves interpretability, trust and understanding of the decisions that automated tools use, empowering both users and moderators.

This project is based on SemEval 2023 - Task 10 - Explainable Detection of Online Sexism (EDOS). Here you can find a detailed introduction to this task.

You only need to complete **TASK A - Binary Sexism Detection: a two-class (or binary) classification where systems have to predict whether a post is sexist or not sexist**. To cut down training time, we only use a subset of the original dataset (5k out of 20k). The dataset can be found in the same folder.

Different from our previous homework, this competition gives you great flexibility (and very few hints), you can determine:

- how to preprocess the input text (e.g., remove emoji, remove stopwords, text lemmatization and stemming, etc.);
- which method to use to encode text features (e.g., TF-IDF, N-grams, Word2vec, GloVe, Part-of-Speech (POS), etc.);
- which model to use.

Requirements

- **Input**: the text for each instance.
- **Output**: the binary label for each instance.
- **Feature engineering**: use at least 2 different methods to extract features and encode text into numerical values.
- **Model selection**: implement with at least 3 different models and compare their performance.
- **Evaluation**: create a dataframe with rows indicating feature+model and columns indicating Precision, Accuracy and F1-score (using weighted average). Your results should have at least 6 rows (2 feature engineering methods x 3 models). Report best performance with (1) your feature engineering method, and (2) the model you choose.
- **Format**: add explainations for each step (you can add markdown cells). At the end of the report, write a summary and answer the following questions:
  - What preprocessing steps do you follow?
  - How do you select the features from the inputs?
  - Which model you use and what is the structure of your model?
  - How do you train your model?
  - What is the performance of your best model?
  - What other models or feature engineering methods would you like to implement in the future?
- **Two Rules**, violations will result in 0 points in the grade:
  - Not allowed to use test set in the training: You CANNOT use any of the instances from test set in the training process.
  - Not allowed to use code from generative AI (e.g., ChatGPT).

Evaluation

The performance should be only evaluated on the test set (a total of 1086 instances). Please split original dataset into train set and test set. The test set should NEVER be used in the training process. The evaluation metric is a combination of precision, recall, and f1-score (use `classification_report` in sklearn).

The total points are 10.0. Each team will compete with other teams in the class on their best performance. Points will be deducted if not following the requirements above.

If ALL the requirements are met:

- Top 25% teams: 10.0 points.
- Top 25% - 50% teams: 8.5 points.
- Top 50% - 75% teams: 7.0 points.
- Top 75% - 100% teams: 6.0 points.

Submission

Similar as homework, submit both a PDF and .ipynb version of the report.

The report should include: (a)code, (b)outputs, (c)explainations for each step, and (d)summary (you can add markdown cells).

The due date is **December 8, Friday by 11:59pm.

In [ ]:
```
# insert your code
# import libraries
import pandas as pd
import matplotlib.pyplot as plt

# read data
df = pd.read_csv('edos_labelled_data.csv')
df.drop(['rewire_id'], axis=1, inplace=True)
df.head(10)
```

Out[ ]:

	text	label	split
0	In Nigeria, if you rape a woman, the men rape ...	not sexist	train
1	Then, she's a keeper. 😉	not sexist	train
2	This is like the Metallica video where the poo...	not sexist	train
3	woman?	not sexist	train
4	I bet she wished she had a gun	not sexist	train
5	Unlicensed day care worker reportedly tells co...	not sexist	train
6	[USER] Leg day is easy. Hot girls who wear min...	sexist	train
7	I don't know if you should avoid this one or e...	not sexist	train
8	I get a new pussy every other week or whenever...	sexist	train
9	I agree with that but at the same time I know ...	sexist	train

