classify

April 26, 2024

[ ]:
```
!unzip AP3-zipped.zip
%cd AP3
```

```
Archive:  AP3-zipped.zip
  inflating: AP3/classify.ipynb
  inflating: AP3/jeopardy.csv
  inflating: AP3/model.csv
   creating: AP3/splits/
  inflating: AP3/splits/dev.txt
  inflating: AP3/splits/test.txt
  inflating: AP3/splits/train.txt
   creating: AP3/transformers/
/content/AP3
```

[ ]:
```
import pandas as pd
import numpy as np
from transformers import BertTokenizer, TFBertForSequenceClassification,␣
 ↪DistilBertTokenizer, TFDistilBertForSequenceClassification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
from tf_keras.callbacks import CSVLogger, ModelCheckpoint
import tf_keras
import warnings
warnings.filterwarnings("ignore")


from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from scipy import stats
import math
```

```
[ ]: jeopardy = pd.read_csv('jeopardy.csv')
train, other = train_test_split(jeopardy, test_size=0.4, random_state=42,␣
 ↪stratify=jeopardy['class'])
dev, test = train_test_split(other, test_size=0.5, random_state=42,␣
 ↪stratify=other['class'])
train.to_csv('splits/train.txt', index=False)
dev.to_csv('splits/dev.txt', index=False)
test.to_csv('splits/test.txt', index=False)
```

0.1 BERT

```
[ ]: # Function to preprocess and encode datasets
def prepare_data(data):
    data['combined_text'] = data['Category'] + " [SEP] " + data['Question'] + "␣
 ↪[SEP] " + data['Answer']
    # Ensure to use the 'text' parameter for the tokenizer
    input_data = tokenizer(text=data['combined_text'].tolist(),␣
 ↪padding="max_length", truncation=True, max_length=128, return_tensors="tf")
    labels = label_encoder.transform(data['class'])  # Transform labels using␣
 ↪the fitted label encoder
    return input_data['input_ids'], tf.keras.utils.to_categorical(labels,␣
 ↪num_classes=len(label_encoder.classes_))


model_name = 'distilbert-base-uncased'
# Initialize and fit label encoder on all possible labels
label_encoder = LabelEncoder()
all_data = pd.concat([train, dev, test])
label_encoder.fit(all_data['class'])

# Initialize tokenizer
tokenizer = DistilBertTokenizer.from_pretrained(model_name)
# Initialize model
model = TFDistilBertForSequenceClassification.from_pretrained(model_name,␣
 ↪num_labels=len(label_encoder.classes_))

# Prepare datasets
train_inputs, train_labels = prepare_data(train)
dev_inputs, dev_labels = prepare_data(dev)
test_inputs, test_labels = prepare_data(test)

# Create TensorFlow datasets
train_dataset = tf.data.Dataset.from_tensor_slices((train_inputs,␣
 ↪train_labels)).shuffle(len(train_labels)).batch(32)
dev_dataset = tf.data.Dataset.from_tensor_slices((dev_inputs, dev_labels)).
 ↪batch(32)
```

```python
test_dataset = tf.data.Dataset.from_tensor_slices((test_inputs, test_labels)).
 ↪batch(32)



# Compile model
loss_fn = tf.keras.losses.CategoricalCrossentropy(from_logits=True)
optimizer = tf_keras.optimizers.legacy.Adam(learning_rate=1e-5)
csv_logger = CSVLogger("model.csv")
checkpoint = ModelCheckpoint("model.keras", monitor='val_loss', verbose=1,␣
 ↪save_best_only=True, save_weights_only=False, mode='auto', save_freq='epoch')

model.compile(optimizer=optimizer, loss=loss_fn, metrics=['accuracy'])

# Train model
model.fit(train_dataset, epochs=25, validation_data=dev_dataset,␣
 ↪callbacks=[csv_logger, checkpoint])

# Evaluate model on the test dataset
test_loss, test_accuracy = model.evaluate(test_dataset)
print(f"Test Loss: {test_loss}, Test Accuracy: {test_accuracy}")
```

```
Some weights of the PyTorch model were not used when initializing the TF 2.0
model TFDistilBertForSequenceClassification: ['vocab_layer_norm.weight',
'vocab_projector.bias', 'vocab_transform.bias', 'vocab_layer_norm.bias',
'vocab_transform.weight']
- This IS expected if you are initializing TFDistilBertForSequenceClassification
from a PyTorch model trained on another task or with another architecture (e.g.
initializing a TFBertForSequenceClassification model from a BertForPreTraining
model).
- This IS NOT expected if you are initializing
TFDistilBertForSequenceClassification from a PyTorch model that you expect to be
exactly identical (e.g. initializing a TFBertForSequenceClassification model
from a BertForSequenceClassification model).
Some weights or buffers of the TF 2.0 model
TFDistilBertForSequenceClassification were not initialized from the PyTorch
model and are newly initialized: ['pre_classifier.weight',
'pre_classifier.bias', 'classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

```
Epoch 1/25
10/10 [==============================] - ETA: 0s - loss: 1.2839 - accuracy:
0.5333
Epoch 1: val_loss improved from inf to 1.11195, saving model to model.keras
10/10 [==============================] - 50s 1s/step - loss: 1.2839 - accuracy:
0.5333 - val_loss: 1.1119 - val_accuracy: 0.6400
Epoch 2/25
```

