17.Random Forest:

Now we will utilize random forest classification instead of logistic regression. To do this we will import xgboost utilizing the built in method in xbgoost to do our random forest modeling. We will conduct the same sequence that we did for Logistic Regression besides finding the best z-score as we already previously did that and found it to be 1.7 and 1.4 for english and universal respectively. Like previously we get the z-score, remove the outliers, use random forest on the cleaned up data set, then get our confusion matrices. It should look like this for the imports:

```
import pickle
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score
from sklearn import model_selection, preprocessing
from xgboost import XGBClassifier
```

We use the same methods as before for remove outliers and getting the z-score so they will not be gone over again refer to the previous logistic regression overview to refresh your knowledge on how to implemented those functions. The main difference is obviously how random forest is implemented and how it is used for the confusion matrices which we will now cover.

The random forest algorithm works by first selecting random samples from a given dataset then the algorithm constructs a decision tree for each training datapoint. A type of collective voting is done by averaging these decision trees out (mathematically). Finally the final prediction result is the most voted on decision tree. We implement this in python by way of XBBClassifier. We then create a dictionary which holds the attributes max_depth, n_estimators, and learning rate which corresponds to the max depth of the tree we go down which we assign to “2” the estimators which is the max amount of decision trees we make until final decision averages take place, then the learning rate which corresponds to the speed at which the model learns. This is from a scale of 0 to 1 which the closer to 0 the number is the more careful your model is when in the training phase. We then use GridSearchCV which helps us find the best parameters for our model. We pass into GridSearchCV our created dictionary then use pickle to open the model name and then we print the result like this:

```
def random_forest(input_df, print_result=False):
    x = input_df.drop(["spammer", "label"], axis=1).values
    if "id" in input_df.columns:
        x = input_df.drop(["id", "spammer", "label"], axis=1).values
    y = input_df["label"].values
    x_train, x_test, y_train, y_test = model_selection.train_test_split(x, y, test_size=0.30, random_state=100)

    x_train_scaled = preprocessing.scale(x_train)

    model = XGBClassifier(use_label_encoder=False)
    optimization_dict = {
        'max_depth': [2],
        'n_estimators': [200],
        'learning_rate': [0.1],
    }
    model = model_selection.GridSearchCV(model, optimization_dict,
                                         scoring='accuracy', verbose=1)
    model.fit(x_train_scaled, y_train, eval_metric="error")

    pickle.dump(model, open(f'rand_forest_eng.dat', 'wb'))

    if print_result:
        print(model.best_score_)
        print(model.best_params_)
```

For confusion matrix nearly the same is done akin to what we did for the logistic regression except now we do it for random forest. We still do the 4 types of normalization for the confusion matrices and all. Like last time we need to call our methods at the bottom to see our work. It should look like this (GitHub code base for random forest):

```
def confusion_matrix(lang, input_df, show=False):
    x = input_df.drop(["spammer", "label"], axis=1).values
    if "id" in input_df.columns:
        x = input_df.drop(["id", "spammer", "label"], axis=1).values
    y = input_df["label"].values
    x_train, x_test, y_train, y_test = model_selection.train_test_split(x, y, test_size=0.30, random_state=100)

    x_train_scaled = preprocessing.scale(x_train)
    x_test_scaled = preprocessing.scale(x_test)

    model = XGBClassifier(use_label_encoder=False)
    optimization_dict = {
        'max_depth': [2],
        'n_estimators': [200],
        'learning_rate': [0.1],
    }
    model = model_selection.GridSearchCV(model, optimization_dict,
                                         scoring='accuracy', verbose=1)
    model.fit(x_train_scaled, y_train, eval_metric="error")

    title_options = [(f"Confusion Matrix, without normalization\n({lang})", None),
                     (f"Normalization: true\n({lang})", "true"),
                     (f"Normalization: pred\n({lang})", "pred"),
                     (f"Normalization: all\n({lang})", "all")]

    for title, normalize in title_options:
        disp = ConfusionMatrixDisplay.from_estimator(model, x_test_scaled, y_test, display_labels=["bot", "human"],
                                                     cmap=plt.cm.Blues, normalize=normalize)
        disp.ax_.set_title(title)
        # plt.savefig(fr'C:\Users\Rachel\Documents\Twitter Data\{normalize}_{lang}_test')

    if show:
        plt.show()


dff = orgs_to_bots(input_path_en)
confusion_matrix("eng", remove_outliers(1.7, dff), True)
```