who_is_speaking_lotr_final_ML

April 30, 2022

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import string


#ML
from sklearn.ensemble import RandomForestClassifier
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer


from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score


from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neural_network import MLPClassifier

from sklearn.model_selection import RandomizedSearchCV
#for execution time
import time
```

[2]:
```
df = pd.read_csv('../Data/lotr/prepped_data.csv',dtype=str)
```

[3]:
```
top10_chars =␣
 ↪['GANDALF','SAM','FRODO','ARAGORN','GOLLUM','PIPPIN','MERRY','GIMLI','THEODEN','FARAMIR']
```

[4]:
```
# X = df.dialog.to_list()
# count_vect = CountVectorizer()
# X = count_vect.fit_transform(X)
```

[5]:
```
def best_model_stats(model,X_train, X_test, y_train, y_test):
    start_time = time.time()
    model.fit(X_train, y_train)
    labels = model.predict(X_test)

    print(y_train.columns.to_list())
    mat = confusion_matrix(y_test.values.argmax(axis=1), labels.argmax(axis=1))


    sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
                xticklabels=y_train.columns.to_list(), yticklabels=y_train.
 ↪columns.to_list())

    plt.xlabel('true label')
    plt.ylabel('predicted label')
    plt.show()
    print('Accuracy score:', accuracy_score(y_test, labels))

    print("Execution time: %s seconds " % (time.time() - start_time))
```

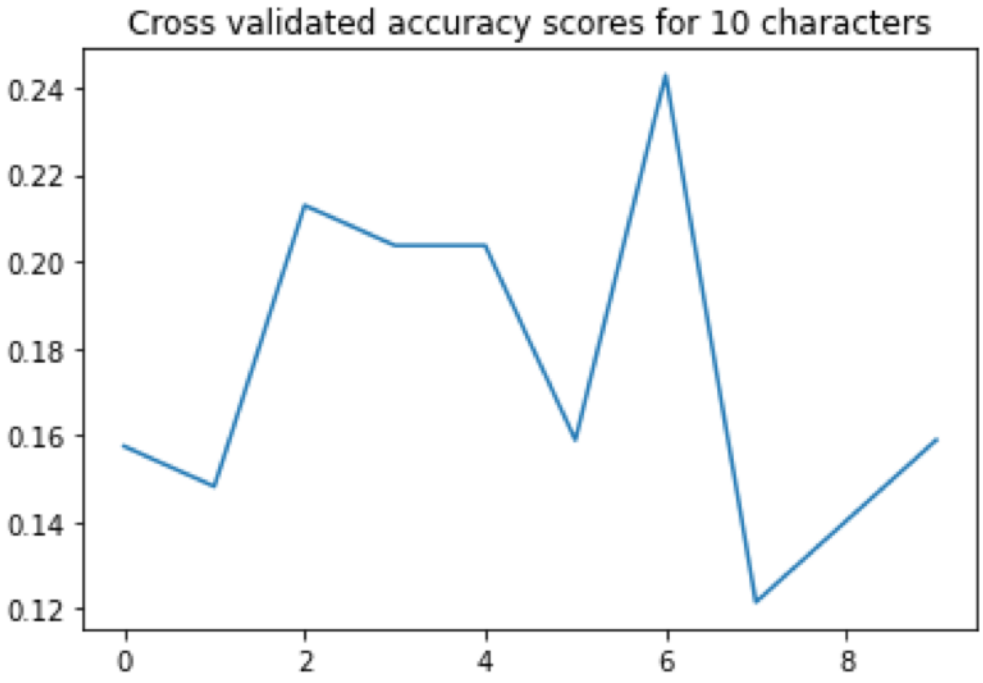
[6]:
```
def get_best_params(X_train, y_train):
    # Number of trees in random forest
    n_estimators = [int(x) for x in np.linspace(start = 100, stop = 2000, num =␣
 ↪10)] # Number of features to consider at every split
    max_features = ['auto', 'sqrt']
    # Maximum number of levels in tree
    max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
    max_depth.append(None)
    # Minimum number of samples required to split a node
    min_samples_split = [2, 5, 10]
    # Minimum number of samples required at each leaf node
    min_samples_leaf = [1, 2, 4]
    # Method of selecting samples for training each tree
    # bootstrap = [True, False]
    oob_score = [True, False]
    class_weight = ['balanced', None]
    # Create the random grid
    random_grid = {'n_estimators': n_estimators,
                   'max_features': max_features,
                   'max_depth': max_depth,
                   'min_samples_split': min_samples_split,
                   'min_samples_leaf': min_samples_leaf,
                   'oob_score': oob_score,
                   'class_weight': class_weight
```