```
10/10 [==============================] - ETA: 0s - loss: 1.0606 - accuracy:
0.6433
Epoch 2: val_loss improved from 1.11195 to 1.02400, saving model to model.keras
10/10 [==============================] - 6s 595ms/step - loss: 1.0606 -
accuracy: 0.6433 - val_loss: 1.0240 - val_accuracy: 0.6400
Epoch 3/25
10/10 [==============================] - ETA: 0s - loss: 1.0266 - accuracy:
0.6433
Epoch 3: val_loss improved from 1.02400 to 1.01152, saving model to model.keras
10/10 [==============================] - 6s 563ms/step - loss: 1.0266 -
accuracy: 0.6433 - val_loss: 1.0115 - val_accuracy: 0.6400
Epoch 4/25
10/10 [==============================] - ETA: 0s - loss: 0.9960 - accuracy:
0.6433
Epoch 4: val_loss improved from 1.01152 to 1.00505, saving model to model.keras
10/10 [==============================] - 6s 562ms/step - loss: 0.9960 -
accuracy: 0.6433 - val_loss: 1.0050 - val_accuracy: 0.6400
Epoch 5/25
10/10 [==============================] - ETA: 0s - loss: 1.0008 - accuracy:
0.6433
Epoch 5: val_loss improved from 1.00505 to 0.99658, saving model to model.keras
10/10 [==============================] - 6s 602ms/step - loss: 1.0008 -
accuracy: 0.6433 - val_loss: 0.9966 - val_accuracy: 0.6400
Epoch 6/25
10/10 [==============================] - ETA: 0s - loss: 0.9797 - accuracy:
0.6433
Epoch 6: val_loss improved from 0.99658 to 0.99170, saving model to model.keras
10/10 [==============================] - 6s 602ms/step - loss: 0.9797 -
accuracy: 0.6433 - val_loss: 0.9917 - val_accuracy: 0.6400
Epoch 7/25
10/10 [==============================] - ETA: 0s - loss: 0.9679 - accuracy:
0.6433
Epoch 7: val_loss improved from 0.99170 to 0.95689, saving model to model.keras
10/10 [==============================] - 6s 629ms/step - loss: 0.9679 -
accuracy: 0.6433 - val_loss: 0.9569 - val_accuracy: 0.6400
Epoch 8/25
10/10 [==============================] - ETA: 0s - loss: 0.9466 - accuracy:
0.6433
Epoch 8: val_loss improved from 0.95689 to 0.91380, saving model to model.keras
10/10 [==============================] - 6s 582ms/step - loss: 0.9466 -
accuracy: 0.6433 - val_loss: 0.9138 - val_accuracy: 0.6400
Epoch 9/25
10/10 [==============================] - ETA: 0s - loss: 0.8884 - accuracy:
0.6433
Epoch 9: val_loss improved from 0.91380 to 0.88861, saving model to model.keras
10/10 [==============================] - 7s 687ms/step - loss: 0.8884 -
accuracy: 0.6433 - val_loss: 0.8886 - val_accuracy: 0.6400
Epoch 10/25
```

```
10/10 [==============================] - ETA: 0s - loss: 0.8598 - accuracy:
0.6433
Epoch 10: val_loss improved from 0.88861 to 0.85342, saving model to model.keras
10/10 [==============================] - 6s 583ms/step - loss: 0.8598 -
accuracy: 0.6433 - val_loss: 0.8534 - val_accuracy: 0.6400
Epoch 11/25
10/10 [==============================] - ETA: 0s - loss: 0.7763 - accuracy:
0.6733
Epoch 11: val_loss improved from 0.85342 to 0.79642, saving model to model.keras
10/10 [==============================] - 6s 591ms/step - loss: 0.7763 -
accuracy: 0.6733 - val_loss: 0.7964 - val_accuracy: 0.6900
Epoch 12/25
10/10 [==============================] - ETA: 0s - loss: 0.7234 - accuracy:
0.7233
Epoch 12: val_loss improved from 0.79642 to 0.76068, saving model to model.keras
10/10 [==============================] - 6s 619ms/step - loss: 0.7234 -
accuracy: 0.7233 - val_loss: 0.7607 - val_accuracy: 0.7200
Epoch 13/25
10/10 [==============================] - ETA: 0s - loss: 0.6691 - accuracy:
0.7933
Epoch 13: val_loss improved from 0.76068 to 0.71401, saving model to model.keras
10/10 [==============================] - 6s 603ms/step - loss: 0.6691 -
accuracy: 0.7933 - val_loss: 0.7140 - val_accuracy: 0.7500
Epoch 14/25
10/10 [==============================] - ETA: 0s - loss: 0.5826 - accuracy:
0.8100
Epoch 14: val_loss did not improve from 0.71401
10/10 [==============================] - 5s 488ms/step - loss: 0.5826 -
accuracy: 0.8100 - val_loss: 0.7265 - val_accuracy: 0.7700
Epoch 15/25
10/10 [==============================] - ETA: 0s - loss: 0.5301 - accuracy:
0.8367
Epoch 15: val_loss did not improve from 0.71401
10/10 [==============================] - 5s 472ms/step - loss: 0.5301 -
accuracy: 0.8367 - val_loss: 0.7191 - val_accuracy: 0.7000
Epoch 16/25
10/10 [==============================] - ETA: 0s - loss: 0.4406 - accuracy:
0.8900
Epoch 16: val_loss improved from 0.71401 to 0.63880, saving model to model.keras
10/10 [==============================] - 6s 636ms/step - loss: 0.4406 -
accuracy: 0.8900 - val_loss: 0.6388 - val_accuracy: 0.7600
Epoch 17/25
10/10 [==============================] - ETA: 0s - loss: 0.3586 - accuracy:
0.9067
Epoch 17: val_loss did not improve from 0.63880
10/10 [==============================] - 5s 464ms/step - loss: 0.3586 -
accuracy: 0.9067 - val_loss: 0.6651 - val_accuracy: 0.7200
Epoch 18/25
```

