CSE 435
Biomedical Signal Analysis
Final Project

Murat Sahilli

Feb 02, 2022

Contents

Preface

PROBLEM

Identifying hand gestures using a touch sensor.

DESCRIPTION

The capacitive touch sensor can measure not only touch but also proximity. By using this feature of the sensor, we can measure the changes caused by different hand movements on the human wrist and find out which hand movement is made. In order to do this, the touch surface of the sensor was extended to cover the entire wrist with the help of copper wire and plate (see Figure 1). Measurement of different hand movements was taken with this wristband. Then the model was created by being trained by machine learning models.

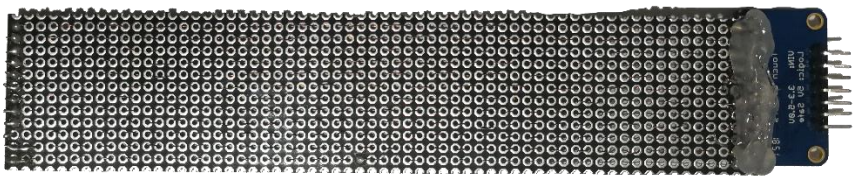


Figure 1 Capacitive Touch Sensor as wristband

PART ONE Data Explanation:

Capacitive touch sensor has twelve channels. Through these channels, it was measured how close or far each cable connected to this channel was from the wrist. Afterwards, the measurements were transferred to the computer via Arduino. A dataset was created using these measurements. Feature extraction is applied to incoming measurements and recorded. The dataset has six classes, and every class means a hand sign.

Class 0: Open hand.
Class 1: Fist.
Class 2: Showing number one with index finger.
Class 3: Showing number two with index and middle finger.
Class 4: Rock sign.
Class 5: Spider-man.

Classes were created as a result of entering which hand gesture was made simultaneously from the keyboard while making a hand gesture. Python pynput library was used to read data from keyboard with Arduino at the same time (see Figure 2). The dataset was obtained by performing each hand movement for an average of 20 seconds.

```
status= "0"

def on_press(key):
    global status

    status= str(key)
    print(status)

app = QtGui.QApplication([])
qt_keys = (
    (getattr(QtCore.Qt, attr), attr[4:])
    for attr in dir(QtCore.Qt)
    if attr.startswith("Key_")
)
keys_mapping = defaultdict(lambda: "unknown", qt_keys)
class KeyPressWindow(pg.GraphicsLayoutWidget):
    sigKeyPress = QtCore.pyqtSignal(object)

    def keyPressEvent(self, ev):
        self.scene().keyPressEvent(ev)
        self.sigKeyPress.emit(ev)

def keymap(event):
    global status,keys_mapping
    status = keys_mapping[event.key()]
    print("status= " , status)
```

Figure 2 Pynput keylistener

PART TWO Feature Extraction

Feature extraction was performed before recording measurements taken with the wristband. First, an array of 100 rows and 11 columns was created to append the incoming data.

```
list_data = np.zeros((100, 11),dtype="int32")
```

As the data came in, it was added as the last line and the first line was deleted, thus creating a window that progressed one by one with a width of 100 lines.

```
list_data = np.append(list_data, a, axis=0)
list_data = np.delete(list_data, (0), axis=0)
```

For each incoming row of data, the mean, standard deviation, min and max values of the entire list were calculated. In addition, the kurtosis value was calculated for all columns of the list. Then these calculations were written to an opened text file with the label and time.

```
fmean=np.mean(list_data)
fstd=np.std(list_data)
fmax=np.max(list_data)
fmin=np.min(list_data)
fkurtosis=kurtosis(list_data)
with open("data_02.txt","a") as file:
    for i in range((len(fkurtosis))):
        file.write(str(round(fkurtosis[i],2))+' ')
    file.write(str(round(fmean,2))+" "+str(round(fstd,2))+" "+str(round(fmax,2))+
               " "+str(round(fmin,2))+" "+status+" "+str(now)+"\n")
```

PART THREE Data Preprocessing and Visualization

The dataset contains eighteen columns. The columns from zero to eleven represent the kurtosis value of each column in list, From the 11th column to the 15th column, the mean standard deviation, min and max values are included, respectively. the 16th column represents the class equivalent of these measurements, and the 17th and 18th columns represent the time.

