M6 Team Assignment: Spooky Authorship With Spark Part 2

Group 13

- Aidan Lonergan
- Daniel Lillard
- Radhika Garg
- Claudine Uwiragiye

Objective

- In this assignment, your team will improve your scores from the first Spooky Authorship assignment. Your goal should be to get at least a 80% accuracy. If you already have over 80% accuracy, aim to get 85% accuracy.

Team Objectives

- Initially we tried to improve the accuracy of our Multilayer Perceptron Classifier, but after hours of cross validations and numerous revisions to the preprocessing, we could not surpass 72% test accuracy with that algorithm. Lemmatization did not help this model for some reason
- We decided to completely revamp the preprocessing and try to improve NaiveBayes which performed the second best originally, this proved to be much easier and tuneable when adding lemmatization.
  - Lemmatizing with POS (Part of speech) helped improve the accuracy greatly
  - Unlike the MLP, increasing the hashing tf numFeatures improved accuracy

---

Stage 0 - Import Data

In [37]:
```
# Stage 0 Solution
from pyspark.sql import SparkSession
import pandas as pd
```

```
# Start spark session and load train and test data sets
spark = SparkSession.builder \
    .appName("Module_5_Project") \
    .master("local[4]") \
    .config("spark.driver.memory", "20g") \
    .config("spark.executor.memory", "20g") \
    .config("spark.python.worker.memory", "1g") \
    .config("spark.executor.pyspark.memory", "2g") \
    .config("spark.rpc.io.connectionTimeout", "30s") \
    .config("spark.default.parallelism", "16") \
    .config("spark.executor.cores", "8") \
    .config("spark.task.cpus", "1") \
    .config("spark.driver.host","127.0.0.1") \
    .config("spark.driver.bindAddress","127.0.0.1") \
    .getOrCreate()

spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", "true")

df_train = spark.read.csv('./train.csv', header=True, inferSchema=True, quote='"', escape='"')
```

Stage 1 - Data Preparation

- Originally we had removed punctuation, stop words, and did not do lemmatization. After redoing the preprocessing to only do lemmatization this greatly increased the accuracy
- It's possible that either identifying information was removed through the excessive preprocessing, or combining lemmatization and other steps removed information.
- Either way we decided to keep it simple and go with what performed best

In [38]:
```
# Step 1 - Preprocessing
import nltk
from nltk import pos_tag, word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

# Download nltk resources
nltk.download('wordnet')
```

```
nltk.download('own-1.4')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

# 'Switch' function for returing the POS tag
def pos_switch(tag):
    if tag.startswith("J"):
        return wordnet.ADJ
    elif tag.startswith("V"):
        return wordnet.VERB
    elif tag.startswith("N"):
        return wordnet.NOUN
    elif tag.startswith("R"):
        return wordnet.ADV
    else:
        return wordnet.NOUN

# Lemmatizes the input text with POS (Part of speech)
lemmatizer = WordNetLemmatizer()
def lemmatize_text(text):
    # Tokenize words
    words = word_tokenize(text)

    # Get POS tag for tokens
    tags = pos_tag(words)

    # Lemmatize based on token and tag
    lemmatized_text = [lemmatizer.lemmatize(word, pos_switch(pos)) for word, pos in tags]
    return " ".join(lemmatized_text)

# Convert to Pandas DataFrame temporarily. Had immense issues with PySpark UDFs and worker timeouts. This just makes
df_tmp = df_train.toPandas()

# Lemmatize text into new column
df_tmp["processed_text"] = df_tmp["text"].apply(lemmatize_text)

# Convert back to Spark DataFrame
df_lemmatized = spark.createDataFrame(df_tmp)
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\acct_pers\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Error loading own-1.4: Package 'own-1.4' not found in
[nltk_data]     index
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\acct_pers\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\acct_pers\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Stage 2 - Feature Extraction

In [39]:

```python
from pyspark.ml import Pipeline
from pyspark.ml.feature import HashingTF, IDF, Tokenizer

# Define stages for pipeline [Tokenization -> TF -> IDF]
tokenizer = Tokenizer(inputCol="processed_text", outputCol="tokens")            # Converts the pos lemmatized text ir
hashingTF = HashingTF(inputCol='tokens', outputCol='tf', numFeatures=524288)    # Computes a hashing TF on the tokens
idf = IDF(inputCol='tf', outputCol='features', minDocFreq=3)                    # Computes an IDF on the hashing TF c