```
                }

    rf_random = RandomizedSearchCV(estimator=RandomForestClassifier(),
                                   param_distributions=random_grid,
                                   n_iter=100,
                                   cv=5,
                                   random_state=42,
                                   n_jobs = -1)
    rf_random.fit(X_train, y_train)

    return rf_random.best_params_
```

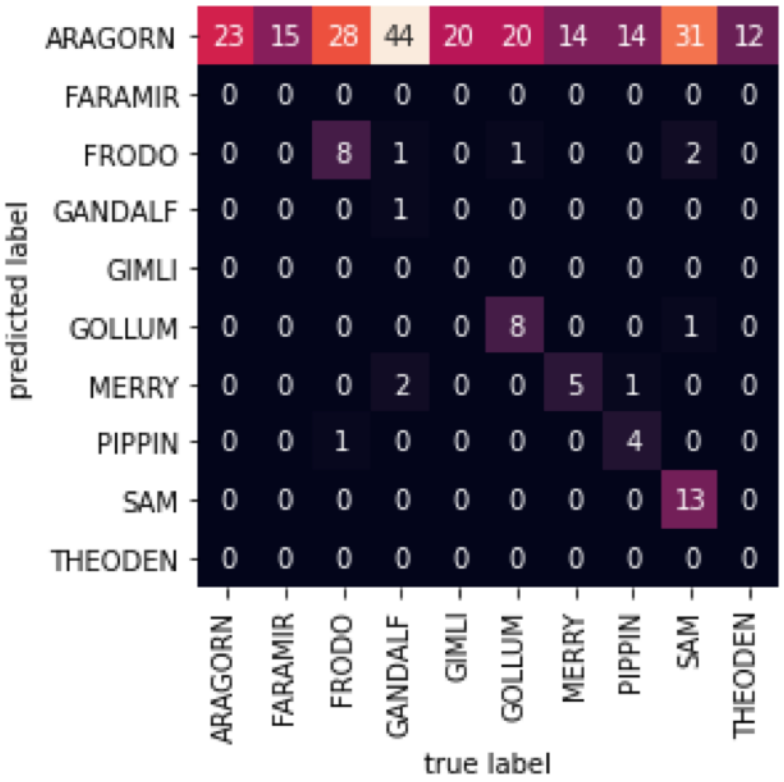
[7]:
```
def get_cv_plot(model, X_train, y_train, num_chars):
    scores = cross_val_score(model, X_train, y_train, scoring='accuracy',␣
 ↪cv=10, n_jobs=-1)
    x = np.arange(10)
    y = scores
    plt.title(f"Cross validated accuracy scores for {num_chars} characters")
    plt.plot(x,y)
    plt.show()
```

[8]:
```
for num_chars in range(10,1,-1):
    # Preprocess and generate train-test data
    df.dropna(inplace=True)
    df = df[df.char.isin(top10_chars[:num_chars])]
    X = df.dialog.to_list()
    tf_idf_vect = TfidfVectorizer()
    X = tf_idf_vect.fit_transform(X)
    y = df['char']
    y = pd.get_dummies(y)
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=.
 ↪8,random_state=1)

    # Get best parameters for random forest model and then create it
    best_params = get_best_params(X_train, y_train)
    best_rf = RandomForestClassifier(oob_score=best_params['oob_score'],
                                     n_estimators=best_params['n_estimators'],
                                    ␣
 ↪min_samples_split=best_params['min_samples_split'],
                                    ␣
 ↪min_samples_leaf=best_params['min_samples_leaf'],
                                     max_features=best_params['max_features'],
                                     max_depth=best_params['max_depth'],
                                     class_weight=best_params['class_weight']
                                    )
    # Show cross validated accuracy scores
```

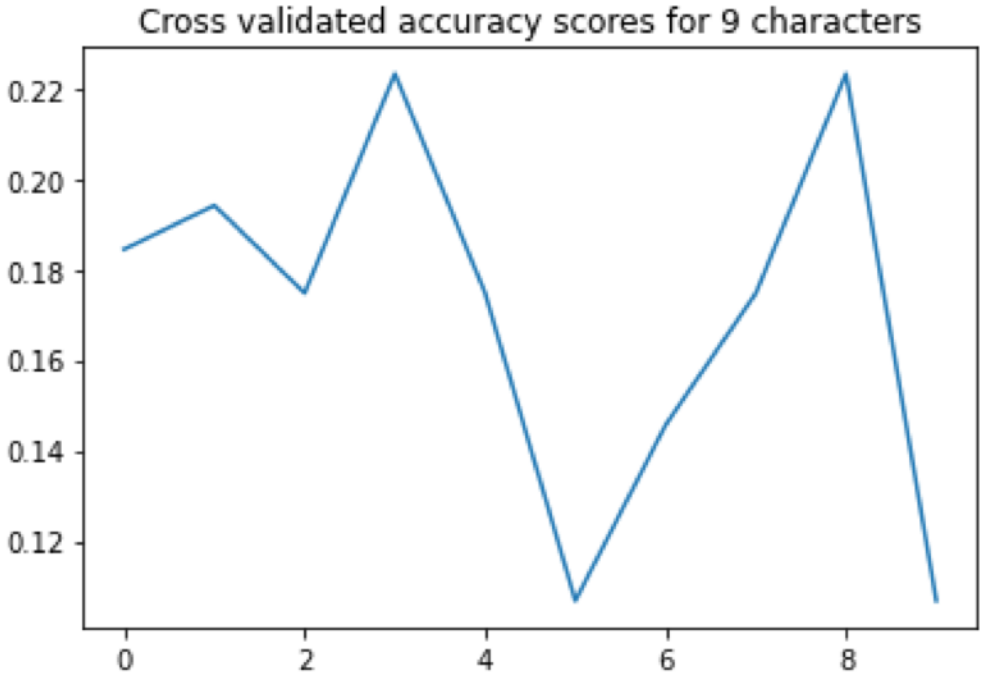
```
    get_cv_plot(best_rf, X_train, y_train, num_chars)

