Spam Detection HW

Read complete instructions before starting the HW

Q1: Load the dataset (1 Point)

- For this Hw you will usespam dataset from kaggle which can be found from this link. You can download this data and either upload it in google drive or in colab workspace. Load the data in pandas dataframe.
- There are only two useful columns. These columns are related to (1) label (ham and spam) and the (2) text of email.
- Rename columns as label and message
- Find the % ham amd spam in the data.

```
# Connect Google colab to drive
from pathlib import Path
if 'google.colab' in str(get_ipython()):
  from google.colab import drive
  drive.mount('/content/drive')
  base_folder = Path('/content/drive/MyDrive/NLP')
data_folder = base_folder/'Datasets'
```

```
    Mounted at /content/drive
```

```
#Load the pandas dataframe
data = data_folder/'spam.csv'
import pandas as pd
spam_data = pd.read_csv(data,encoding='latin-1')
spam_data = spam_data.drop(columns = ['Unnamed: 2','Unnamed: 3','Unnamed: 4'])
#Rename the columns
spam_data.columns = ['Label' , 'Message']
```

```
spam_data.head()
```

	Label	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

```
#Find the percentage of spam and ham in data.
spam_data['Label'].value_counts(normalize = True)
```

```
    ham     0.865937
    spam    0.134063
    Name: Label, dtype: float64
```

Q2 : Provide the metric for evaluating model (1 Point)

As you will notice, the data is highly imbalanced (most messages are labelled as ham and only few are labelled as spam). Always predicting ham will give us very good accuracy (close to 90%). So you need to choose a different metric.

Task: Provde the metric you will choose to evaluate your model. Explain why this is an appropriate metric for this case.

The area under the ROC curve,classification report and confusion matrix are of the best metrics for imbalanced data.The area under the ROC curve focuses on the model's ability to distinguish between positive and negative classes. It is a useful metric when you want to assess the model's overall performance. The confusion gives an overview of how many the model is able to predict correctly in both classes.

Q3 : Classification Pipelines (18 Points)

In the previous lectures you learned Data processing, Featurization such as CountVectorizer, TFIDFVectorizer, and also Feature Engineering.

- You will now use folllowing methods to create fearures which you can use in your model.

1. Sparse Embeddings (TF-IDF) (6 Points)
2. Feature Engineering (see examples below) (6 Points)
3. Sparse Embeddings (TF-IDF) + Feature Engineering (6 Points)

Approach:

****Use a smaller subset of dataset (e.g. 5-10 %) to evaluate the three pipelines . Based on your analysis (e.g. model score, learning curves) , choose one pipeline from the three. Provde your rational for choosing the pipleine. Train only the final pipeline on randomly selected larger subset (e.g. 40%) of the data.**

Requirements:

1. You can use any ML model (Logistic Regression, XgBoost) for the classification. You will need to tune the **model for imbalanced dataset** (The link on XGBoost tutorial for imbalanced data: https://machinelearningmastery.com/xgboost-for-imbalanced-classification/).
2. For feature engineering, you can choose from the examples below. You do not have to use all of them. You can add other featues as well. Think about what faetures can distinguish a spam from a regular email. Some examples :

> > Count of following (Words, characters, digits, exclamation marks, numbers, Nouns, ProperNouns, AUX, VERBS, Adjectives, named entities, spelling mistakes (see the link on how to get spelling mistakes https://pypi.org/project/pyspellchecker/).

3. For Sparse embeddings you will use **tfidf vectorization**. You need to choose appropriate parameters e.g. min_df, max_df, max_faetures, n-grams etc.).
4. Think carefully about the pre-processing you will do.