```
10/10 [==============================] - ETA: 0s - loss: 0.3130 - accuracy:
0.9300
Epoch 18: val_loss did not improve from 0.63880
10/10 [==============================] - 5s 466ms/step - loss: 0.3130 -
accuracy: 0.9300 - val_loss: 0.6528 - val_accuracy: 0.7700
Epoch 19/25
10/10 [==============================] - ETA: 0s - loss: 0.2593 - accuracy:
0.9500
Epoch 19: val_loss did not improve from 0.63880
10/10 [==============================] - 5s 463ms/step - loss: 0.2593 -
accuracy: 0.9500 - val_loss: 0.7041 - val_accuracy: 0.7700
Epoch 20/25
10/10 [==============================] - ETA: 0s - loss: 0.2363 - accuracy:
0.9533
Epoch 20: val_loss did not improve from 0.63880
10/10 [==============================] - 5s 464ms/step - loss: 0.2363 -
accuracy: 0.9533 - val_loss: 0.6579 - val_accuracy: 0.7600
Epoch 21/25
10/10 [==============================] - ETA: 0s - loss: 0.1943 - accuracy:
0.9567
Epoch 21: val_loss did not improve from 0.63880
10/10 [==============================] - 5s 462ms/step - loss: 0.1943 -
accuracy: 0.9567 - val_loss: 0.7152 - val_accuracy: 0.7500
Epoch 22/25
10/10 [==============================] - ETA: 0s - loss: 0.1724 - accuracy:
0.9633
Epoch 22: val_loss did not improve from 0.63880
10/10 [==============================] - 5s 459ms/step - loss: 0.1724 -
accuracy: 0.9633 - val_loss: 0.6850 - val_accuracy: 0.8200
Epoch 23/25
10/10 [==============================] - ETA: 0s - loss: 0.1544 - accuracy:
0.9767
Epoch 23: val_loss did not improve from 0.63880
10/10 [==============================] - 5s 479ms/step - loss: 0.1544 -
accuracy: 0.9767 - val_loss: 0.7428 - val_accuracy: 0.8100
Epoch 24/25
10/10 [==============================] - ETA: 0s - loss: 0.1452 - accuracy:
0.9800
Epoch 24: val_loss did not improve from 0.63880
10/10 [==============================] - 5s 457ms/step - loss: 0.1452 -
accuracy: 0.9800 - val_loss: 0.7318 - val_accuracy: 0.8000
Epoch 25/25
10/10 [==============================] - ETA: 0s - loss: 0.1281 - accuracy:
0.9767
Epoch 25: val_loss did not improve from 0.63880
10/10 [==============================] - 5s 456ms/step - loss: 0.1281 -
accuracy: 0.9767 - val_loss: 0.7453 - val_accuracy: 0.8000
4/4 [==============================] - 0s 112ms/step - loss: 0.7547 - accuracy:
```

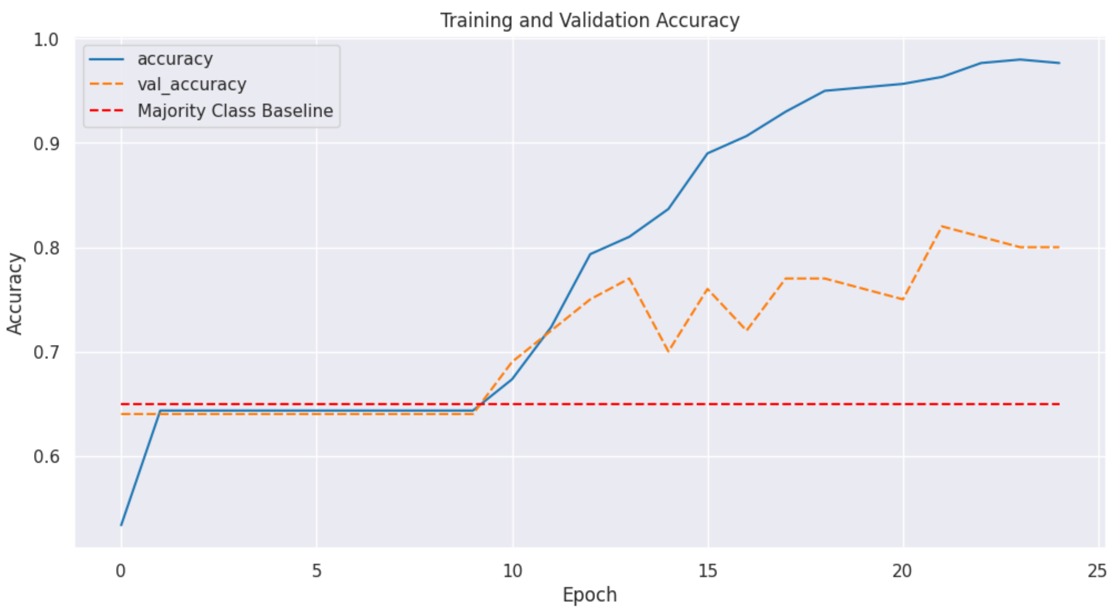
```
0.7800
Test Loss: 0.7546696662902832, Test Accuracy: 0.7799999713897705
```

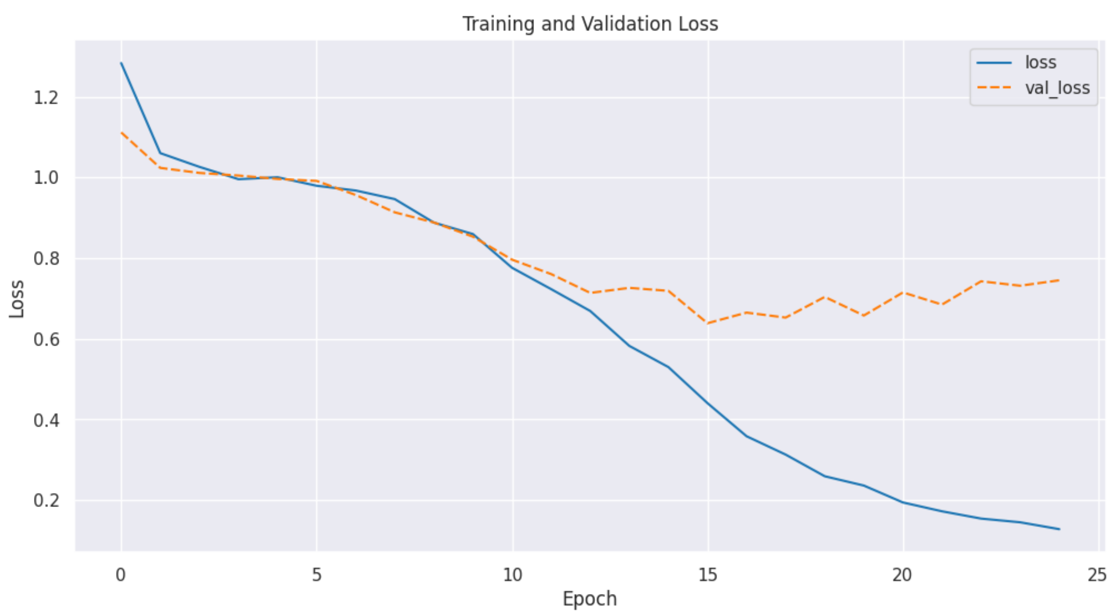
[ ]:
```
majority_class = test['class'].value_counts().idxmax()
majority_class_accuracy = (test['class'] == majority_class).mean()
print(f"Majority Class Baseline: {majority_class_accuracy}")
```

