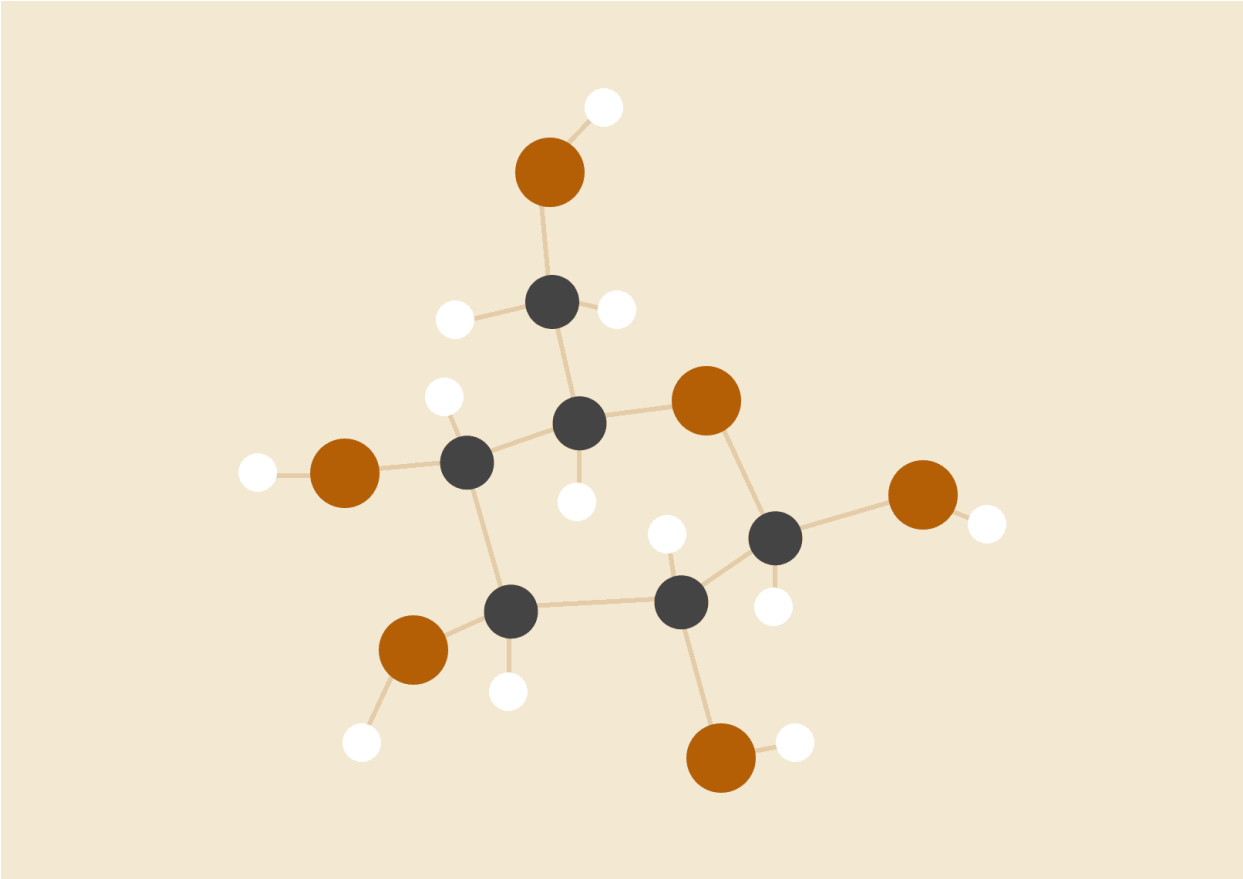
Forensics investigation from fingerprint microbes



Atakan AK 1030516753
Dilara KIZILKAYA 1030510143

Dr.Öğr. Üyesi Özkan Ufuk NALBANTOĞLU

INTRODUCTION

In this project, we aimed to identify which hand of an individual touched the computer keyboard using ML techniques.

The dataset consists of DNA samples that are taken from the computer keyboards.