The correlation coefficient of the variables in the data set with each other. It can be seen in Figure 3 that the relationships of kurtosis values and other values differ from each other.

```
print("The correlation coefficient of the variables in the data set with each other")
corr = np.abs(data.corr(method='pearson'))
plt.figure(figsize = (8,6))
sns.heatmap(corr, annot = True)
```

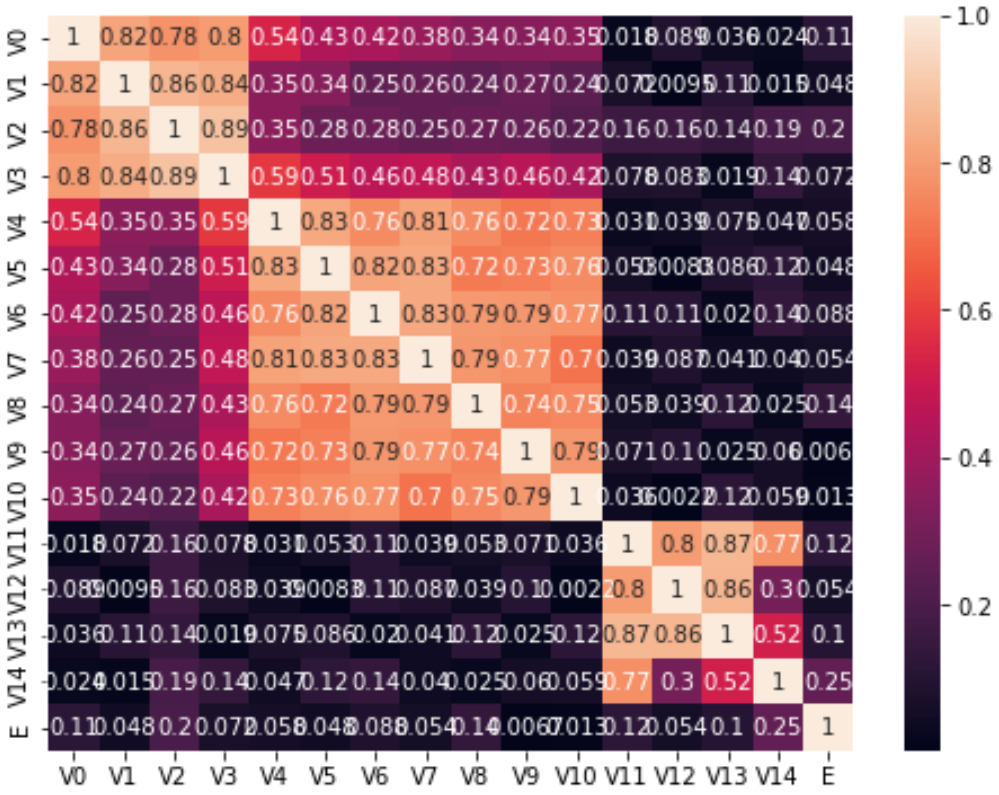


Figure 3 Correlation Coefficient

In Figure 3, the value ranges and value weights of the input and output columns can be seen.

```
data.hist(bins=18,figsize=(16,9),grid=False);
```