```
Majority Class Baseline: 0.65
```

[ ]:
```
# Confidence interval function from example notebooks
def confidence_intervals(accuracy, n, significance_level):
    critical_value=(1-significance_level)/2
    z_alpha=-1*stats.norm.ppf(critical_value)
    se=math.sqrt((accuracy*(1-accuracy))/n)
    return accuracy-(se*z_alpha), accuracy+(se*z_alpha)
```

[ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style='darkgrid', palette="tab10", context='notebook')
performance = pd.read_csv('model.csv')
performance['epoch'] = performance.index + 1
plt.figure(figsize=(12, 6))
sns.lineplot(data=performance[['accuracy', 'val_accuracy']])
plt.hlines(majority_class_accuracy, color='red', linestyles='dashed',␣
 ↪label='Majority Class Baseline', xmin=0, xmax=len(performance)-1)
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()
plt.figure(figsize=(12, 6))
sns.lineplot(data=performance[['loss', 'val_loss']])
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.show()
```

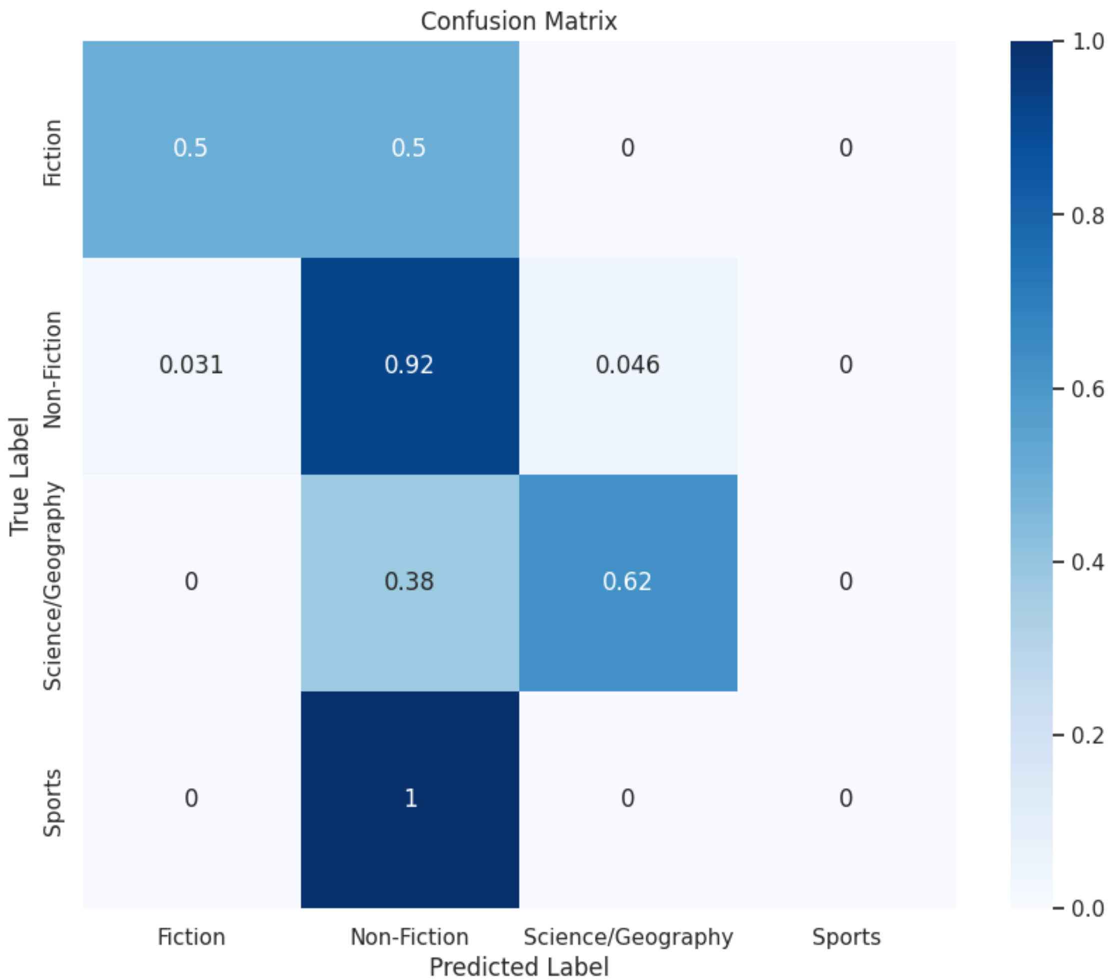




[ ]:
```
preds = model.predict(test_inputs)
classes = label_encoder.classes_
y_hat = classes[np.argmax(preds.logits, axis=1)]
y = label_encoder.inverse_transform(np.argmax(test_labels, axis=1))
```

```
4/4 [==============================] - 6s 117ms/step
```