STEPS

1. Preprocessing
2. Cross Validation in Feature Selection
3. Training/Testing
4. Evaluate Results
5. Hyperparameter Tuning

PREPROCESSING

At first we imported the csv with pandas's read_csv() method. We parsed the "unicode" parameter because we got a warning due to the different object types.

```
In [25]: data = pd.read_csv("otu.csv", dtype = "unicode")
         data.head()
```

Out[25]:

	Sample1	Sample2	Sample3	Sample4	Sample5	Sample6	Sample7	Sample8	Sample9	Sample10	...	Sample262	Sample263	Sample264	Sample265	San
0	left	left	left	left	left	left	left	left	left	left	...	right	right	right	right	
1	0.33364	0.62579	0	0	0.56233	0	0	0	0	0	...	0	6.11234	0	0	
2	0.33364	0.62579	0	0	0.56233	0	0	0	0	0	...	0	6.11234	0	0	
3	0.33364	0.62579	0	0	0.56233	0	0	0	0	0	...	0	6.10627	0	0	
4	0.33364	0.62579	0	0	0.56233	0	0	0	0	0	...	0	6.10627	0	0	

5 rows × 271 columns

As can be seen above, we used the head() method to see the first 5 rows. What we have realized was that the rows and columns were reversed. So we decided to transpose them.

But before doing so, we wanted to check whether we have empty samples. What we mean by the empty samples is the samples which contain entirely zero.

Both of the codes can be seen below.

In [26]:
```
data.drop(data.columns[(data == 0).all()], axis=1)
```

Out[26]:

	Sample1	Sample2	Sample3	Sample4	Sample5	Sample6	Sample7	Sample8	Sample9	Sample10	...	Sample262	Sample263	Sample264	Sample265
0	left	left	left	left	left	left	left	left	left	left	...	right	right	right	right
1	0.33364	0.62579	0	0	0.56233	0	0	0	0	0	...	0	6.11234	0	0
2	0.33364	0.62579	0	0	0.56233	0	0	0	0	0	...	0	6.11234	0	0
3	0.33364	0.62579	0	0	0.56233	0	0	0	0	0	...	0	6.10627	0	0
4	0.33364	0.62579	0	0	0.56233	0	0	0	0	0	...	0	6.10627	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3298	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
3299	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
3300	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
3301	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
3302	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0

3303 rows × 271 columns

In [27]:
```
dataset = data.T
dataset.head()
```

Out[27]:

	0	1	2	3	4	5	6	7	8	9	...	3293	3294	3295	3296	3297	3298	3299	3300	3301	3302
Sample1	left	0.33364	0.33364	0.33364	0.33364	0.33364	0.33364	0.33364	0.33364	0	...	0	0	0	0	0	0	0	0	0	0
Sample2	left	0.62579	0.62579	0.62579	0.62579	0.62579	0.62579	0.49776	0.49776	0.12802	...	0	0	0	0	0	0	0	0	0	0
Sample3	left	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
Sample4	left	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
Sample5	left	0.56233	0.56233	0.56233	0.56233	0.56233	0.56233	0.49446	0.49446	0.06786	...	0	0	0	0	0	0	0	0	0	0

5 rows × 3303 columns

As seen above, after transposing the dataset, now our samples are rows and features are the columns.

In [28]:
```
#turning lefts and rights into 0s and 1s
for i in range(len(dataset)):
    if dataset[0][i] == 'left':
        dataset[0][i] = 0
    elif dataset[0][i] == 'right':
        dataset[0][i] = 1
```

While we were working with the dataset, we first tried changing our target class manually. So we did this piece of code to make “left” values 0 and “right” values 1.

But after a while we decided to use label encoding to achieve what we need.

In [50]:
```
label_encoder = preprocessing.LabelEncoder() #creating the label encoder object
```

In [51]:
```
dataset[0] = label_encoder.fit_transform(dataset[0])
```

Here dataset[0] means our target class since it has the “left”, “right” data.

```
In [55]: X.columns[(X == 0).all()] #features which are entirely 0
Out[55]: Int64Index([  18,   43,   44,  132,  206,  207,  208,  209,  210,  211,
                     ...
                     3293, 3294, 3295, 3296, 3297, 3298, 3299, 3300, 3301, 3302],
                    dtype='int64', length=1915)
```

We had an idea of checking the features which are entirely 0s. We actually dropped them as well but it didn't affect our code as we wanted so we decided not to drop them at the end.

CROSS-VALIDATION IN FEATURE SELECTION

1. We create k-fold object with 5 splits
2. We split our dataset to 5 subsets and train subset by subset.
3. We evaluate the scores and record them.
4. End of the loop, We divide the sum of scores to num of scores to get a mean score.

Using KFold

```
accs =list()
losses = list()
for f,(train_idx,test_idx) in enumerate(kfold.split(X,y)):
    print("Fold ",f+1)
    X_train,X_test = X[train_idx],X[test_idx]
    y_train,y_test = y[train_idx],y[test_idx]

    print(f"Fold {f+1} training")
    model.fit(
        X_train,
        y_train,
        epochs=1,
        batch_size=32,
        validation_split=0.2
    )

    loss,accuracy = model.evaluate(X_test,y_test)

    print("Fold Acc: ",accuracy)
    accs.append(accuracy)
    losses.append(loss)


print("Mean Acc: ", sum(accs)/len(accs))
print("Mean Loss: ", sum(losses)/len(losses))
```

Using Cross Validation Score

```
models = [lr,dc,rfc,knn,svc,gnb,bnb]
#lr  LogisticRegression
#dc  DecisionTreeClassifier
#rfc RandomForestClassifier
#knn KNeighborsClassifier
#svc SVC
#gnb GaussianNB
#bnb BernoulliNB
for model in models:
    score = cross_val_score(model, X_train,y_train,cv=5).mean()
    loss = cross_val_score(model, X_train,y_train,cv=5).std()

    print(f"{model} Accuracy :", score)
    print(f"{model} Loss :", loss)
    print("\n")
```

TRAINING/TESTING

Firstly we import our classifier model with default parameters.We use eight different machine learning algorithms and one simple ann model.

1. Adaboost
2. XGBoost
3. Random Forest Classifier
4. Logistic Regression
5. KNeighbors Classifier
6. Decision Tree
7. Support Vector Machine
8. Naive Bayes (Gaussian and Bernoulli)
9. ANN (Artificial Neural Network)

ADABOOST

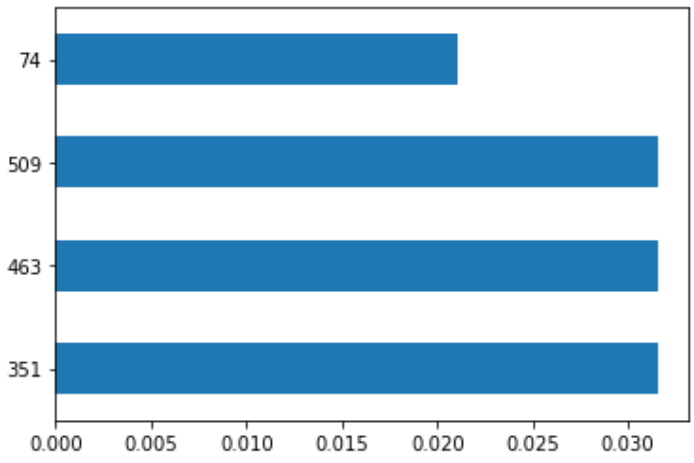
ADABOOST was one of the models that we tried in our early model selection process. It gave an accuracy score of 0.65.

As can be seen in the code below, we set the hyperparameters as 95, 0.8, 42. We tried the model with different hyperparameter values though these ones were the best ones.

```
In [60]: #ADABOOST
         ada_boost = AdaBoostClassifier(n_estimators=95, learning_rate=0.8, random_state=42)
         model = ada_boost.fit(X_train, y_train)
         y_pred = model.predict(X_test)

In [61]: from sklearn.metrics import accuracy_score

In [63]: accuracy_score(y_test, y_pred)

Out[63]: 0.6545454545454545
```

Also after we used adaboost, we really wanted to see the feature importances. So we saw that the feature 509 was the most important yet 463 and 351's importances were pretty close to it.

```
In [36]: (pd.Series(model.feature_importances_, index=X.columns)
            .nlargest(4)
            .plot(kind='barh'))

Out[36]: <AxesSubplot:>
```



XGBOOST

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.model_selection import KFold,cross_val_score,train_test_split
from sklearn.utils import shuffle
import xgboost as xgb
from sklearn.metrics import accuracy_score
```

The libraries we used while using xgboost can be seen above.

```
kfold=KFold(n_splits=5)
data = pd.read_csv("otu.csv", dtype = "unicode")
dataset = data.T

dataset = dataset.sample(frac = 1).reset_index(drop=True)

le=LabelEncoder()
sc = StandardScaler()


X=dataset.iloc[:, 1:].astype("float64").to_numpy()
```

```
kfold=KFold(n_splits=5)
data = pd.read_csv("otu.csv", dtype = "unicode")
dataset = data.T

dataset = dataset.sample(frac = 1).reset_index(drop=True)

le=LabelEncoder()
sc = StandardScaler()


X=dataset.iloc[:, 1:].astype("float64").to_numpy()

#Encoding the labels
y = le.fit_transform(dataset.iloc[:, 0])

#Fitting the values between 0 and 1
X=sc.fit_transform(X)
```

The steps above were for the data preparation. Here we scaled our data with using StandardScaler() method as well.

```
data_dmatrix = xgb.DMatrix(data=X, label=y)


xg_reg = xgb.XGBClassifier(objective ='reg:squarederror', colsample_bytree = 0.3, learning_rate = 0.1,
            max_depth = 5, alpha = 10, n_estimators = 10)

accs =list()
losses = list()
for f,(train_idx,test_idx) in enumerate(kfold.split(X,y)):
   print("Fold ",f+1)
   X_train,X_test = X[train_idx],X[test_idx]
   y_train,y_test = y[train_idx],y[test_idx]

   print(f"Fold {f+1} training")
   xg_reg.fit(X_train,y_train)
   y_pred=xg_reg.predict(X_test)
   accuracy = accuracy_score(y_test, y_pred)

   print("Fold Acc: ",accuracy)
   accs.append(accuracy)
```

First of all we turned our dataset into a DMatrix. This step is a must before using Xgboost. Then we created the xgboost object with the given hyperparameters.

Then we manually did k-fold.

After calculating the accuracies,

```
print("XGBoost Mean Acc: ", sum(accs)/len(accs))


preds = xg_reg.predict(X_test).round()

score = accuracy_score(y_test, preds)
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test, preds)
```

```
XGBoost Mean Acc:  0.6012121212121213
```

	0	1
0	18	12
1	7	17

These were our results.

EVALUATE RESULTS

When we calculated the results, we wanted to see them as a whole so we printed them as accuracy and loss.

```
LogisticRegression() Accuracy : 0.6527027027027027
LogisticRegression() Loss : 0.13840765586384393


DecisionTreeClassifier() Accuracy : 0.6238738738738739
DecisionTreeClassifier() Loss : 0.03361150602011968


RandomForestClassifier() Accuracy : 0.6406906906906907
RandomForestClassifier() Loss : 0.06810430790569617
```

```
KNeighborsClassifier() Accuracy : 0.6127627627627628
KNeighborsClassifier() Loss : 0.10232972331231076


SVC() Accuracy : 0.662912912912913
SVC() Loss : 0.07557789574095679


GaussianNB() Accuracy : 0.6243243243243243
GaussianNB() Loss : 0.07955492847273485


BernoulliNB() Accuracy : 0.6684684684684685
BernoulliNB() Loss : 0.04662059401265065
```

ANN model loss and accuracy:

```
Mean Acc:  0.8385185122489929
Mean Loss:  0.47933833599090575
2/2 [==============================] - 0s 3ms/step
              precision    recall  f1-score   support

           0       0.74      0.82      0.78        28
           1       0.78      0.69      0.73        26

    accuracy                           0.76        54
   macro avg       0.76      0.76      0.76        54
weighted avg       0.76      0.76      0.76        54
```

SENSITIVITY & SPECIFICITY SCORES

Here are the sensitivity and specificity scores we got for each model:

```
LogisticRegression() Sensivity:  40.62
LogisticRegression() Specificity:  40.6
```

```
DecisionTreeClassifier() Sensivity:  40.68
DecisionTreeClassifier() Specificity:  40.56
```

```
RandomForestClassifier() Sensivity:  40.62
RandomForestClassifier() Specificity:  40.42
```

```
KNeighborsClassifier() Sensivity:  40.52
KNeighborsClassifier() Specificity:  40.52
```

```
SVC() Sensivity:  40.64
SVC() Specificity:  40.5
```

```
GaussianNB() Sensivity:  40.64
GaussianNB() Specificity:  40.48
```

```
XGBoost Sensivity:  24.5
XGBoost Specificity:  24.366666666666667
```

```
BernoulliNB() Sensivity:  40.66
BernoulliNB() Specificity:  40.48
```

```
ANN Sensivity:  26.75
ANN Specificity:  26.785714285714285
```

HYPERPARAMETER TUNING

According to results, we tried to improve our accuracy values by changing the parameters of the algorithms. For the AdaBoost algorithm we changed the "n_estimator" and "learning_rate" values and chose the best values ("n_estimator:95 , learning_rate:0.8"). Then we changed the ann parameters like dense layer neurons number, num of epochs and batch size. We started with 2 layer ,(64,128) neurons,2 epoch , 16 batch size then we added layer with 64 layer , increased the batch size to 32, decreased epoch to 1. End of the result we improved our accuracy to 67 to 80.