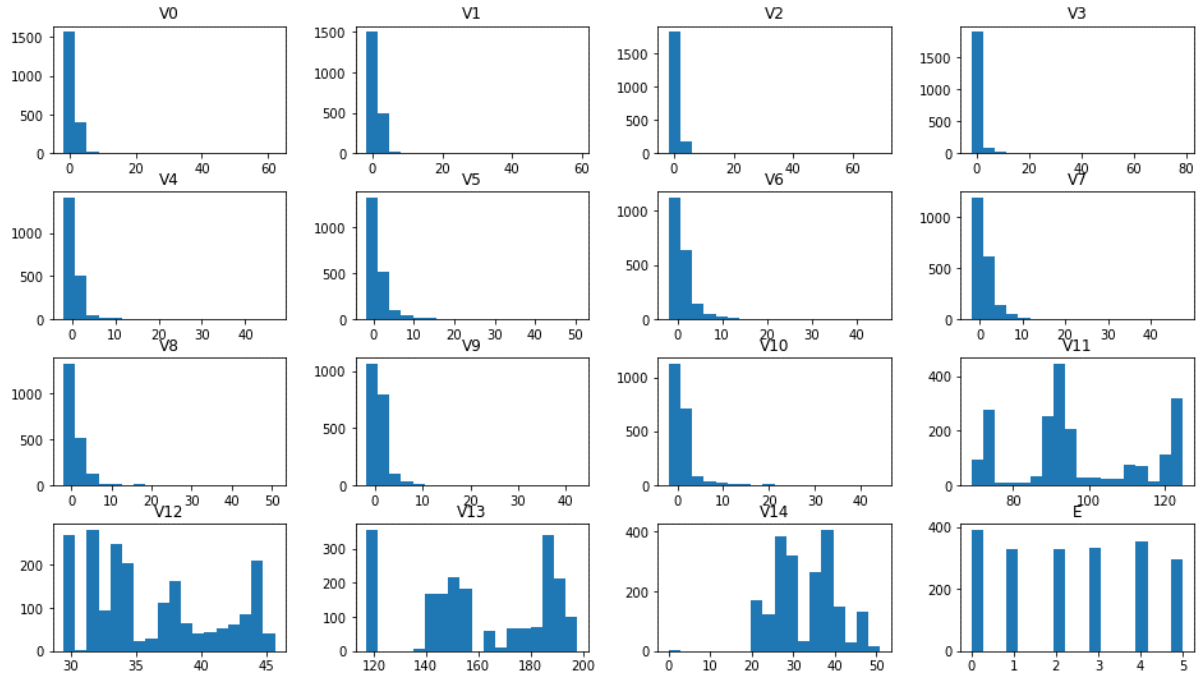
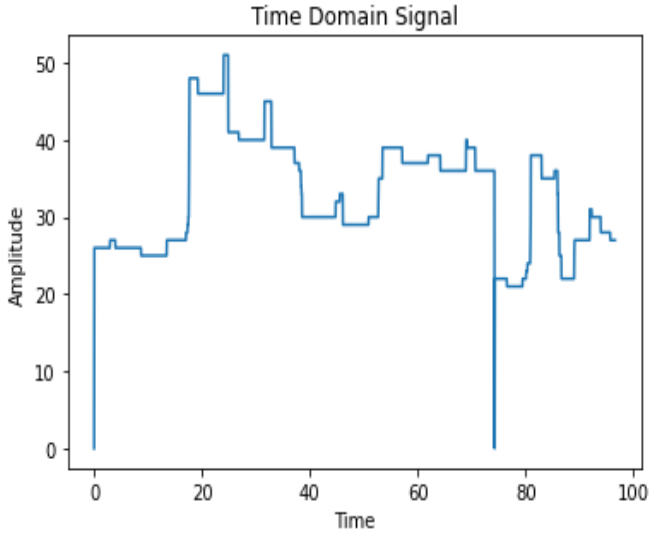


Figure 4 Data Histogram

Time-amplitude plot created using only one channel. By looking at sharp changes in amplitude, it can be seen when hand movements change.

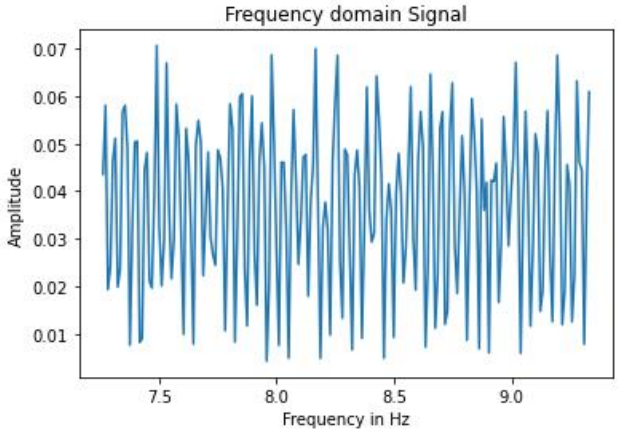



```
#create time domain plot
sampling_rate = 21
N= len(X)
zaman = np.linspace(0, N/sampling_rate, N)

plt.plot (zaman, a)
plt.title ('Time Domain Signal')
plt.xlabel ('Time')
plt.ylabel ('Amplitude')
plt.show ()
```

Figure 5 Time-Domain

Frequency analysis of data in the tenth to two hundredth row range of a single channel. Frequency amplitude plot of 120 values between line 750 and line 870. There is a hand gesture change in the dataset starting from the line 788, but it is not very clear from this graph.




```
frequency = np.linspace(0.0, 21/2, int(N/2))
freq_data = fft(a)
freq_abs = 2/N * np.abs(freq_data[0:int(N/2)])

plt.plot(frequency[700:900], freq_abs[700:900])
plt.title('Frequency domain Signal')
plt.xlabel('Frequency in Hz')
plt.ylabel('Amplitude')
plt.show()
```

Figure 6 Frequency-Domain

In the for loop, the STFT of each channel is calculated separately and added on top of each other to create a STFT graph of the data. Color changes seen as columns indicate changes in hand Window size 2 and hop length set to 1 when plotting the graph.