In [ ]:

```
#Function to generate the final report in a dictionary format
def put_in_final_dict(cls_rep, final_report_dict, name):
    if(final_report_dict == {}):
        final_report_dict["Feature+Model"] = [name]
        final_report_dict["Sexist Precision"] = [cls_rep["1"]["precision"]]
        final_report_dict["Sexist Recall"] = [cls_rep["1"]["recall"]]
        final_report_dict["Sexist F1-Score"] = [cls_rep["1"]["f1-score"]]
        final_report_dict["Non-Sexist Precision"] = [cls_rep["0"]["precision"]]
        final_report_dict["Non-Sexist Recall"] = [cls_rep["0"]["recall"]]
        final_report_dict["Non-Sexist F1-Score"] = [cls_rep["0"]["f1-score"]]
        final_report_dict["Weighted Average Precision"] = [cls_rep["weighted avg"]["precision"]]
        final_report_dict["Weighted Average Recall"] = [cls_rep["weighted avg"]["recall"]]
        final_report_dict["Weighted Average F1-Score"] = [cls_rep["weighted avg"]["recall"]]
    else:
        final_report_dict["Feature+Model"].append(name)
        final_report_dict["Non-Sexist Precision"].append(cls_rep["0"]["precision"])
        final_report_dict["Non-Sexist Recall"].append(cls_rep["0"]["recall"])
        final_report_dict["Non-Sexist F1-Score"].append(cls_rep["0"]["f1-score"])
        final_report_dict["Sexist Precision"].append(cls_rep["1"]["precision"])
        final_report_dict["Sexist Recall"].append(cls_rep["1"]["recall"])
        final_report_dict["Sexist F1-Score"].append(cls_rep["1"]["f1-score"])
```

```
        final_report_dict["Weighted Average Precision"].append(cls_rep["weighted avg"]["precision"])
        final_report_dict["Weighted Average Recall"].append(cls_rep["weighted avg"]["recall"])
        final_report_dict["Weighted Average F1-Score"].append(cls_rep["weighted avg"]["recall"])
    return final_report_dict
```

Data Cleanup

In the data cleanup step, we performed several operations to prepare our data for the machine learning model:

1. **Tokenization**: We broke down the text into individual words, or "tokens". This is a common first step in text analysis.
2. **Lemmatization**: We reduced words to their base or root form (e.g., "running" to "run"). This helps in consolidating different variations of the same word.
3. **Removing Out-of-Vocabulary Things**: We removed emojis, URLs, and other things that are not words.
4. **Removing Stop Words**: We removed "stop words", which are common words like "is", "the", and "a". These words don't provide meaningful information for the analysis.

These steps helped in reducing noise and dimensionality in our data, and made it suitable for training our machine learning model.

In [ ]:
```
import re
# Function to remove contractions from the text
def decontracted(phrase):
    # specific
    phrase = re.sub(r"won\'t", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    phrase = re.sub(r"don\'t", "do not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
```

```
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [ ]:

```
#data cleaning
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
from nltk.stem import   WordNetLemmatizer
emoji_pattern = re.compile("["
        u"\U0001F600-\U0001F64F"
        u"\U0001F300-\U0001F5FF"
        u"\U0001F680-\U0001F6FF"
        u"\U0001F1E0-\U0001F1FF"
        "]+", flags=re.UNICODE)
bracket_pattern = re.compile("\[[A-Z]+\]")
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
stop_words = set(stopwords.words('english'))
tokenizer = RegexpTokenizer(r'\w+')
wordnet_lemmatizer = WordNetLemmatizer()
df['word_tokens'] = ''
for i in range(len(df)):
    if "sexist" == df.loc[i,'label']:
        df.loc[i,'label'] = "1"
    else:
        df.loc[i,'label'] = "0"
    df.loc[i,'text'] = emoji_pattern.sub(r'', df.loc[i,'text'])
    df.loc[i,'text'] = bracket_pattern.sub(r'', df.loc[i,'text'])
    df.loc[i,'text'] = df.loc[i,'text'].lower()
    df.loc[i,'text'] = decontracted(df.loc[i,'text'])
    word_tokens = tokenizer.tokenize(df.loc[i,'text'])
    lemmatized_tokens = [wordnet_lemmatizer.lemmatize(w) for w in word_tokens]
    filtered_sentence = [w for w in lemmatized_tokens if not w in stop_words and not w.isdigit()]
    df.loc[i,'word_tokens'] = " ".join(filtered_sentence)

#splitting the data into train and test
df_train = df[df['split'] != 'test']
df_test = df[df['split'] == 'test']
df.head(10)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Shreygoel\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Shreygoel\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Shreygoel\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[ ]:

	text	label	split	word_tokens
0	in nigeria, if you rape a woman, the men rape ...	0	train	nigeria rape woman men rape back nsfw nigeria ...
1	then, she is a keeper.	0	train	keeper
2	this is like the metallica video where the poo...	0	train	like metallica video poor mutilated bastard wa...
3	woman?	0	train	woman
4	i bet she wished she had a gun	0	train	bet wished gun
5	unlicensed day care worker reportedly tells co...	0	train	unlicensed day care worker reportedly tell cop...
6	leg day is easy. hot girls who wear miniskirt...	1	train	leg day easy hot girl wear miniskirt get asked
7	i do not know if you should avoid this one or ...	0	train	know avoid one encourage woman rape man rape f...
8	i get a new pussy every other week or whenever...	1	train	get new pussy every week whenever want really ...
9	i agree with that but at the same time i know ...	1	train	agree time know well enough say love woman min...