# Define pipeline with all stages
nb_pipeline = Pipeline(stages=[tokenizer, hashingTF, idf])

# Execute the pipeline with the pos lemmatized text and transform to tfidf
tfidf = nb_pipeline.fit(df_lemmatized).transform(df_lemmatized)
tfidf.select("features").show(5, truncate=False)
```

```
+--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
---------------------------------+
|features
|
+--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
---------------------------------+
|(524288,[8732,19036,20901,38308,91767,100941,117491,131709,141331,145207,148880,167503,214862,219087,248200,261675,2
89720,293094,296332,307548,309043,312145,350734,351861,358033,364180,369251,370685,404383,445483,446082,448456,47166
2,482059,483171,499532,503088,518612],[7.397357266296501,2.036730623534044,3.108592115558026,4.405800364152991,6.2713
46003440277,7.243206586469243,2.6495307799068866,4.2987676073028025,1.1219589976525253,0.5407198584929742,4.261863050
367351,4.706114183510672,6.448276711599355,1.611653523768082,6.704210085736555,4.16523621467828,0.815216713934794
9,1.425457874683359,5.839212648249951,0.044969586283172255,5.427916619830993,2.072722591431091,6.550059405909297,3.04
8155177270663,1.2477762646451427,6.271346003440277,0.7931876316013579,2.0739408656934453,4.007333185232471,3.99061970
425873,3.332613173850691,2.5505489463580355,4.706114183510672,0.5701795602250346,7.579678823090456,3.411464412301899
3,1.60918258241867,3.5055369681858743])|
|(524288,[17046,39275,113673,145207,277457,289720,293094,307548,310592,312145,358033,369251,383545,514987,518612],
[5.77139005191119,7.802822374404665,3.6075018948425623,0.5407198584929742,5.382454245754237,0.8152167139347949,1.4254
57874683359,0.044969586283172255,1.3425267602333695,2.072722591431091,0.31194406616128567,0.7931876316013579,5.061982
3504794645,3.8636707015882665,3.5055369681858743])
|
|(524288,[4106,48648,49120,98424,102006,126466,141331,145207,219087,224255,227860,284514,287361,290482,296004,30454
8,307548,317783,324934,337436,358033,363313,369251,375091,405346,432558,461320,494511,501487,507667,512999,523989,524
014],[4.764270103667746,5.146065467690006,1.9229879559681051,4.376932380152138,5.349664422931245,1.660247478882305
4,1.1219589976525253,0.5407198584929742,1.0744356825120547,3.580660920209216,4.912450616508501,5.349664422931245,3.79
32190406376556,7.802822374404665,1.8967795593506789,2.347501259046964,0.044969586283172255,3.3885100762328153,4.66732
8158475516,6.663388091216301,0.6238881323225713,2.041557464335102,1.5863752632027157,15.370078677496563,2.45809863504
2473,5.349664422931245,6.326915854595088,3.7641667180431533,0.0,4.564143922240285,1.014132202581468,5.692609174058075
5,3.434958053543288])
```