    #
    best_model_stats(best_rf,X_train, X_test, y_train, y_test)
```



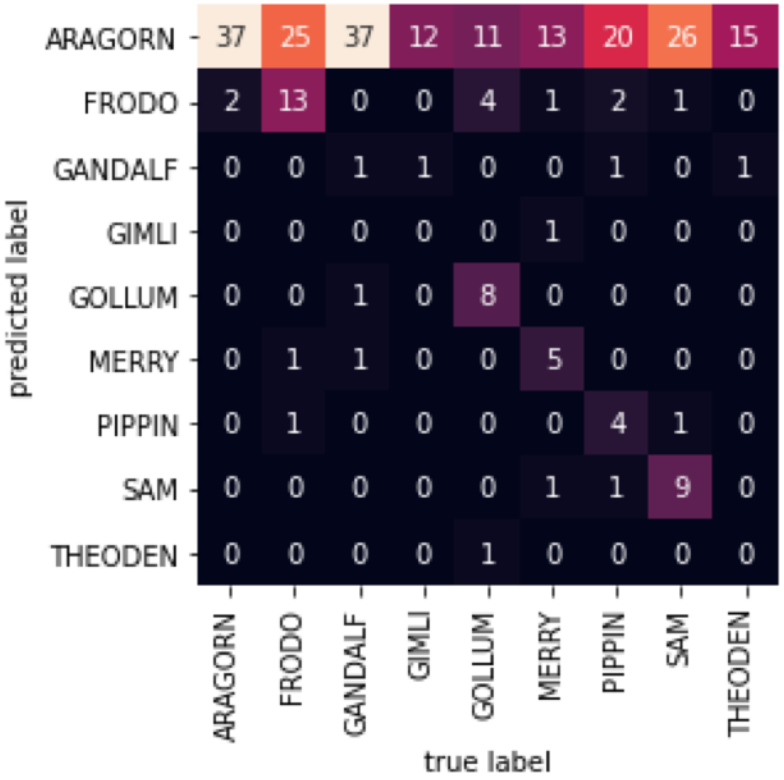

```
['ARAGORN', 'FARAMIR', 'FRODO', 'GANDALF', 'GIMLI', 'GOLLUM', 'MERRY', 'PIPPIN',
'SAM', 'THEODEN']
```



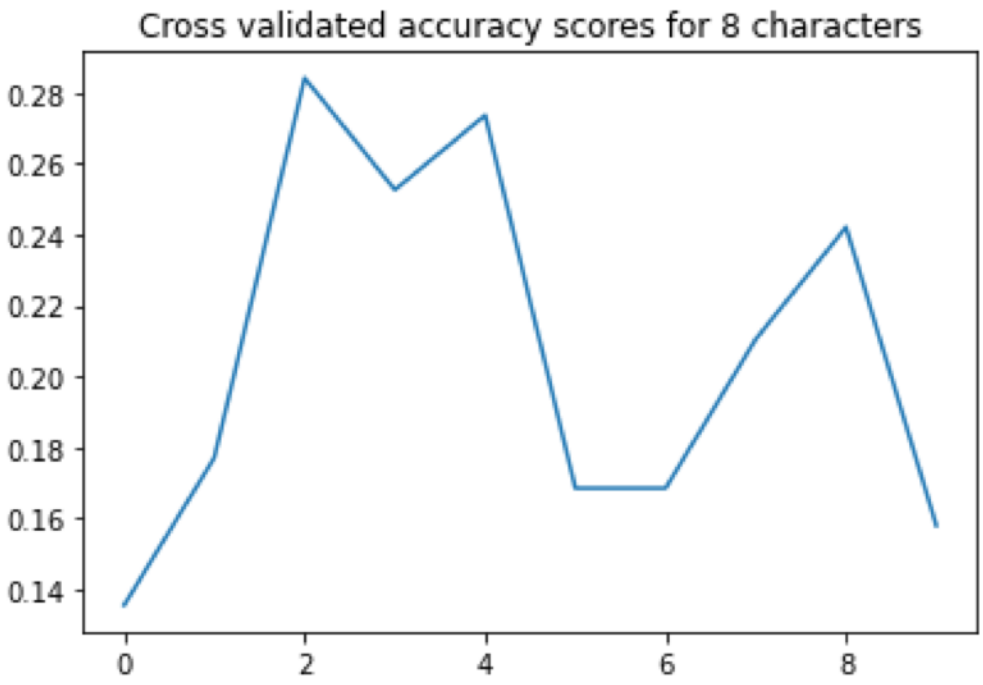