Tip: **Using GridSearch for hyperparameter tuning might take a lot of time. Try using RandomizedSearch.** You can also explore faster implementation of Gridsearch and RandomizedSearch in sklearn:

1. Halving Grid Search
2. HalvingRandomSearchCV

SOLUTION

Pipeline 1 - Sparse Embeddings (TF-IDF)

```
# Get a smaller data set (20%) for modelling
small_sample = spam_data.sample(frac = 0.20 , random_state = 11)
small_sample.shape
```

```
    (1114, 2)
```

```
small_sample.head()
```

	Label	Message
4460	ham	Thanks again for your reply today. When is ur ...
1049	spam	18 days to Euro2004 kickoff! U will be kept in...
1810	ham	Now, whats your house # again ? And do you hav...
2848	spam	YOUR CHANCE TO BE ON A REALITY FANTASY SHOW ca...
2428	ham	She.s find. I sent you an offline message to k...

```
#Preprocessing
import re
from bs4 import BeautifulSoup
def cleaned_text(text):
  #Remove HTML tags
  clean_text = BeautifulSoup(text , "html.parser").get_text()
  #Remove new line characters and replace them with space
  clean_text = re.sub(r'[\n\r]',' ', clean_text)
  #Remove URLs:
  clean_text = re.sub(r'http\S+' , '' , clean_text)
  #Remove Emails:
  clean_text = re.sub(r'\S+@\S+' , '' , clean_text)
  #Remove punctuation:
  clean_text = re.sub(r'[^\w\s.]' , '' , clean_text)
  return clean_text
small_sample['cleaned_text'] = small_sample['Message'].apply(cleaned_text)
```

```
<ipython-input-175-7769aadac917>:6: MarkupResemblesLocatorWarning: The input looks more like a filename than markup. You may want t
  clean_text = BeautifulSoup(text , "html.parser").get_text()
```

```
#Tfidf Vectorizer
new_sample = small_sample.drop(columns = ['Message'])
train_sample = new_sample.iloc[:557]
test_sample = new_sample.iloc[557:]
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_Vect = TfidfVectorizer(max_features = 1000 , max_df = 300)
tfidf_Vect.fit(train_sample['cleaned_text'].values)
```

```
▾              TfidfVectorizer
TfidfVectorizer(max_df=300, max_features=1000)
```

```
features = tfidf_Vect.get_feature_names_out()
len(features)
```

```
1000
```

```
#Train-Test Split-
X_train_transform = tfidf_Vect.transform(train_sample['cleaned_text'].values)
X_test_transform = tfidf_Vect.transform(test_sample['cleaned_text'].values)
vocab = tfidf_Vect.get_feature_names_out()
X_train1 = pd.DataFrame(X_train_transform.toarray(), columns=vocab)
X_test1 = pd.DataFrame(X_test_transform.toarray() , columns = vocab)
X_train1.head()
```

	0800	08000839402	08000930705	08002986906	0870	08707509020	08712460324
0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0
1	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0
2	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0
3	0.0	0.0	0.0	0.0	0.219896	0.219896	0.0
4	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0

5 rows × 1000 columns

```
label_map = {"ham": 0, "spam": 1}
y_train1 = train_sample['Label'].map(label_map)
y_test1 = test_sample['Label'].map(label_map)
```

```
#Model Building(XGBoost Classifier)-
import xgboost
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report,confusion_matrix
xgb_classifier = XGBClassifier()
xgb_classifier.fit(X_train1, y_train1)
y_pred1 = xgb_classifier.predict(X_test1)
training_scores = cross_val_score(xgb_classifier, X_train1, y_train1, scoring='roc_auc', cv=3, n_jobs=-1)
testing_scores = cross_val_score(xgb_classifier, X_test1 , y_test1 , cv = 3 , scoring = 'roc_auc', n_jobs=-2)
print('Mean Training ROC AUC for Pipeline 1: %.5f' % training_scores.mean())
print('Mean Testing ROC AUC for Pipeline 1: %.5f' % testing_scores.mean())
```

```
Mean Training ROC AUC for Pipeline 1: 0.93775
Mean Testing ROC AUC for Pipeline 1: 0.93608
```

```
#Classification Report-
print('\nTest set classification report for Pipeline 1:\n\n',
      classification_report(y_test1, y_pred1))
```

```
Test set classification report for Pipeline 1:

               precision    recall  f1-score   support

           0       0.96      0.99      0.97       489
           1       0.87      0.69      0.77        68

    accuracy                           0.95       557
   macro avg       0.91      0.84      0.87       557
```

```
     weighted avg       0.95      0.95      0.95       557
```

```
#Confusion Matrix
cm1 = confusion_matrix(y_test1, y_pred1)

# Print the confusion matrix
print("Confusion Matrix:")
print(cm1)
```

```
     Confusion Matrix:
     [[482   7]
      [ 21  47]]
```