Feature Extraction and Model Training

In this project, we used Count Vectorization to transform our text data into a format that can be used by machine learning models. Count Vectorization converts a collection of text documents into a matrix of token counts. This is a simple and effective way to represent text data numerically.

After transforming our text data into a document-term matrix using Count Vectorization, we trained several machine learning models:

1. **Logistic Regression**: This is a simple yet powerful linear model that is widely used for classification problems. It's particularly good at handling binary classification problems.

2. **Multinomial Naive Bayes**: This is a probabilistic classifier that is highly effective for text data. It's based on applying Bayes' theorem with strong independence assumptions between the features.

3. **MLPClassifier**: This stands for Multi-Layer Perceptron Classifier. It's a type of neural network that consists of at least three layers of nodes and can handle complex data.

Each of these models was trained on our document-term matrix and then used to make predictions on unseen data. The performance of each model was evaluated using various metrics.

In [ ]:
```
from sklearn.feature_extraction.text import CountVectorizer
#vectorizing the data
vectorize = CountVectorizer()
vectorize.fit(df_train['word_tokens'])
X_train = vectorize.transform(df_train['word_tokens'])
X_test = vectorize.transform(df_test['word_tokens'])
y_train = df_train['label']
y_test = df_test['label']
final_report_dict = {}
```

Logistic Regression

In [ ]:
```
from sklearn.linear_model import LogisticRegression

#Logistic Regression with Count Vectorizer
best_score = 0
best_c = 0
for i in range(1,100):
    i = i/100
    model = LogisticRegression(penalty='l1', C=i, solver='liblinear', max_iter=1000)
    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
    if score > best_score:
        best_score = score
        best_c = i
```

```
print("Best score: ", best_score*100)
print("Best C: ", best_c)
```

```
Best score:  82.96500920810314
Best C:  0.4
```

In [ ]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
#Using the best C value to train the model
model = LogisticRegression(penalty='l1', C=best_c, solver='liblinear', max_iter=1000)
model.fit(X_train, y_train)

y_pred_LR = model.predict(X_test)

#Printing the confusion matrix and classification report
print(confusion_matrix(y_test, y_pred_LR))
cls_rep = classification_report(y_test, y_pred_LR)
print(cls_rep)

#Adding the results to the final report
cls_rep = (classification_report(y_test, y_pred_LR, output_dict=True))
final_report_dict = put_in_final_dict(cls_rep, final_report_dict, "CountVectorizer+LogisticRegression")
```

```
[[757  32]
 [153 144]]
              precision    recall  f1-score   support

           0       0.83      0.96      0.89       789
           1       0.82      0.48      0.61       297

    accuracy                           0.83      1086
   macro avg       0.83      0.72      0.75      1086
weighted avg       0.83      0.83      0.81      1086
```

Multinomial Naive Bayes

In [ ]:
```
from sklearn.naive_bayes import MultinomialNB

#Multinomial Naive Bayes with Count Vectorizer
best_score = 0
```

```
best_alpha = 0
for i in range(1,100):
    i = i/100
    model = MultinomialNB(alpha=i)
    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
    if score > best_score:
        best_score = score
        best_alpha = i
print("Best score: ", best_score*100)
print("Best alpha: ", best_c)
```

```
Best score:  77.53222836095765
Best alpha:  0.4
```

In [ ]:

```
#Using the best alpha value to train the model
model = MultinomialNB(alpha=best_alpha)
model.fit(X_train, y_train)

y_pred_MNB = model.predict(X_test)

#Printing the confusion matrix and classification report
print(confusion_matrix(y_test, y_pred_MNB))
cls_rep = classification_report(y_test, y_pred_MNB)
print(cls_rep)

#Adding the results to the final report
cls_rep = (classification_report(y_test, y_pred_MNB, output_dict=True))
final_report_dict = put_in_final_dict(cls_rep, final_report_dict, "CountVectorizer+MultinomialNB")
```