[ ]:
```python
# make confusion matrix using sklearn
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y, y_hat, labels=classes, normalize='true')
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, cmap='Blues', xticklabels=classes,␣
 ↪yticklabels=classes)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



[ ]:
```python
print(f"Test Accuracy: {test_accuracy}")
```

```
Test Accuracy: 0.7799999713897705
```

[ ]:
```
confidence_intervals(test_accuracy, len(test_inputs), 0.95)
```

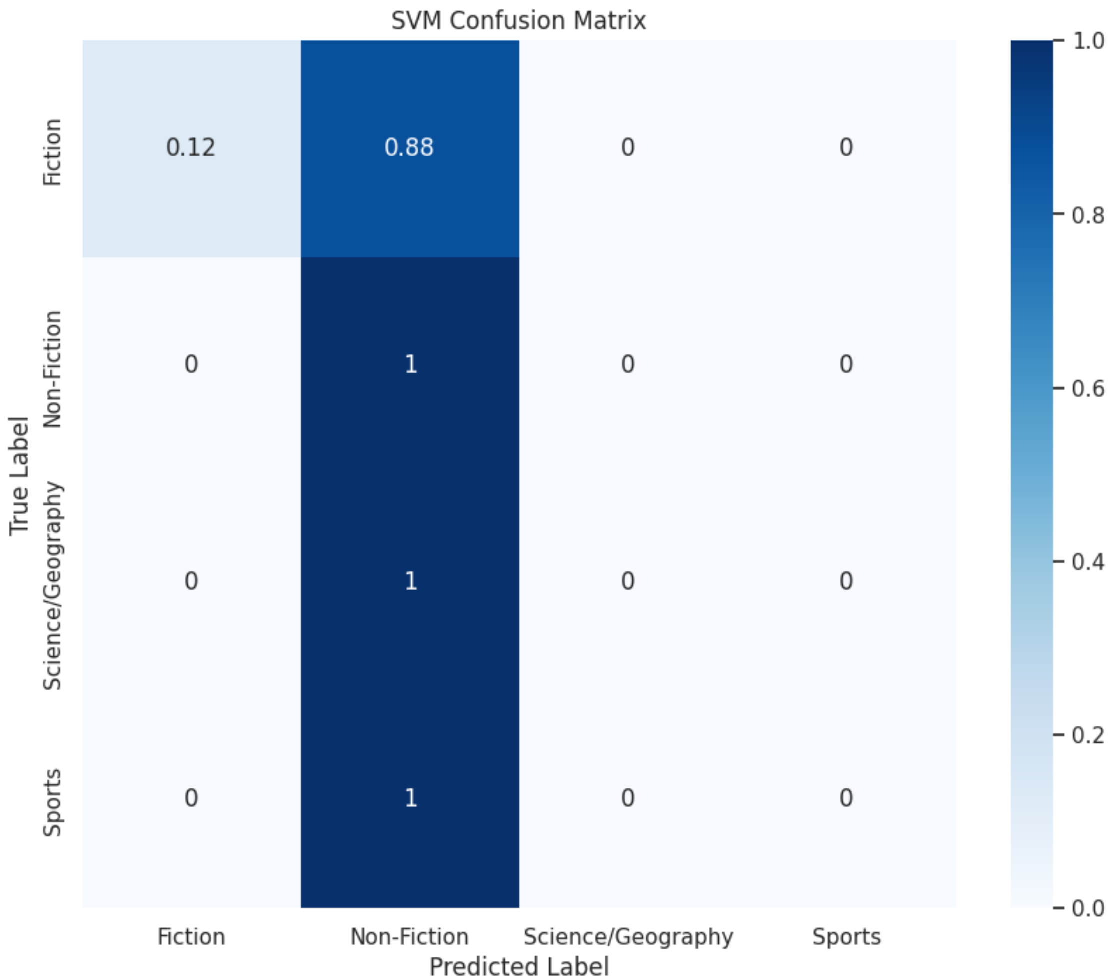
[ ]: (0.6988091840319723, 0.8611907587475687)

0.2 SVM

[ ]:
```
for data in [train, dev, test]:
    data['text'] = data['Category'] + " [SEP] " + data['Question'] + " [SEP] "␣
 ↪+ data['Answer']

# Prepare the features and labels for each set
X_train = train['text']
y_train = train['class']
X_dev = dev['text']
y_dev = dev['class']
X_test = test['text']
y_test = test['class']

# Create a TF-IDF vectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_features=10000)

# Transform features into TF-IDF coefficients
X_train = vectorizer.fit_transform(X_train)
X_dev = vectorizer.transform(X_dev)
X_test = vectorizer.transform(X_test)

# Initialize the SVM classifier
svm = SVC(kernel='sigmoid', random_state=42)

# Train the model using the training set
svm.fit(X_train, y_train)

# Validate the model using the development set
dev_predictions = svm.predict(X_dev)
dev_accuracy = accuracy_score(y_dev, dev_predictions)
print(f'Development Set Accuracy: {dev_accuracy}')
print(classification_report(y_dev, dev_predictions))

# Finally, evaluate the model using the test set
test_predictions = svm.predict(X_test)
test_accuracy = accuracy_score(y_test, test_predictions)
print(f'Test Set Accuracy: {test_accuracy}')
print(classification_report(y_test, test_predictions))
cm = confusion_matrix(y_test, test_predictions, labels=classes,␣
 ↪normalize='true')
plt.figure(figsize=(10, 8))
```

```
sns.heatmap(cm, annot=True, cmap='Blues', xticklabels=classes,␣
 ↪yticklabels=classes)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('SVM Confusion Matrix')
plt.show()
```