Pipeline 2 - Feature Engineering

```
#Import spacy
import spacy
nlp = spacy.load('en_core_web_sm')
```

```
#Feature Engineering -

disabled = nlp.select_pipes(disable=['lemmatizer','ner'])
entities_count = []
symbol_count = []
punct_count = []
num_count = []
uppercase_count = []
special_char_count = []
url_email_count = []
for doc in nlp.pipe(small_sample.Message.values , batch_size = 100):
    entities = [token.text for token in doc if (token.pos_ in ['PROPN'])]
    entities_count.append(len(entities))
    symbols = [token.text for token in doc if (token.pos_ in ['SYM'])]
    symbol_count.append(len(symbols))
    punct = [token.text for token in doc if (token.pos_ in ['PUNCT'])]
    punct_count.append(len(punct))
    number = [token.text for token in doc if (token.pos_ in ['NUM'])]
    num_count.append(len(number))
    uppercase = [token.text for token in doc if (token.is_upper)]
    uppercase_count.append(len(uppercase))
    special_char = [token.text for token in doc if (c.isalpha() and not c.isascii() for c in token.text)]
    special_char_count.append(len(special_char))
    url_email = [token.text for token in doc if (token.like_url or token.like_email)]
    url_email_count.append(len(url_email))
disabled.restore()
```

```
small_sample2 = small_sample.copy()
small_sample2['Entity_count'] = entities_count
small_sample2['Symbol_count'] = symbol_count
small_sample2['Punct_count'] = punct_count
small_sample2['Num_count'] = num_count
small_sample2['Uppercase_count'] = uppercase_count
small_sample2['Special_char_count'] = special_char_count
small_sample2['Url_email_count'] = url_email_count
small_sample2.head()
```

	Label	Message	cleaned_text	Entity_count	Symbol_count	Punct_count	Num_
4460	ham	Thanks again for your reply today. When is ur ...	Thanks again for your reply today. When is ur ...	3	0	6	
1049	spam	18 days to Euro2004 kickoff! U will be kept in...	18 days to Euro2004 kickoff U will be kept inf...	3	0	3	
1810	ham	Now, whats your house # again ? And do	Now whats your house again And do you have a...	0	0	3	

```
# Train-Test Split
data_sample2 = small_sample2.drop(columns = ['Message' , 'cleaned_text'])
train_sample2 = data_sample2.iloc[:557]
test_sample2 = data_sample2.iloc[557:]
label_map = {"ham": 0, "spam": 1}
X_train2 = train_sample2.drop(columns = ['Label'])
X_test2 = test_sample2.drop(columns = ['Label'])
y_train2 = train_sample2['Label'].map(label_map)
y_test2 = test_sample2['Label'].map(label_map)


#Model Building
xgb_classifier.fit(X_train2, y_train2)
y_pred2 = xgb_classifier.predict(X_test2)
training_scores2 = cross_val_score(xgb_classifier, X_train2, y_train2, scoring='roc_auc', cv=3, n_jobs=-1)
testing_scores2 = cross_val_score(xgb_classifier, X_test2 , y_test2 , cv = 3 , scoring = 'roc_auc', n_jobs=-2)
print('Mean Training ROC AUC for Pipeline 2: %.5f' % training_scores2.mean())
print('Mean Testing ROC AUC for Pipeline 2: %.5f' % testing_scores2.mean())
```

```
Mean Training ROC AUC for Pipeline 2: 0.92809
Mean Testing ROC AUC for Pipeline 2: 0.94049
```

```
#Classification Report-
print('\nTest set classification report for Pipeline 2:\n\n',
      classification_report(y_test2, y_pred2))
```

```

Test set classification report for Pipeline 2:

               precision    recall  f1-score   support

           0       0.96      0.96      0.96       489
           1       0.70      0.71      0.70        68

    accuracy                           0.93       557
   macro avg       0.83      0.83      0.83       557
weighted avg       0.93      0.93      0.93       557