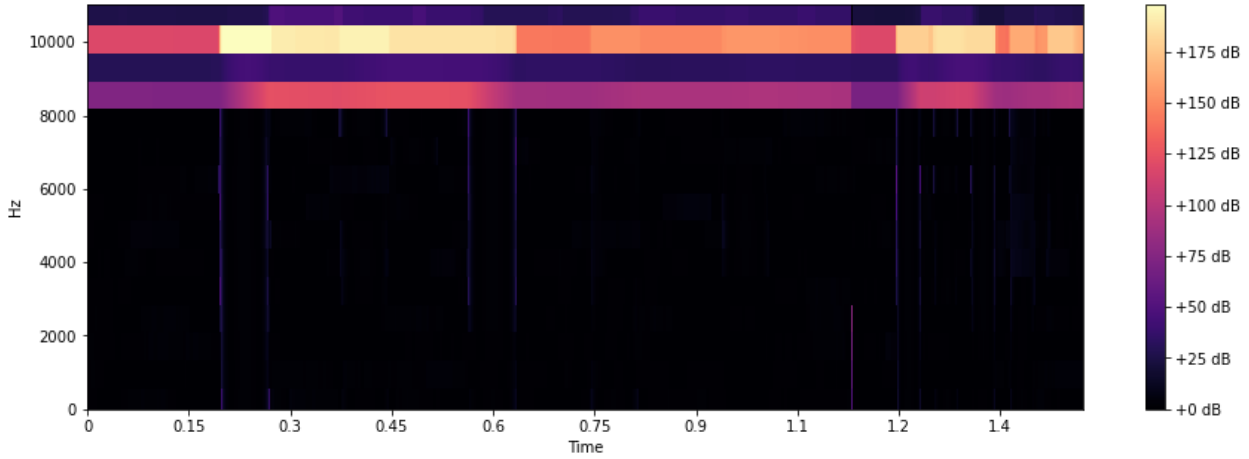
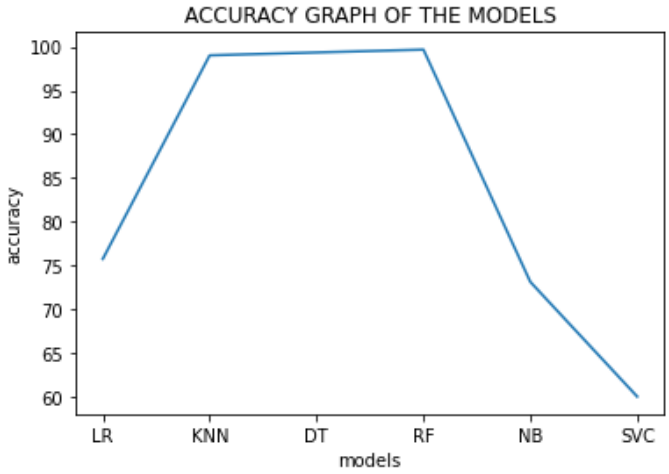


Figure 7 STFT Graph

PART FOUR Machine Learning and Comparison

Six different algorithms were used while training in machine learning. These are Logistic Regression, K-Neighbors Classifier, Decision Tree Classifier, Random Forest Classifier, Gaussian NB and SVC respectively. The dataset was split 0.85 – 0.15 percent as training and testing.



After the model was trained and tested, the accuracy graph of the models was drawn (see Figure 7). According to the data set we created the two models that learned best were Decision Tree Classifier with 99.34% and Random Forest Classifier with 99.67%.