```
Development Set Accuracy: 0.65
                   precision    recall  f1-score   support

          Fiction       1.00      0.06      0.11        17
      Non-Fiction       0.65      1.00      0.79        64
Science/Geography       0.00      0.00      0.00        15
           Sports       0.00      0.00      0.00         4

         accuracy                           0.65       100
        macro avg       0.41      0.26      0.22       100
     weighted avg       0.58      0.65      0.52       100

Test Set Accuracy: 0.67
                   precision    recall  f1-score   support

          Fiction       1.00      0.12      0.22        16
      Non-Fiction       0.66      1.00      0.80        65
Science/Geography       0.00      0.00      0.00        16
           Sports       0.00      0.00      0.00         3

         accuracy                           0.67       100
        macro avg       0.42      0.28      0.25       100
     weighted avg       0.59      0.67      0.55       100
```



[ ]:
```
confidence_intervals(test_accuracy, len(test_inputs), 0.95)
```

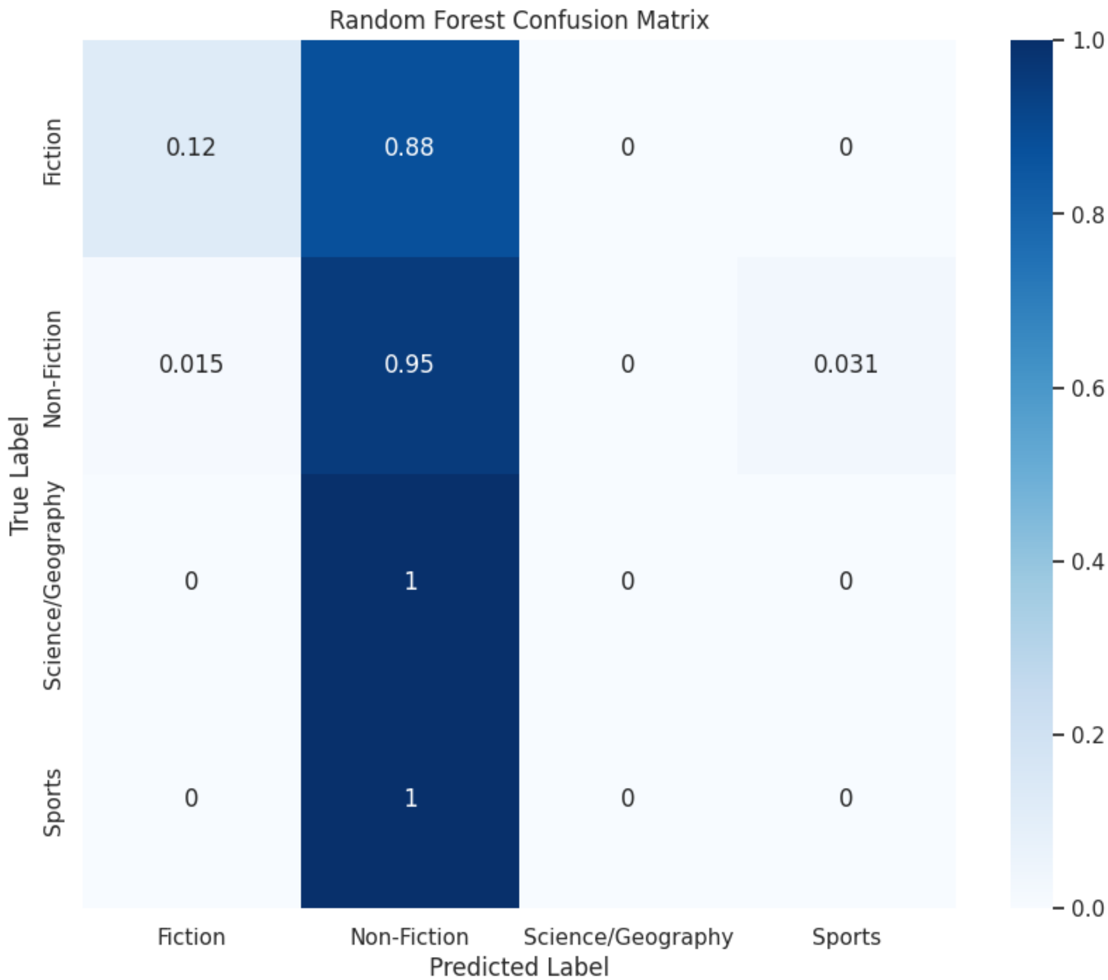
[ ]: (0.5778400007999419, 0.7621599992000582)

0.3 Random Forests

[ ]:
```
random_forest = RandomForestClassifier(n_estimators=10, random_state=42)

# Train the model using the training set
random_forest.fit(X_train, y_train)

# Validate the model using the development set
dev_predictions = random_forest.predict(X_dev)
dev_accuracy = accuracy_score(y_dev, dev_predictions)
print(f'Development Set Accuracy: {dev_accuracy}')
print(classification_report(y_dev, dev_predictions))
```

```python
# Finally, evaluate the model using the test set
test_predictions = random_forest.predict(X_test)
test_accuracy = accuracy_score(y_test, test_predictions)
print(f'Test Set Accuracy: {test_accuracy}')
print(classification_report(y_test, test_predictions))

cm = confusion_matrix(y_test, test_predictions, labels=classes,␣
 ↪normalize='true')
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, cmap='Blues', xticklabels=classes,␣
 ↪yticklabels=classes)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Random Forest Confusion Matrix')
plt.show()
```

```
Development Set Accuracy: 0.66
                   precision    recall  f1-score   support

          Fiction       1.00      0.06      0.11        17
      Non-Fiction       0.65      1.00      0.79        64
Science/Geography       1.00      0.07      0.12        15
           Sports       0.00      0.00      0.00         4

         accuracy                           0.66       100
        macro avg       0.66      0.28      0.26       100
     weighted avg       0.74      0.66      0.54       100