```
Accuracy score: 0.16728624535315986
Execution time: 7.914515972137451 seconds
```



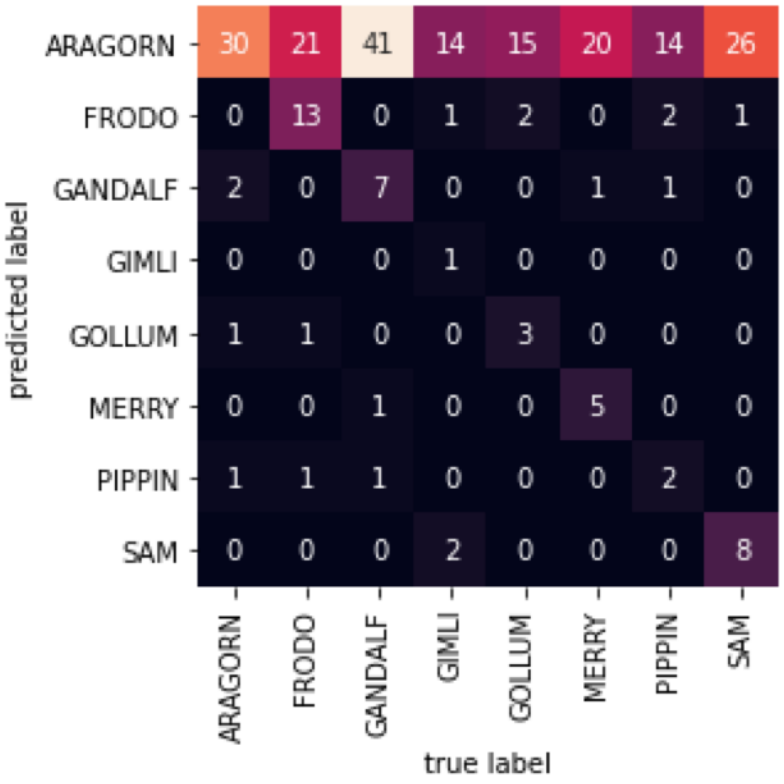
```
['ARAGORN', 'FRODO', 'GANDALF', 'GIMLI', 'GOLLUM', 'MERRY', 'PIPPIN', 'SAM',
'THEODEN']
```



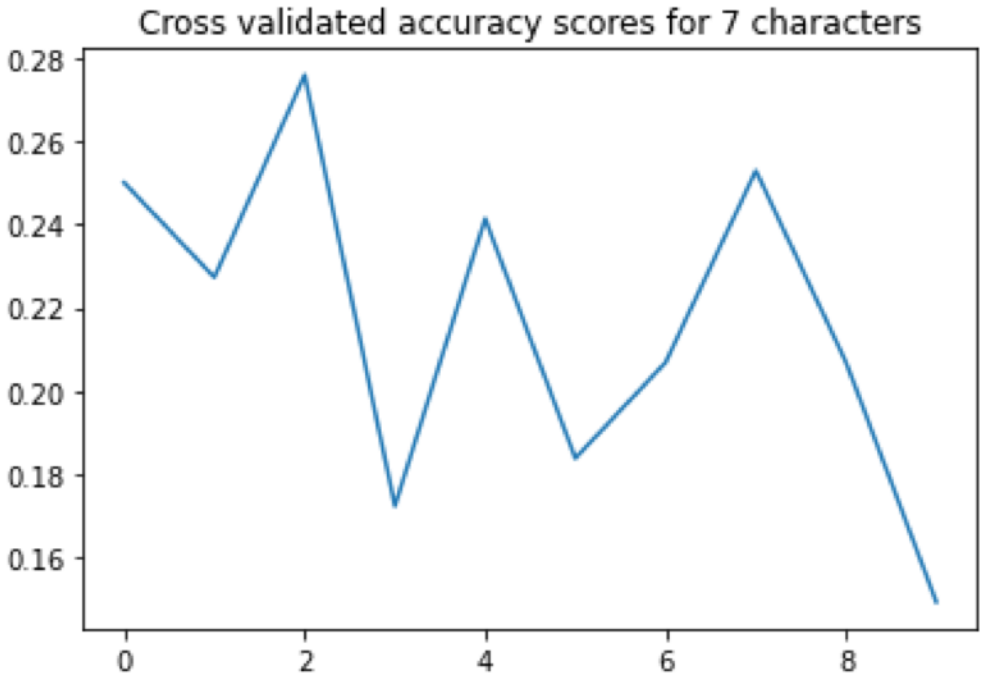

```
Accuracy score: 0.2131782945736434
Execution time: 7.082962274551392 seconds
```



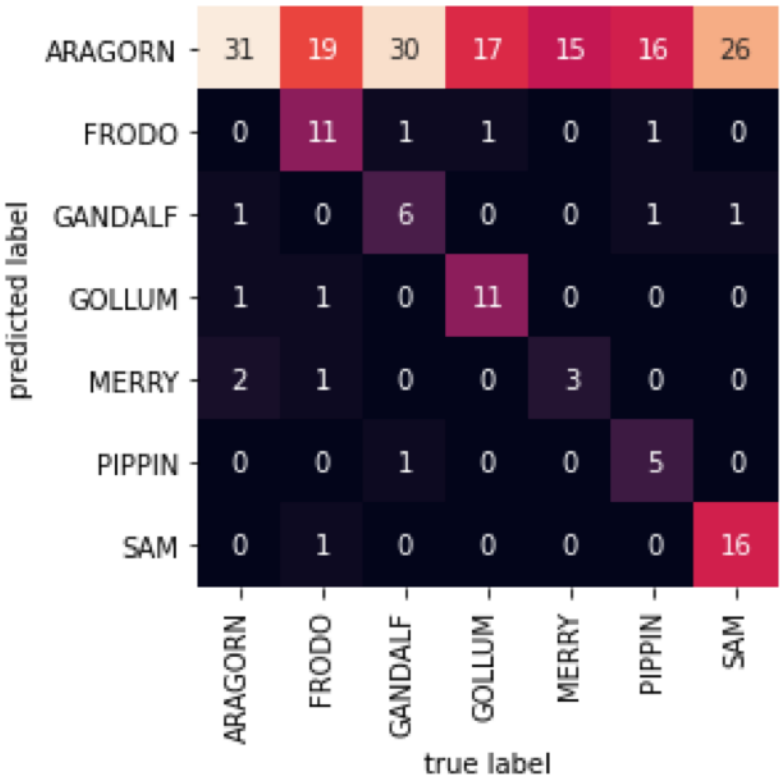