Figure 8 Accuracy Graph - 1

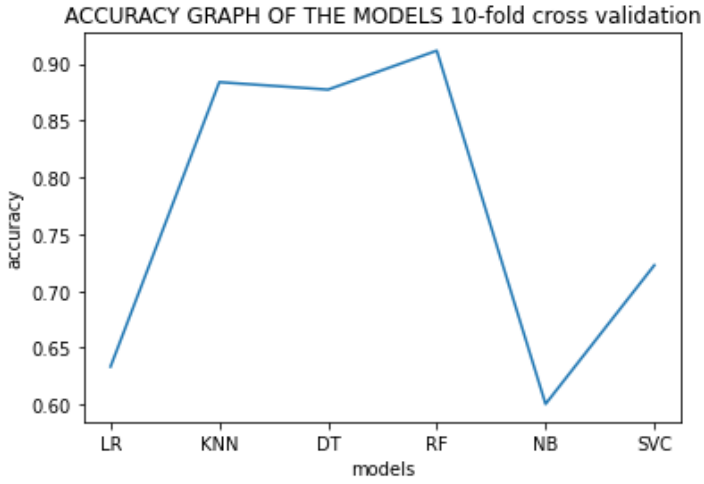
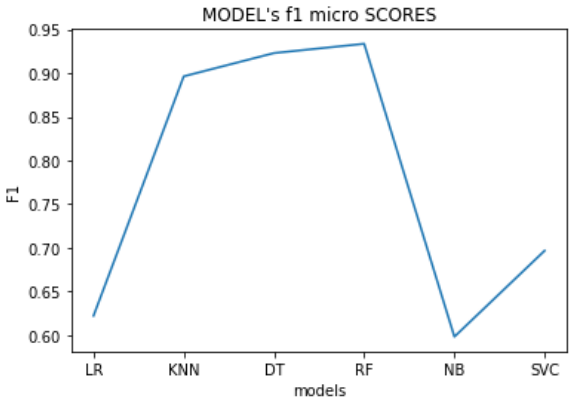


Figure 8 was drawn by applying K-fold to the input data and averaging the accuracy of the models for 10 folds.

As a result of this implementation, the accuracy of only Logistic Regression increased by 3%. On the other hand accuracy of Decision Tree decreased by 1%

The model with the highest accuracy Random Forest with 99.5%, followed by K-Neighbors Classifier with 98.7%.

Figure 9 Accuracy Graph - 2 K-Fold



F1 micro values of the models were plotted. F1 Score value shows us the harmonic average of Precision and Recall values. A micro-average will aggregate the contributions of all classes to compute the average metric.

Figure 10 Accuracy Graph - 3

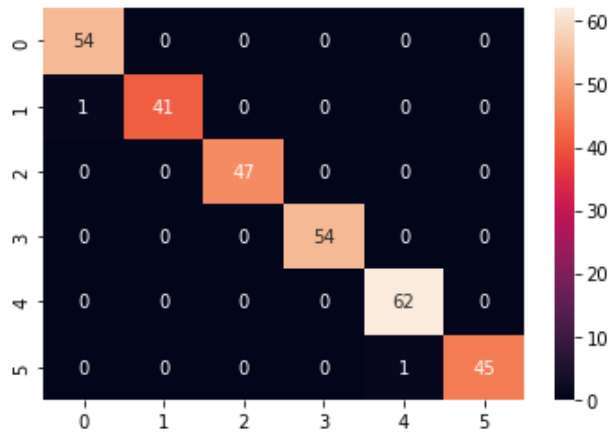


Figure 12 Confusion Matrix - RF

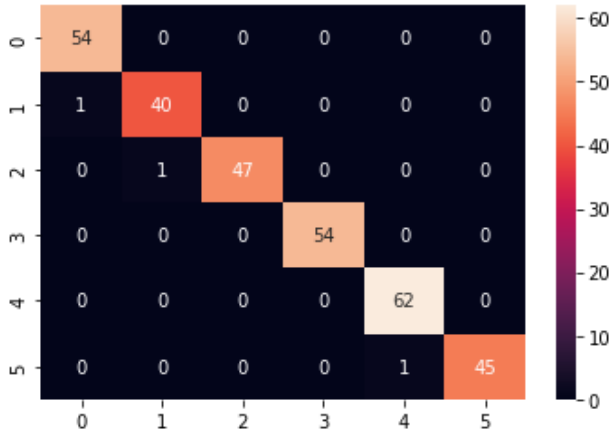


Figure 11 Confusion Matrix - KNN

```
rf = RandomForestClassifier()
rf.fit(X_train,Y_train)
y_pred = rf.predict(X_test)

filename = 'finalized_model_rf.sav'
pickle.dump(rf, open(filename, 'wb'))

conf= confusion_matrix(y_pred,Y_test)
label=["0","1","2","3","4","5"]
plt.figure()
sns.heatmap(conf,annot=True,xticklabels=label,yticklabels=label,fmt="g")
```

Figure 10 and Figure 11 showing the difference between the estimation made by the model created with the Random Forest Classifier algorithm and K-Neighbors algorithm and the actual values. From these graphs, it can be said that the model learned very well.