Test Set Accuracy: 0.64
                   precision    recall  f1-score   support

          Fiction       0.67      0.12      0.21        16
      Non-Fiction       0.65      0.95      0.78        65
Science/Geography       0.00      0.00      0.00        16
           Sports       0.00      0.00      0.00         3

         accuracy                           0.64       100
        macro avg       0.33      0.27      0.25       100
     weighted avg       0.53      0.64      0.54       100
```



[ ]:
```
confidence_intervals(test_accuracy, len(test_inputs), 0.95)
```

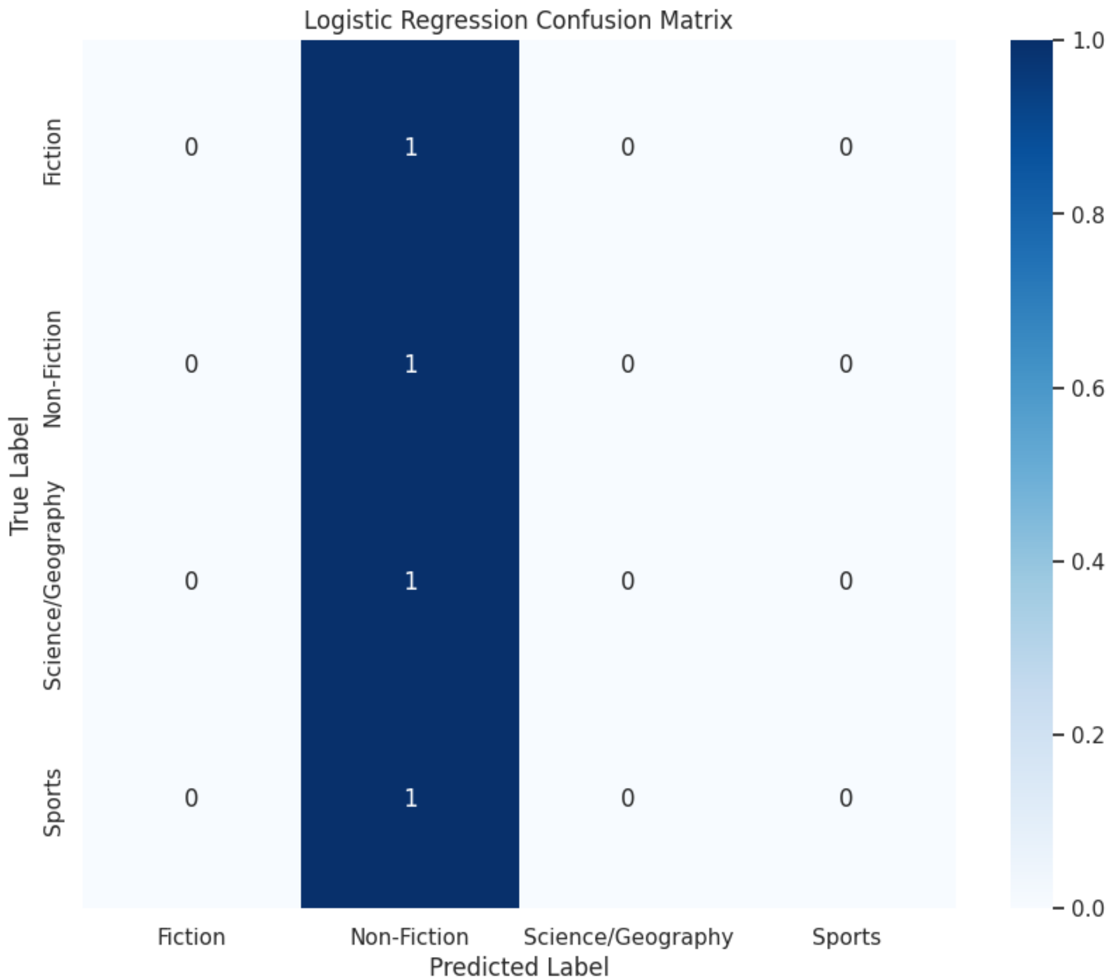
[ ]: (0.5459217287420775, 0.7340782712579226)

0.4 Logistic Regression

[ ]:
```
logistic_regression = LogisticRegression(max_iter=1000)

# Train the model using the training set
logistic_regression.fit(X_train, y_train)

# Validate the model using the development set
dev_predictions = logistic_regression.predict(X_dev)
dev_accuracy = accuracy_score(y_dev, dev_predictions)
print(f'Development Set Accuracy: {dev_accuracy}')
print(classification_report(y_dev, dev_predictions))
```

```python
# Finally, evaluate the model using the test set
test_predictions = logistic_regression.predict(X_test)
test_accuracy = accuracy_score(y_test, test_predictions)
print(f'Test Set Accuracy: {test_accuracy}')
print(classification_report(y_test, test_predictions))

cm = confusion_matrix(y_test, test_predictions, labels=classes,␣
 ↪normalize='true')
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, cmap='Blues', xticklabels=classes,␣
 ↪yticklabels=classes)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Logistic Regression Confusion Matrix')
plt.show()
```

```
Development Set Accuracy: 0.64
                   precision    recall  f1-score   support

          Fiction       0.00      0.00      0.00        17
      Non-Fiction       0.64      1.00      0.78        64
Science/Geography       0.00      0.00      0.00        15
           Sports       0.00      0.00      0.00         4

         accuracy                           0.64       100
        macro avg       0.16      0.25      0.20       100
     weighted avg       0.41      0.64      0.50       100

Test Set Accuracy: 0.65
                   precision    recall  f1-score   support