```
[[716  73]
 [171 126]]
              precision    recall  f1-score   support

           0       0.81      0.91      0.85       789
           1       0.63      0.42      0.51       297

    accuracy                           0.78      1086
   macro avg       0.72      0.67      0.68      1086
weighted avg       0.76      0.78      0.76      1086
```

MLPClassifier

In [ ]:
```
from sklearn.neural_network import MLPClassifier

#MLP Classifier with Count Vectorizer
activation = 'tanh'
solver = 'adam'
best_score = 0
best_alpha = 0
batch_size = 2**5
hidden_layer_sizes = (7,7,7)

## For finding best alpha, Code commneted out becuae takes too long to run everytime
# for i in range(1,100):
#     i = i/100
#     clf = MLPClassifier(activation=activation, solver=solver, alpha=i, hidden_layer_sizes=hidden_layer_sizes, rando
#     clf.fit(X_train, y_train)
#     score = clf.score(X_test, y_test)
#     # print("Score for ",i, ": ", score*100)
#     if score > best_score:
#         best_score = score
#         best_alpha = i
# print("Best score: ", best_score*100)
# print("Best alpha: ", best_alpha)
best_alpha = 0.47

#Using the best alpha value to train the model
clf = MLPClassifier(activation=activation, solver=solver, alpha=best_alpha, hidden_layer_sizes=hidden_layer_sizes, ra
clf.fit(X_train, y_train)

y_pred_MLP = clf.predict(X_test)

#Printing the confusion matrix and classification report
print(confusion_matrix(y_test, y_pred_MLP))
cls_rep = classification_report(y_test, y_pred_MLP)
print(cls_rep)

#Adding the results to the final report
```

```
cls_rep = (classification_report(y_test, y_pred_MLP, output_dict=True))
final_report_dict = put_in_final_dict(cls_rep, final_report_dict, "CountVectorizer+MLPClassifier")
```

```
[[724  65]
 [136 161]]
              precision    recall  f1-score   support

           0       0.84      0.92      0.88       789
           1       0.71      0.54      0.62       297

    accuracy                           0.81      1086
   macro avg       0.78      0.73      0.75      1086
weighted avg       0.81      0.81      0.81      1086
```

Feature Extraction with TF-IDF

In this project, we used TF-IDF (Term Frequency-Inverse Document Frequency) to transform our text data into a format that can be used by machine learning models.

TF-IDF is a numerical statistic that reflects how important a word is to a document in a collection or corpus. It is often used in information retrieval and text mining. The TF-IDF value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

After transforming our text data into a matrix of TF-IDF features, we trained several machine learning models. The models were able to use these features to learn patterns in the data and make predictions on unseen data.

1. **Logistic Regression**: This is a simple yet powerful linear model that is widely used for classification problems. It's particularly good at handling binary classification problems.
2. **Multinomial Naive Bayes**: This is a probabilistic classifier that is highly effective for text data. It's based on applying Bayes' theorem with strong independence assumptions between the features.
3. **MLPClassifier**: This stands for Multi-Layer Perceptron Classifier. It's a type of neural network that consists of at least three layers of nodes and can handle complex data.

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

#TF-IDF Vectorizer
vectorize = TfidfVectorizer()
vectorize.fit(df_train['word_tokens'])
X_train = vectorize.transform(df_train['word_tokens'])
X_test = vectorize.transform(df_test['word_tokens'])
y_train = df_train['label']
y_test = df_test['label']
```

Logistic Regression

In [ ]:
```
#Logistic Regression with TF-IDF Vectorizer
best_score = 0
best_c = 0
for i in range(1,100):
    i = i/100
    model = LogisticRegression(penalty='l1', C=i, solver='liblinear', max_iter=1000)
    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
    if score > best_score:
        best_score = score
        best_c = i
print("Best score: ", best_score*100)
print("Best C: ", best_c)
```

```
Best score:  80.38674033149171
Best C:  0.77
```