```
['ARAGORN', 'FRODO', 'GANDALF', 'GIMLI', 'GOLLUM', 'MERRY', 'PIPPIN', 'SAM']
```



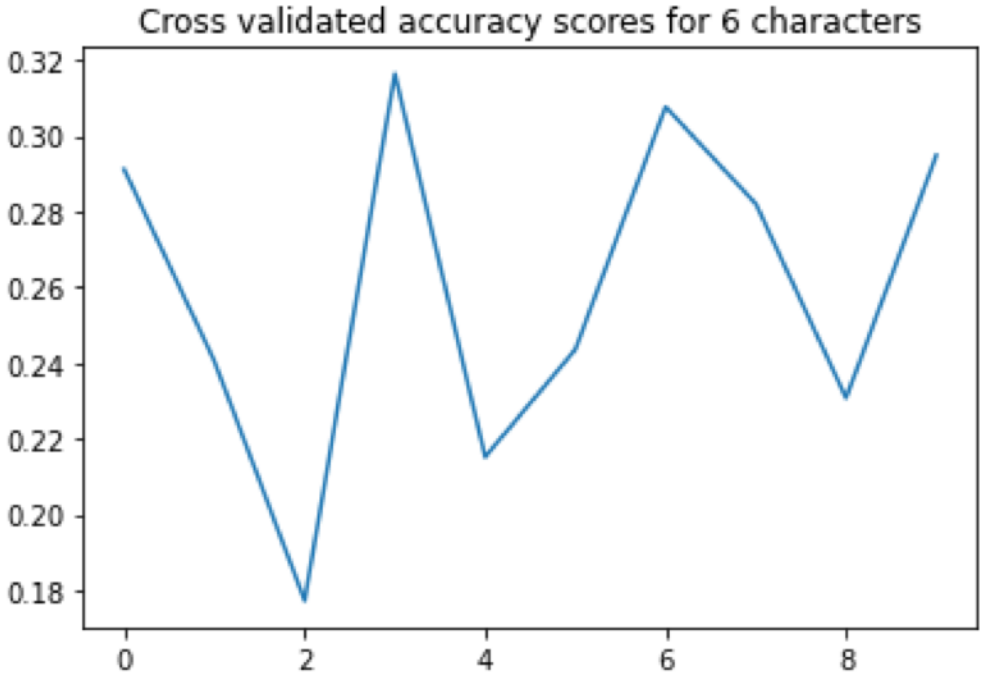
```
Accuracy score: 0.2184873949579832
Execution time: 6.125537872314453 seconds
```



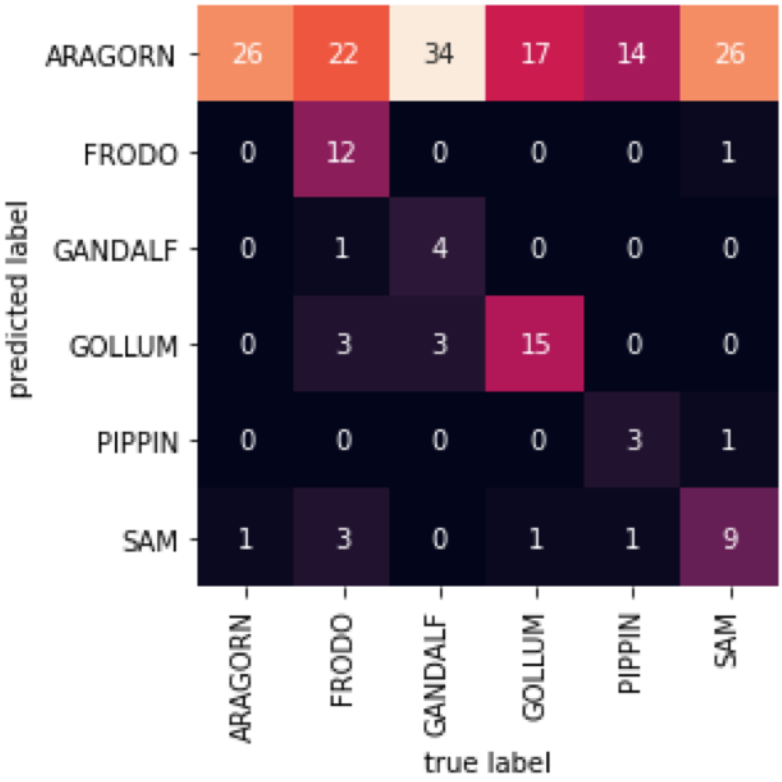