          Fiction       0.00      0.00      0.00        16
      Non-Fiction       0.65      1.00      0.79        65
Science/Geography       0.00      0.00      0.00        16
           Sports       0.00      0.00      0.00         3

         accuracy                           0.65       100
        macro avg       0.16      0.25      0.20       100
     weighted avg       0.42      0.65      0.51       100
```



[ ]:
```
confidence_intervals(test_accuracy, len(test_inputs), 0.95)
```

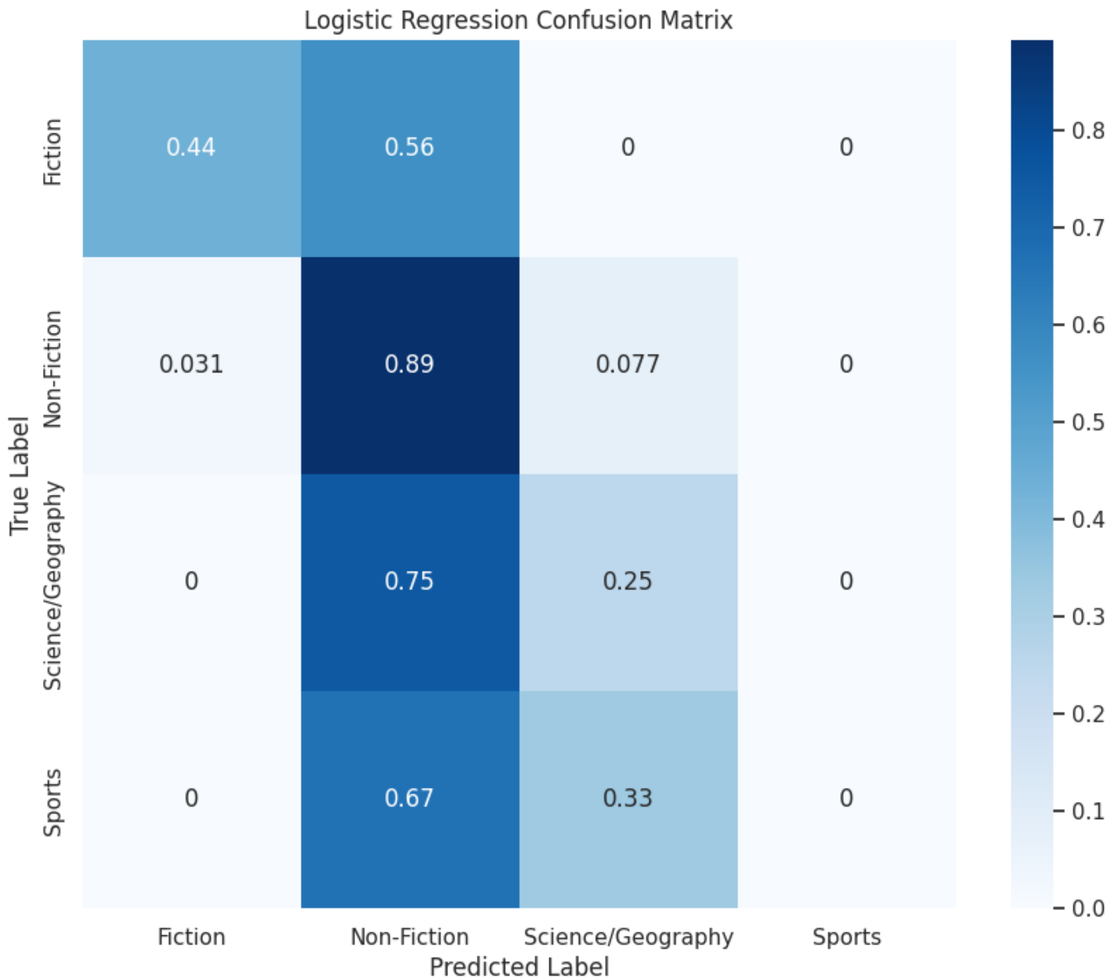
[ ]: (0.5565156760890944, 0.7434843239109057)

0.5 Balanced Logistic

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

logistic_regression = LogisticRegression(max_iter=1000, class_weight='balanced')

# Train the model using the training set
logistic_regression.fit(X_train, y_train)

# Validate the model using the development set
dev_predictions = logistic_regression.predict(X_dev)
dev_accuracy = accuracy_score(y_dev, dev_predictions)
```

```python
print(f'Development Set Accuracy: {dev_accuracy}')
print(classification_report(y_dev, dev_predictions))

# Finally, evaluate the model using the test set
test_predictions = logistic_regression.predict(X_test)
test_accuracy = accuracy_score(y_test, test_predictions)
print(f'Test Set Accuracy: {test_accuracy}')
print(classification_report(y_test, test_predictions))

cm = confusion_matrix(y_test, test_predictions, labels=classes,␣
 ↪normalize='true')
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, cmap='Blues', xticklabels=classes,␣
 ↪yticklabels=classes)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Logistic Regression Confusion Matrix')
plt.show()
```

```
Development Set Accuracy: 0.6
                   precision    recall  f1-score   support

          Fiction       0.25      0.18      0.21        17
      Non-Fiction       0.68      0.81      0.74        64
Science/Geography       0.50      0.33      0.40        15
           Sports       0.00      0.00      0.00         4

         accuracy                           0.60       100
        macro avg       0.36      0.33      0.34       100
     weighted avg       0.55      0.60      0.57       100

Test Set Accuracy: 0.69
                   precision    recall  f1-score   support

          Fiction       0.78      0.44      0.56        16
      Non-Fiction       0.72      0.89      0.79        65
Science/Geography       0.40      0.25      0.31        16
           Sports       0.00      0.00      0.00         3

         accuracy                           0.69       100
        macro avg       0.47      0.39      0.42       100
     weighted avg       0.65      0.69      0.66       100
```



[ ]:
```
confidence_intervals(test_accuracy, len(test_inputs), 0.95)
```

[ ]: (0.5993529900246857, 0.7806470099753142)

[ ]:
```

```