All Precision, Recall and F1 scores comparison table

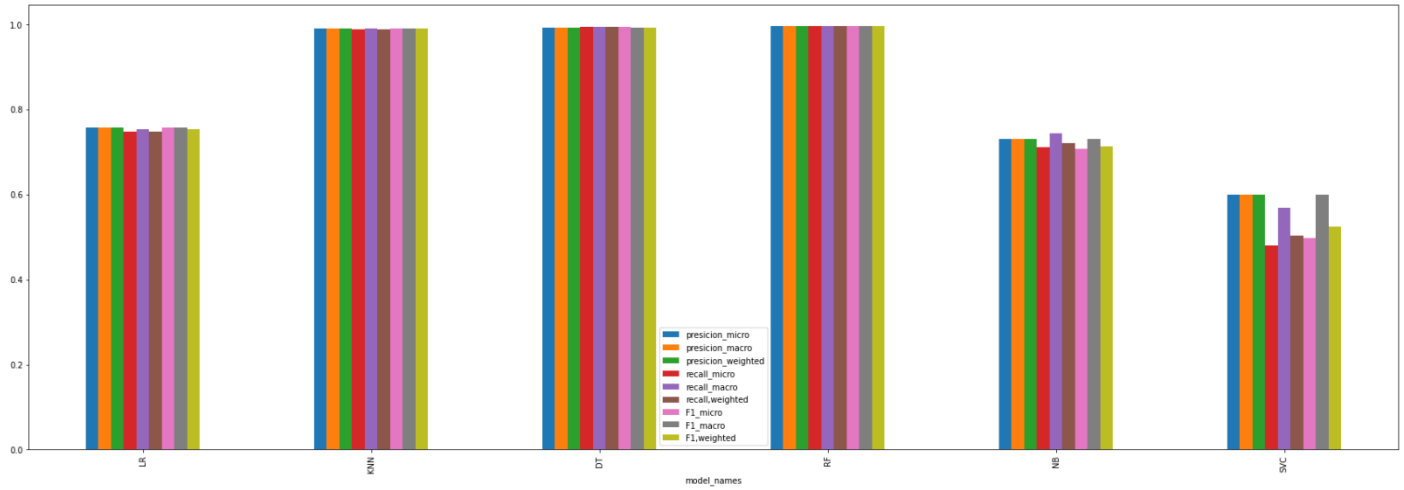


Figure 13 Comparison Table

```
namesdf=pd.DataFrame(names)
microsdf=pd.DataFrame(micros)
macrosdf=pd.DataFrame(macros)
weightsdf=pd.DataFrame(weights)
concat_df   = pd.concat([namesdf,microsdf,macrosdf,weightsdf], axis=1)
concat_df.columns=["model_names","presicion_micro","presicion_macro","presicion_weighted",
                   "recall_micro","recall_macro","recall,weighted",
                   "F1_micro","F1_macro","F1,weighted"]

# plot the dataframe
concat_df.plot(x="model_names", y=["presicion_micro","presicion_macro","presicion_weighted",
                   "recall_micro","recall_macro","recall,weighted",
                   "F1_micro","F1_macro","F1,weighted"], kind="bar", figsize=(30,10))
plt.legend(loc="lower center")
# print bar graph
plt.show()
```

PART FIVE Model Test and Results

A new unfit dataset is created to test the trained model. Prediction was made with the model using this dataset.

The models we obtained as a result of the training were loaded.

```
loaded_model_rf = pickle.load(open("finalized_model_rf.sav", 'rb'))
loaded_model_knn = pickle.load(open("finalized_model_knn.sav", 'rb'))
```

Then the data we recorded for testing was predicted.

```
pred_res_rf= loaded_model_rf.predict(X)
```

A confusion matrix was drawn by comparing the estimated values with the actual values.

```
conf_pr_rf= confusion_matrix(pred_res_rf,output)
label=["0","1","2","3","4","5"]
plt.figure()
sns.heatmap(conf_pr_rf,annot=True,xticklabels=label,yticklabels=label,fmt="g")
print(accuracy_score(pred_res_rf,output))
```