```
['ARAGORN', 'FRODO', 'GANDALF', 'GOLLUM', 'MERRY', 'PIPPIN', 'SAM']
```



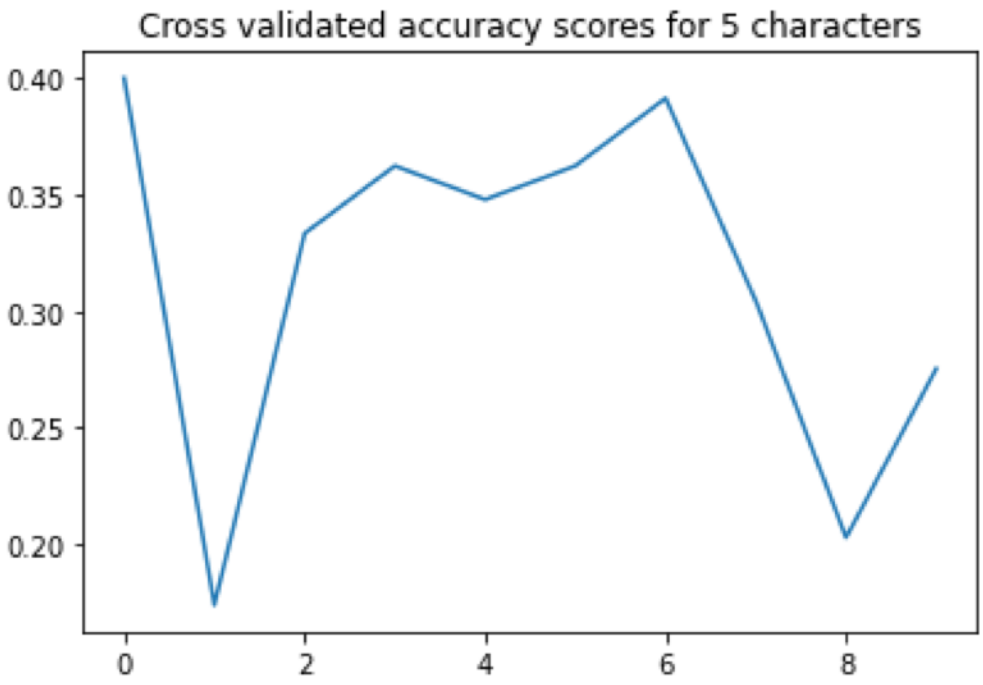

```
Accuracy score: 0.3105022831050228
Execution time: 5.214872121810913 seconds
```



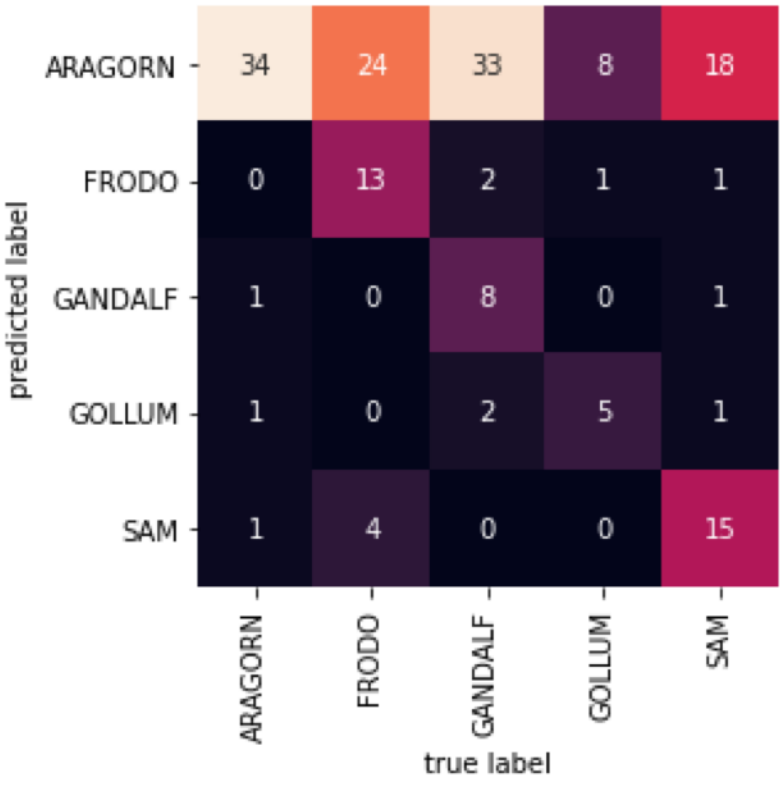
```
['ARAGORN', 'FRODO', 'GANDALF', 'GOLLUM', 'PIPPIN', 'SAM']
```



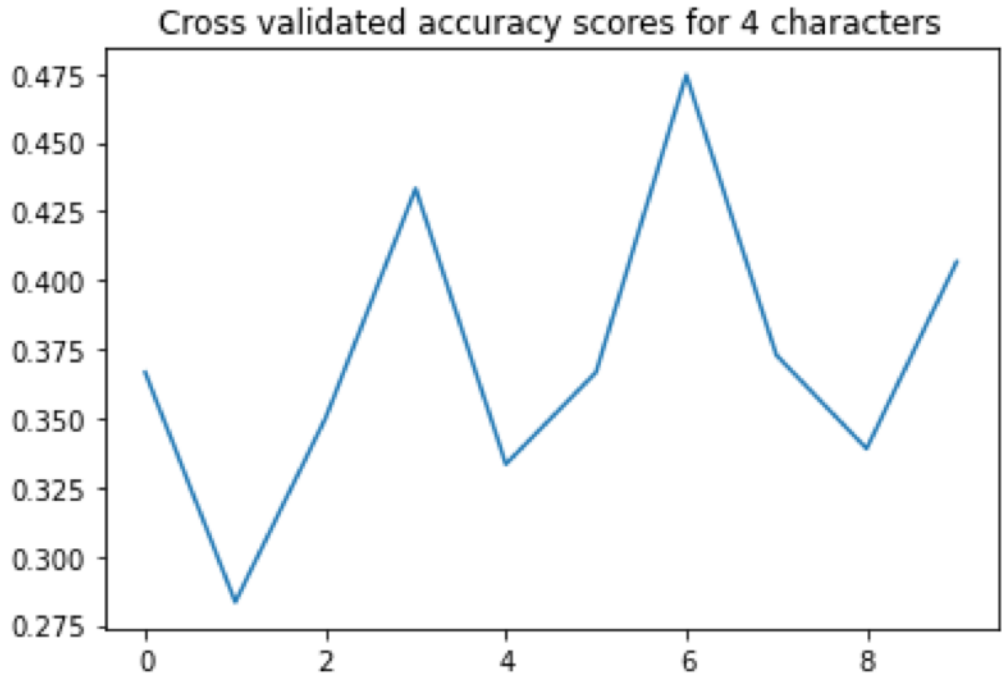