In [ ]:
```
#Using the best C value to train the model
model = LogisticRegression(penalty='l1', C=best_c, solver='liblinear', max_iter=1000)
model.fit(X_train, y_train)

y_pred_LR = model.predict(X_test)

#Printing the confusion matrix and classification report
print(confusion_matrix(y_test, y_pred_LR))
cls_rep = classification_report(y_test, y_pred_LR)
print(cls_rep)
```

```
#Adding the results to the final report
cls_rep = (classification_report(y_test, y_pred_LR, output_dict=True))
final_report_dict = put_in_final_dict(cls_rep, final_report_dict, "TfidfVectorizer+LogisticRegression")
```

```
[[760  29]
 [184 113]]
              precision    recall  f1-score   support

           0       0.81      0.96      0.88       789
           1       0.80      0.38      0.51       297

    accuracy                           0.80      1086
   macro avg       0.80      0.67      0.70      1086
weighted avg       0.80      0.80      0.78      1086
```

Multinomial Naive Bayes

In [ ]:
```
#Multinomial Naive Bayes with TF-IDF Vectorizer
best_score = 0
best_alpha = 0
for i in range(1,100):
    i = i/100
    model = MultinomialNB(alpha=i)
    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
    if score > best_score:
        best_score = score
        best_alpha = i
print("Best score: ", best_score*100)
print("Best alpha: ", best_alpha)
```

```
Best score:  75.87476979742172
Best alpha:  0.35
```

In [ ]:
```
#Using the best alpha value to train the model
model = MultinomialNB(alpha=best_alpha)
model.fit(X_train, y_train)

y_pred_MNB = model.predict(X_test)
```

```
#Printing the confusion matrix and classification report
print(confusion_matrix(y_test, y_pred_MNB))
cls_rep = classification_report(y_test, y_pred_MNB)
print(cls_rep)

#Adding the results to the final report
cls_rep = (classification_report(y_test, y_pred_MNB, output_dict=True))
final_report_dict = put_in_final_dict(cls_rep, final_report_dict, "TfidfVectorizer+MultinomialNB")
```

```
[[773  16]
 [246  51]]
              precision    recall  f1-score   support

           0       0.76      0.98      0.86       789
           1       0.76      0.17      0.28       297

    accuracy                           0.76      1086
   macro avg       0.76      0.58      0.57      1086
weighted avg       0.76      0.76      0.70      1086
```

MLPClassifier

In [ ]:
```
#MLP Classifier with TF-IDF Vectorizer
activation = 'relu'
solver = 'adam'
best_score = 0
best_alpha = 0
batch_size = 2**3
hidden_layer_sizes = (2,4,4)

## Found after a long and extremly slow process, code removed becuase it would take 10+ minutes to run
# for i in range(1,100):
#     i = i/100
#     clf = MLPClassifier(activation=activation, solver=solver, alpha=best_alpha, hidden_layer_sizes=hidden_layer_siz
#     clf.fit(X_train, y_train)
#     score = clf.score(X_test, y_test)
#     # print("Score for ",i, ": ", score*100)
#     if score > best_score:
```

```
#         best_score = score
#         best_alpha = i
# print(best_alpha)
# print(best_score)
best_alpha = 0.0411002

#Using the best alpha value to train the model
clf = MLPClassifier(activation=activation, solver=solver, alpha=best_alpha, hidden_layer_sizes=hidden_layer_sizes, ra
clf.fit(X_train, y_train)

y_pred_MLP = clf.predict(X_test)

#Printing the confusion matrix and classification report
print(confusion_matrix(y_test, y_pred_MLP))
cls_rep = classification_report(y_test, y_pred_MLP)
print(cls_rep)

#Adding the results to the final report
cls_rep = (classification_report(y_test, y_pred_MLP, output_dict=True))
final_report_dict = put_in_final_dict(cls_rep, final_report_dict, "TfidfVectorizer+MLPClassifier")
```

```
[[686 103]
 [129 168]]
              precision    recall  f1-score   support

           0       0.84      0.87      0.86       789
           1       0.62      0.57      0.59       297

    accuracy                           0.79      1086
   macro avg       0.73      0.72      0.72      1086
weighted avg       0.78      0.79      0.78      1086
```

Creating the Final Report

In [ ]:
```
#Puting the final report in a dataframe and sorting it by the weighted average F1-Score
df_final_report = pd.DataFrame(final_report_dict)
df_final_report.sort_values(by=['Weighted Average F1-Score'], inplace=True, ascending=False)
df_final_report.head(10)
```

Out[ ]:

	Feature+Model	Sexist Precision	Sexist Recall	Sexist F1-Score	Non-Sexist Precision	Non-Sexist Recall	Non-Sexist F1-Score	Weighted Average Precision	Weighted Average Recall	Weighte Averag F1-Scor
0	CountVectorizer+LogisticRegression	0.818182	0.484848	0.608879	0.831868	0.959442	0.891112	0.828125	0.829650	0.82965
2	CountVectorizer+MLPClassifier	0.712389	0.542088	0.615679	0.841860	0.917617	0.878108	0.806453	0.814917	0.81491
3	TfidfVectorizer+LogisticRegression	0.795775	0.380471	0.514806	0.805085	0.963245	0.877092	0.802539	0.803867	0.80386
5	TfidfVectorizer+MLPClassifier	0.619926	0.565657	0.591549	0.841718	0.869455	0.855362	0.781062	0.786372	0.78637
1	CountVectorizer+MultinomialNB	0.633166	0.424242	0.508065	0.807215	0.907478	0.854415	0.759616	0.775322	0.77532
4	TfidfVectorizer+MultinomialNB	0.761194	0.171717	0.280220	0.758587	0.979721	0.855088	0.759300	0.758748	0.75874

Summary

1. What preprocessing steps do you follow?

   Your answer: We first wanted to clean the provided data so we can have a better understanding of the data. We removed all the punctuations, stopwords, and emojis. We also lemmatized the data so we can have a better understanding of the data. We also removed all the numbers from the data, and all the "User" data that was present in the tweet. We used some regex and nltk library for this process.

2. How do you select the features from the inputs?

   Your answer: Because we already cleaned up and lemmatized the data, we were able to very easyly "vectorize" the strings using both N-grams (Count Vectorizing) and Tf-idf vectorizing method. For the CountVectorizing nad TF-idf, we allowed every word to be its own vectors thus we ended up with 1000+ features which when put together will be the input for our model.

3. Which model you use and what is the structure of your model?

   Your answer: In this project, we used several machine learning models to classify our text data. These models include:

a. **Logistic Regression**: This is a linear model for classification that predicts the probability of categorical outcomes. It's particularly good at binary classification problems.

b. **Multinomial Naive Bayes**: This is a probabilistic classifier that assumes all the features are mutually independent. It's highly effective for text data.

c. **MLPClassifier**: This stands for Multi-Layer Perceptron Classifier. It's a type of neural network that consists of at least three layers of nodes. It can handle complex data and perform tasks that linear classifiers cannot.

The structure of our models varies. Logistic Regression and Multinomial Naive Bayes are simpler models with fewer parameters, while the MLPClassifier is a more complex model with many more parameters, including weights and biases for each node in the network.

Each of these models was trained on our TF-IDF/CountVectorisation feature matrix and then used to make predictions on unseen data. The performance of each model was evaluated using various metrics.

4. How do you train your model?

   Your answer: Training a machine learning model involves providing the model with input data and the corresponding output labels, and allowing the model to learn the relationship between the data and the output. Here's a general outline of the steps we took to train our models:

   a. **Prepare the Data**: We first prepared our data by transforming our text data into a matrix of TF-IDF features. This involved tokenizing the text, removing stop words, and calculating the TF-IDF values for each word in our vocabulary.

   b. **Initialize the Model**: We then initialized our machine learning model. This could be a Logistic Regression model, a Multinomial Naive Bayes model, or a MLPClassifier.

   c. **Fit the Model**: We used the `fit` method of our model to train it. We passed our TF-IDF feature matrix and the corresponding labels to this method.

   d. **Evaluate the Model**: After training, we evaluated our model's performance on unseen test data. We used various metrics such as accuracy, precision, recall, and F1 score to measure the performance of our model.

This process allows the model to learn from our data and make accurate predictions on unseen data.

5. What is the performance of your best model?

   Your answer: Our best performing model achieved the following results on the test data:

   - Accuracy: 83%
   - Precision: 82.81%
   - Recall: 82.96%
   - F1 Score: 82.96%

These results indicate that our model is highly effective at predicting [CountVectorizer+LogisticRegreasion].

6. What other models or feature engineering methods would you like to implement in the future?

   Your answer: In the future, we would like to implement the following models and feature engineering methods:

   - **Support Vector Machines**: This is a powerful model that can be used for both classification and regression problems. It's particularly effective for text data.

   - **Word Embeddings**: This is a technique for representing text data as vectors of real numbers. It's a powerful feature engineering method that can be used to train deep learning models.

   - **Bidirectional Encoder Representations from Transformers (BERT)**: This is a powerful deep learning model that can be used for a variety of natural language processing tasks. It's particularly effective for text classification problems.