```
|
|(524288,[4181,67416,99179,141331,145207,161773,168342,189996,195453,199693,206227,209164,216689,270873,294536,30454
8,307548,324544,358033,363313,369251,374496,418228,428950,445555,459587,460733,471280,482059,483459,485907,501167,512
999,521193],[5.382454245754237,2.1463935961319347,4.678257229007706,0.841469248239394,0.5407198584929742,6.8865316425
3051,5.439612659594185,5.209435081622595,0.0,2.057018624652727,5.633768674035142,6.746769700155352,5.59180447493611
1,6.991892158188337,5.6926091740580755,2.347501259046964,0.044969586283172255,5.475544668820248,0.3119440661612856
7,2.041557464335102,0.7931876316013579,3.9033781511833796,2.608477598239607,6.991892158188337,5.070079560712084,3.939
4645409578004,3.442913544984403,7.484368643286131,1.1403591204500692,3.8348917370382236,7.101524132381621,8.272826003
650401,1.014132202581468,4.926436858483241])
|
|(524288,[20901,33917,37673,49120,56194,64760,66221,67562,116996,128076,141331,174966,187114,224255,236309,255882,296
338,307548,334519,358033,368093,369251,415567,441976,483837,485907,523989],[1.554296057779013,1.8151148761743958,7.57
9678823090456,5.768963867904315,4.700480365792417,7.802822374404665,6.448276711599355,5.578198822880331,4.21583722797
2069,4.947789982953809,0.5609794988262626,3.307188075484881,1.9177605525329535,1.790330460104608,6.16869184938019
4,6.550059405909297,7.579678823090456,0.044969586283172255,7.685039338748282,0.31194406616128567,5.04598200913302
4,1.5863752632027157,3.6226824520195784,3.548984287944811,1.8544610677141897,3.5507620661908104,5.6926091740580755])
|
+--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
---------------------------------+
only showing top 5 rows
```

Stage 3 - Machine Learning

1. Perform 80/20 train/test split
2. Train Naive Bayes to achieve >80% accuracy

In [49]:
```
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.feature import StringIndexer
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

# Convert authors to numerical labels
```

```
indexer = StringIndexer(inputCol="author", outputCol="label")
# had to break these steps out to grab the indicies.
indexer_model = indexer.fit(tfidf)
df_labeled = indexer_model.transform(tfidf)

# Do a 80/20 split for validations
df_train, df_test = df_labeled.randomSplit([0.8, 0.2], seed=42)

# Setup model and evaluators
naive_bayes = NaiveBayes()
accuracy_evaluator = MulticlassClassificationEvaluator(metricName='accuracy')
f1_evaluator = MulticlassClassificationEvaluator(metricName='f1')

# Setup hyper param to test, here we're just adjusting the smoothing for NB
params = ParamGridBuilder().addGrid(
    naive_bayes.smoothing, [0.5, 0.7, 0.9, 1.0, 1.1, 1.3, 1.5]
    ).build()

# Set cross validator params
cv = CrossValidator(
    estimator=naive_bayes,
    estimatorParamMaps=params,
    evaluator=accuracy_evaluator,
    parallelism=8
    )

# Train the cross validator to find the best model
model = cv.fit(df_train)
best_model = model.bestModel

# Predict on the test set
nb_prediction = best_model.transform(df_test)

# Evaluate
print('Test Accuracy:', accuracy_evaluator.evaluate(nb_prediction))
print('Test F1 Score:', f1_evaluator.evaluate(nb_prediction))
print(f'Best smoothing parameter: {best_model.getSmoothing()}')
```

```
Test Accuracy: 0.8351962568234989
Test F1 Score: 0.8352423061400291
Best smoothing parameter: 1.1
```

Stage 4 - Evaluation and Visualization

In [74]:
```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from pyspark.sql.functions import col, lit

nb_test_predictions = best_model.transform(df_test)

nb_evaluator = MulticlassClassificationEvaluator(labelCol='label', predictionCol='prediction', metricName='f1')
acc_score = nb_evaluator.evaluate(nb_test_predictions)
print(f"Naive Bayes Accuracy Score: {acc_score}")

label_mapping = indexer_model.labels

nb_test_predictions = nb_test_predictions.withColumn(
    "predicted_author",
    col("prediction").cast("int").cast("string")
).replace(
    to_replace={str(i): label for i, label in enumerate(label_mapping)},
    subset=["predicted_author"]
)

# Create confusion matrix
nb_test_predictions_pd = nb_test_predictions.select('predicted_author', 'author').toPandas()
nb_conf_mat = confusion_matrix(
    nb_test_predictions_pd['author'],
    nb_test_predictions_pd['predicted_author'],
    labels=label_mapping
)

# Wrap in DataFrame with labels
conf_df = pd.DataFrame(nb_conf_mat, index=label_mapping, columns=label_mapping)

# Plot
plt.figure(figsize=(8, 6))
sns.heatmap(conf_df, annot=True, fmt='d', cmap='Blues')
plt.title('Naive Bayes Confusion Matrix')
plt.xlabel('Predicted Author')
plt.ylabel('True Author')
```

```
plt.show()
```

```
Naive Bayes Accuracy Score: 0.8352423061400291
```