```
Accuracy score: 0.2893401015228426
Execution time: 4.321263313293457 seconds
```



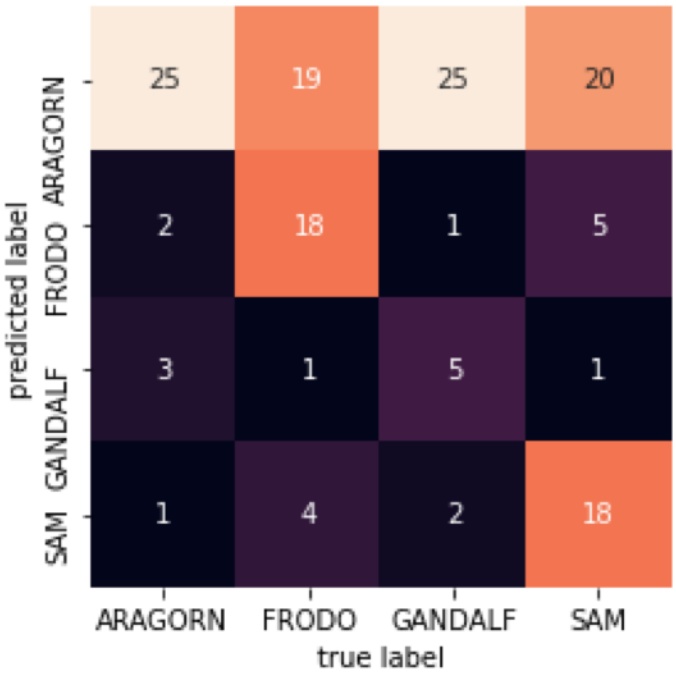

```
['ARAGORN', 'FRODO', 'GANDALF', 'GOLLUM', 'SAM']
```



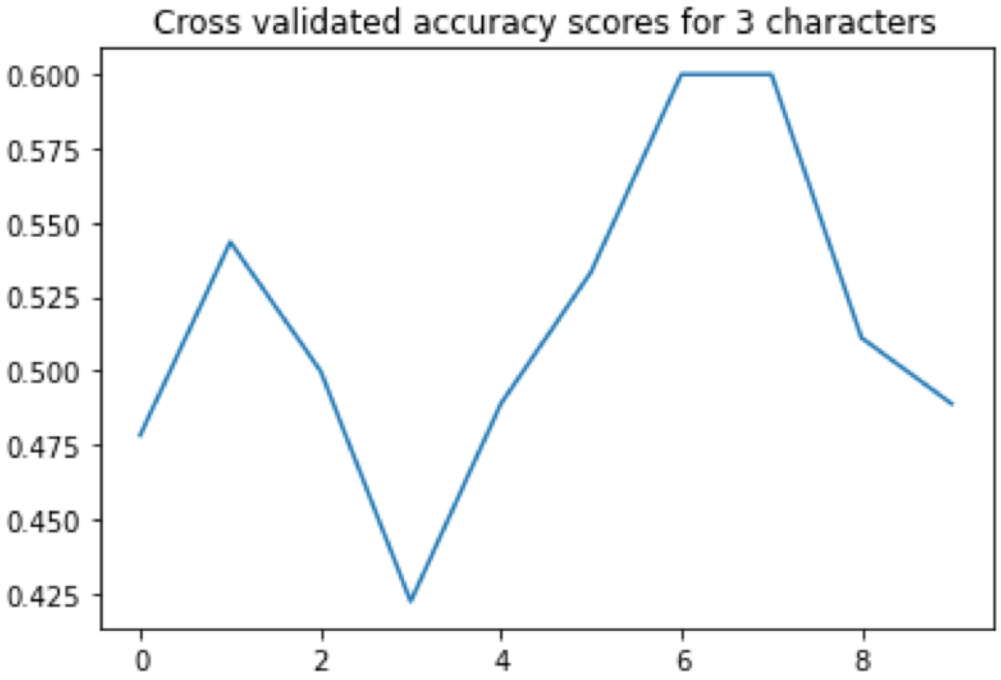
```
Accuracy score: 0.3352601156069364
Execution time: 3.5695831775665283 seconds
```



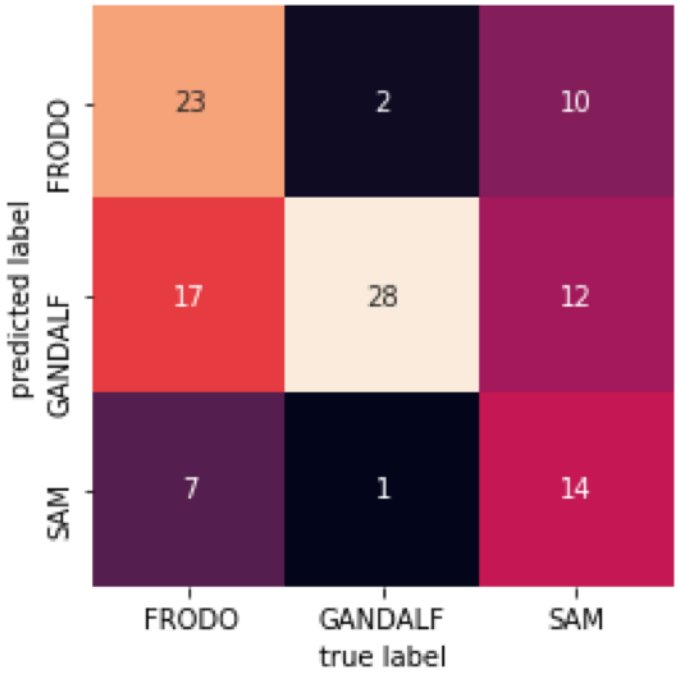