```

```
#Confusion Matrix-
cm2 = confusion_matrix(y_test2, y_pred2)

# Print the confusion matrix
print("Confusion Matrix:")
print(cm2)
```

```
Confusion Matrix:
[[468  21]
 [ 20  48]]
```

Pipeline 3 - Sparse Embeddings (TF-IDF) + Feature Engineering

```
#Train-Test Split-
df3 = small_sample2.drop(columns = ['Message'])
X_train3 = df3.drop(columns = ['Label']).iloc[:557,]
X_test3 = df3.drop(columns = ['Label']).iloc[557:,]
y_train3 = df3['Label'].iloc[:557,].map(label_map)
y_test3 = df3['Label'].iloc[557:,].map(label_map)


#Pipeline Building
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
vectorizer = Pipeline([('tfidf', TfidfVectorizer(max_features=5)), ])
combined_features = ColumnTransformer(
    transformers=[
        ('tfidf', vectorizer, 'cleaned_text'),
    ],remainder='passthrough'
)


#Testing the Pipeline
test = combined_features.fit_transform(X_train3)
test
```

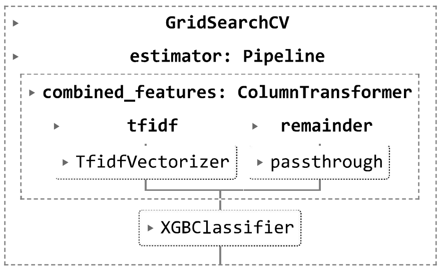
```
array([[ 0.69216262,  0.36062749,  0.32944234, ...,  0.        ,
        55.        ,  0.        ],
       [ 0.48431125,  0.        ,  0.46102644, ...,  4.        ,
        28.        ,  0.        ],
       [ 0.78007312,  0.        ,  0.        , ...,  0.        ,
        17.        ,  0.        ],
       ...,
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         7.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ...,  1.        ,
        11.        ,  0.        ],
       [ 0.        ,  0.56157885,  0.        , ...,  4.        ,
        34.        ,  0.        ]])
```

```
#Final Classifier
classifier_3 = Pipeline([('combined_features',  combined_features),
                         ('classifier', XGBClassifier()),
                         ])
```

```
#Parameters for GridSearchCV
param_grid_classifier_3 = {'combined_features__tfidf__tfidf__max_features': [500, 1000, 2000],
                           'classifier__learning_rate': [0.01, 0.1, 0.2]
                           }
```

```
#GridSearchCV
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
grid_classifier_3 = GridSearchCV(estimator=classifier_3,
                                 param_grid=param_grid_classifier_3,
                                 cv=3)
```

```
grid_classifier_3.fit(X_train3, y_train3)
```



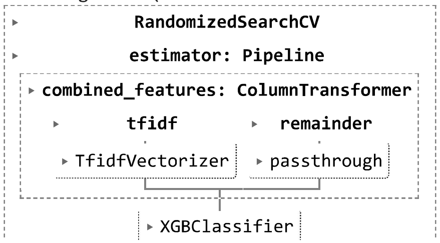
```
#Results of GridSearchCV-
print(
    "Best cross-validation score: {:.2f}".format(grid_classifier_3.best_score_))
print("\nBest parameters: ", grid_classifier_3.best_params_)
print("\nBest estimator: ", grid_classifier_3.best_estimator_)
```

```
    Best cross-validation score: 0.95

    Best parameters:  {'classifier__learning_rate': 0.2, 'combined_features__tfidf__tfidf__max_features': 500}

    Best estimator:  Pipeline(steps=[('combined_features',
                     ColumnTransformer(remainder='passthrough',
                                       transformers=[('tfidf',
                                                      Pipeline(steps=[('tfidf',
                                                                       TfidfVectorizer(max_features=500))]),
                                                      'cleaned_text')])),
                    ('classifier',
                     XGBClassifier(base_score=None, booster=None, callbacks=None,
                                   colsample_bylevel=None, colsample_bynode=None,
                                   colsample_bytree=None,
                                   early_stopping_rounds=None,
                                   e...
                                   feature_types=None, gamma=None, gpu_id=None,
                                   grow_policy=None, importance_type=None,
                                   interaction_constraints=None, learning_rate=0.2,
                                   max_bin=None, max_cat_threshold=None,
                                   max_cat_to_onehot=None, max_delta_step=None,
                                   max_depth=None, max_leaves=None,
                                   min_child_weight=None, missing=nan,
                                   monotone_constraints=None, n_estimators=100,
                                   n_jobs=None, num_parallel_tree=None,
                                   predictor=None, random_state=None, ...))])
```

```
#RandomizedSearchCV
random_classifier_3 = RandomizedSearchCV(estimator=classifier_3,
                                 param_distributions=param_grid_classifier_3,
                                 cv=3)
```

```
random_classifier_3.fit(X_train3, y_train3)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:305: Us
  warnings.warn(
```



```
#Results of RandomizedSearchCV-
print("Best Parameters:", random_classifier_3.best_params_)
print("Best Score:", random_classifier_3.best_score_)
```

```
Best Parameters: {'combined_features__tfidf__tfidf__max_features': 500, 'classifier__learning_rate': 0.2}
Best Score: 0.9515160321611935
```

```
training_scores3 = cross_val_score(grid_classifier_3.best_estimator_, X_train3, y_train3, scoring='roc_auc', cv=3, n_jobs=-1)
testing_scores3 = cross_val_score(grid_classifier_3.best_estimator_, X_test3 , y_test3 , cv = 3 , scoring = 'roc_auc', n_jobs=-2)
print('Mean Training ROC AUC for Pipeline 3: %.5f' % training_scores3.mean())
print('Mean Testing ROC AUC for Pipeline 3: %.5f' % testing_scores3.mean())
```

```
Mean Training ROC AUC for Pipeline 3: 0.96132
Mean Testing ROC AUC for Pipeline 3: 0.97390
```

```
model3 = grid_classifier_3.best_estimator_
y_pred3 = model3.predict(X_test3)
```

```
#Classification Report-
print('\nTest set classification report for Pipeline 3:\n\n',
      classification_report(y_test3, y_pred3))
```

```
Test set classification report for Pipeline 3:

               precision    recall  f1-score   support

           0       0.97      0.99      0.98       489
           1       0.88      0.78      0.83        68

    accuracy                           0.96       557
   macro avg       0.93      0.88      0.90       557
weighted avg       0.96      0.96      0.96       557
```

```
#Confusion Matrix-
cm3 = confusion_matrix(y_test3, y_pred3)

# Print the confusion matrix
print("Confusion Matrix:")
print(cm3)
```

```
Confusion Matrix:
[[482   7]
 [ 15  53]]
```

Required Submissions:

1. Submit two colab/jupyter notebooks

- (analysis with smaller subset and all three pipelines)
- (analysis with bigger subset and only final pipeline)

2. Pdf version of the notebooks (HWs will not be graded if pdf version is not provided.
3. **The notebooks and pdf files should have the output.**
4. **Name files as follows : FirstName_file1_hw2, FirstName_file2_h2**