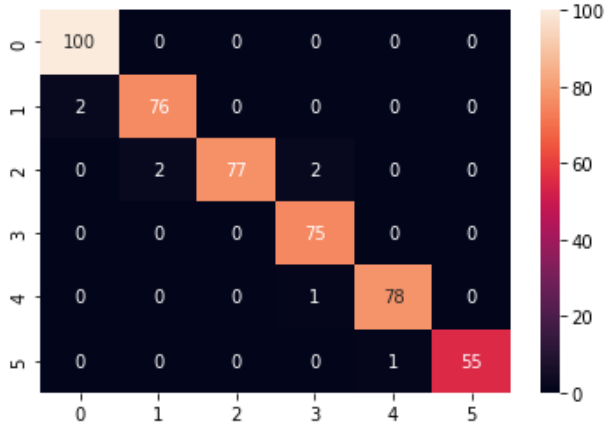


Figure 15 Confusion Matrix - KNN

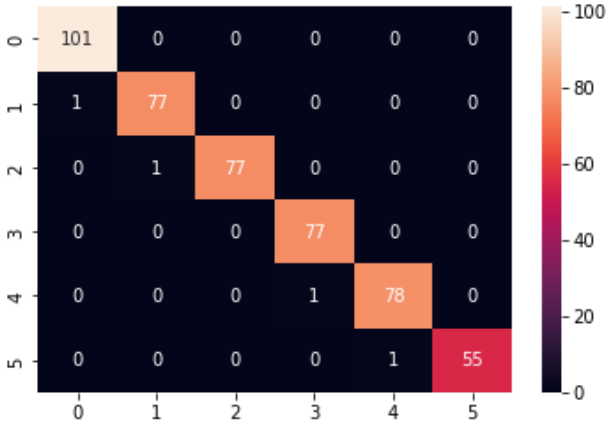
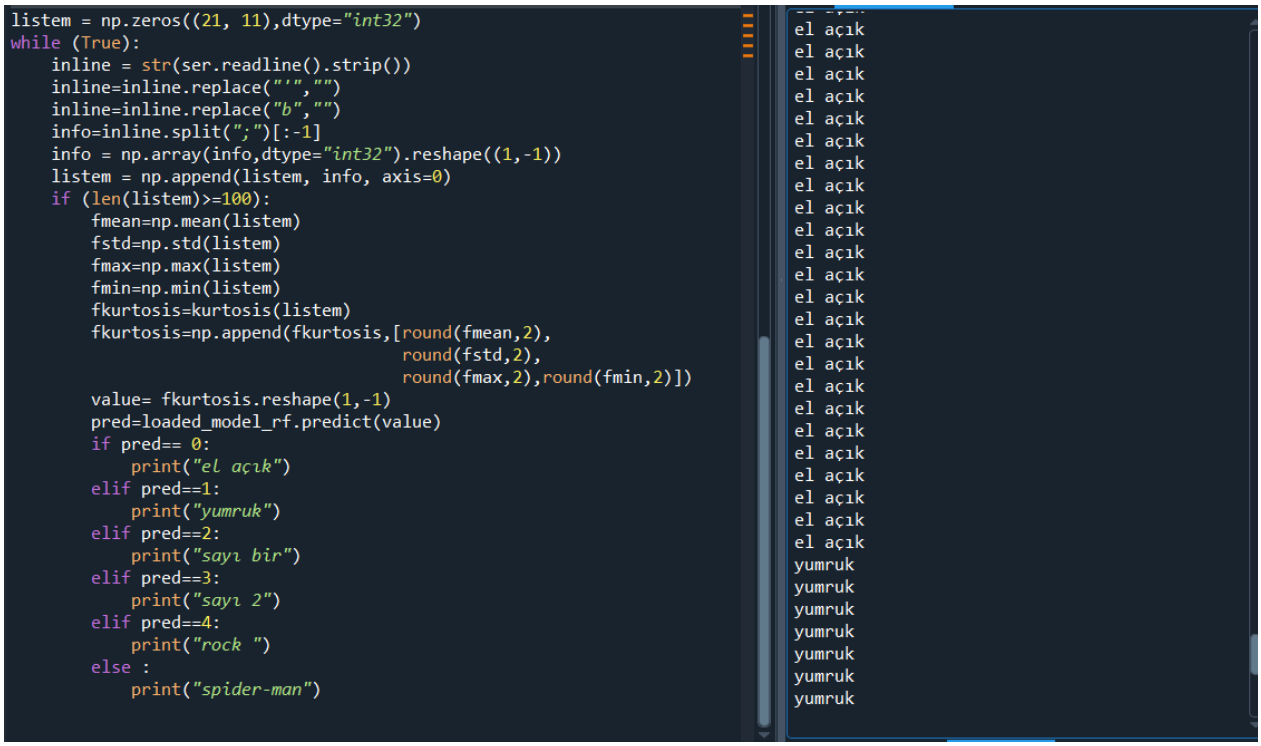


Figure 14 Confusion Matrix - RF

The model created with the K-Neighbors algorithm made an accurate prediction of 98 percent, while the model created with the Random Forest algorithm made an accurate prediction with 99 percent. Based on these results and figures, it can be said that the models can predict with high accuracy.

PART SIX Deployment

The trained model was tested in real time. For this, same feature extraction steps were applied to the incoming data. Although it could predict the fist movement with the open hand in real time, it was seen that some movements could not be predicted correctly.

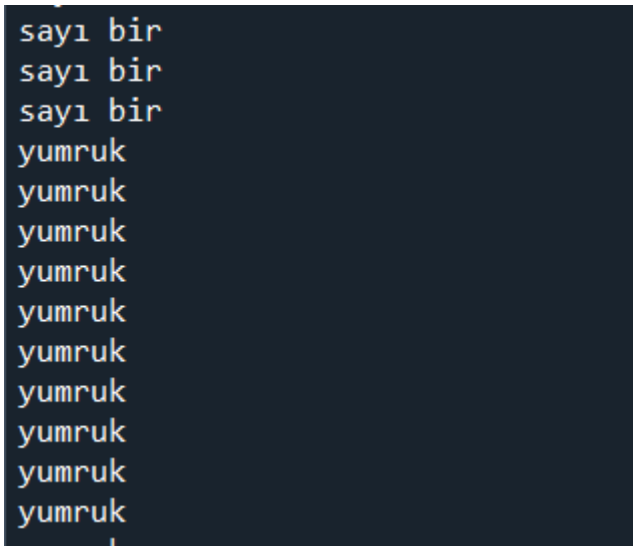


```
listem = np.zeros((21, 11),dtype="int32")
while (True):
    inline = str(ser.readline().strip())
    inline=inline.replace("'","")
    inline=inline.replace("b","")
    info=inline.split(";")[:-1]
    info = np.array(info,dtype="int32").reshape((1,-1))
    listem = np.append(listem, info, axis=0)
    if (len(listem)>=100):
        fmean=np.mean(listem)
        fstd=np.std(listem)
        fmax=np.max(listem)
        fmin=np.min(listem)
        fkurtosis=kurtosis(listem)
        fkurtosis=np.append(fkurtosis,[round(fmean,2),
                                       round(fstd,2),
                                       round(fmax,2),round(fmin,2)])
        value= fkurtosis.reshape(1,-1)
        pred=loaded_model_rf.predict(value)
        if pred== 0:
            print("el açık")
        elif pred==1:
            print("yumruk")
        elif pred==2:
            print("sayı bir")
        elif pred==3:
            print("sayı 2")
        elif pred==4:
            print("rock ")
        else :
            print("spider-man")
```

```
el açık
el açık
el açık
el açık
el açık
el açık
el açık
el açık
el açık
el açık
el açık
el açık
el açık
el açık
el açık
el açık
el açık
el açık
el açık
el açık
el açık
el açık
el açık
el açık
el açık
yumruk
yumruk
yumruk
yumruk
yumruk
yumruk
yumruk
```

Figure 16 Deployment and Prediciton

As an example, it was seen that when the class 2 sign was made, it was generally considered as a class 1.



```
sayı bir
sayı bir
sayı bir
yumruk
yumruk
yumruk
yumruk
yumruk
yumruk
yumruk
yumruk
yumruk
yumruk
```

PART SEVEN Conclusion

In this project, the capacitive touch sensor was used as a wristband and hand signals were tried to be predicted by using machine learning algorithms. To do this, proximity data of six different hand gestures were measured using the sensor, tagged, and the dataset was created. A 99% success rate was achieved with the random forest algorithm in machine learning trainings using the created data set and the model was saved for later testing. A different dataset was created with the same method to test the model. A 99 percent success rate was achieved in testing the new dataset using the trained model. Then, real time prediction was tried to be made by wearing the wristband and processing the incoming data. It was seen that some hand movements were predicted incorrectly. The decrease in the real time results can be attributed to reasons such as not performing the hand gestures same way, dislocation of the wristband, if it is removed, it is worn more tightly or loosely.