```
['ARAGORN', 'FRODO', 'GANDALF', 'SAM']
```



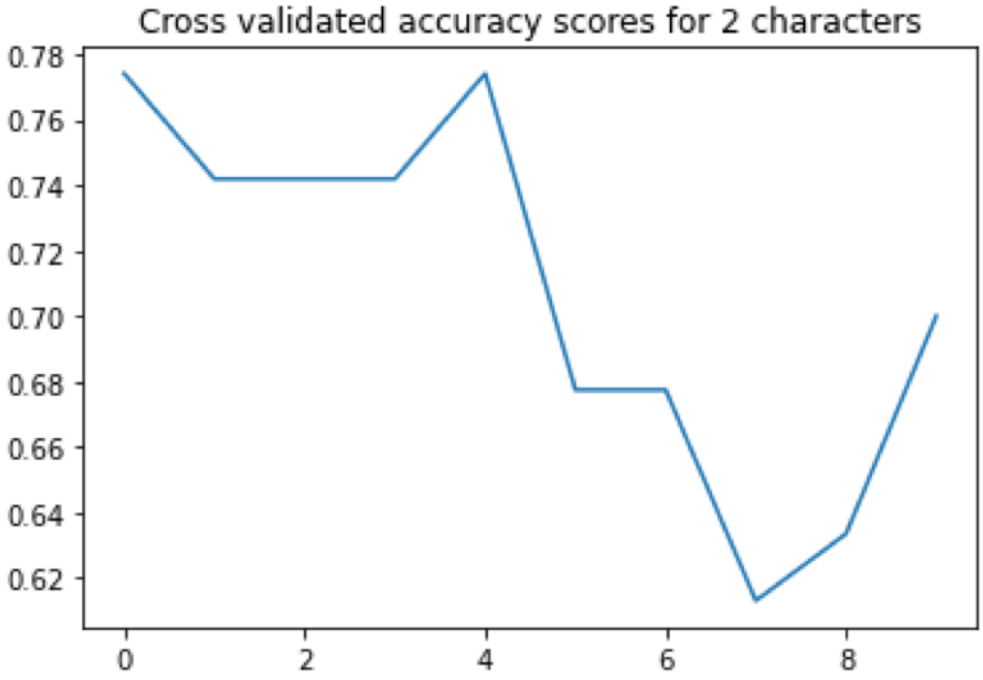

```
Accuracy score: 0.3933333333333333
Execution time: 2.900568723678589 seconds
```



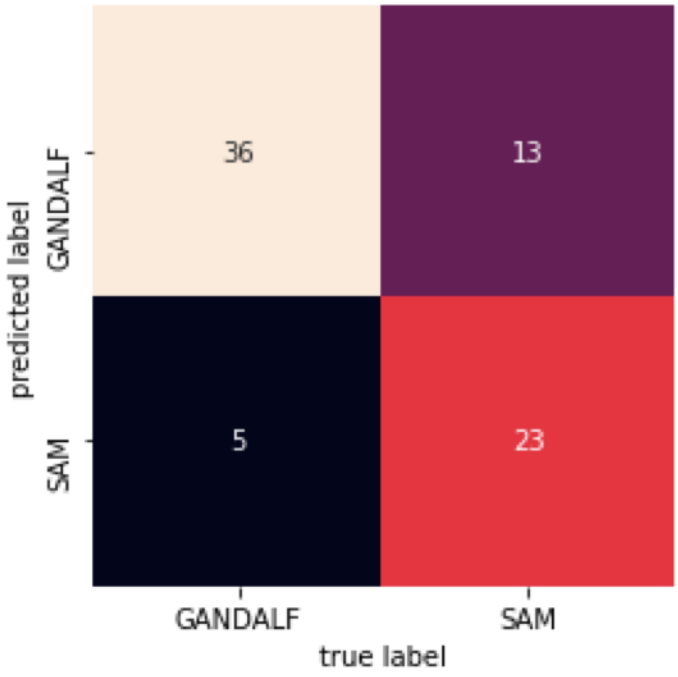
```
['FRODO', 'GANDALF', 'SAM']
```




```
Accuracy score: 0.5
Execution time: 5.684235095977783 seconds
```



```
['GANDALF', 'SAM']
```




```
Accuracy score: 0.7662337662337663
Execution time: 0.31673216819763184 seconds
```

[ ]:

[ ]: