Assignment 3: Gesture Recognition

Group 2: Jakob Dittrich, David Reiter, Thomas Sickinger, Lea Franz

Setup

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import scipy.signal
from scipy.signal import savgol_filter, medfilt, wiener

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split

from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import recall_score, precision_score, accuracy_score
from sklearn.metrics import confusion_matrix, f1_score, classification_report
from sklearn.metrics import ConfusionMatrixDisplay

# color for plots
pltcolor = '#002b36'

# supress all warnings concerning functions that will be deprecated in the future
# to clean up the outputs

from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)
```

Preprocessed data

```
# save first 3 column names with gesture, person and sample name data
number_cols = len(pd.read_csv('gesture_recognition_preprocessed_data.csv',
        header=None).columns)
names_cols = ['gesture', 'person', 'sample']

# read in all sample names from dataset
for x in range(number_cols - 3):
  names_cols.append("C"+str(x))
```

```
# Save sample values in dataframe
df = pd.read_csv('gesture_recognition_preprocessed_data.csv')

# discard person and sample columns as they are not relevant at this time
# because we want to detect gestures in general
df.drop(columns=["person", "sample"], inplace=True)

# rename column names for x, y and z data as follows:
# X1 - X20 -> X1 -X20
# X1.1 - X1.20 -> Y1 - Y20
# X2.1 - X2.20 -> Z1 - Z20

for i in range(1, 21):
    df.rename(columns={"X"+str(i)+".1": "Y"+str(i)}, inplace=True)
    df.rename(columns={"X"+str(i)+".2": "Z"+str(i)}, inplace=True)

# print column names
df.columns
```

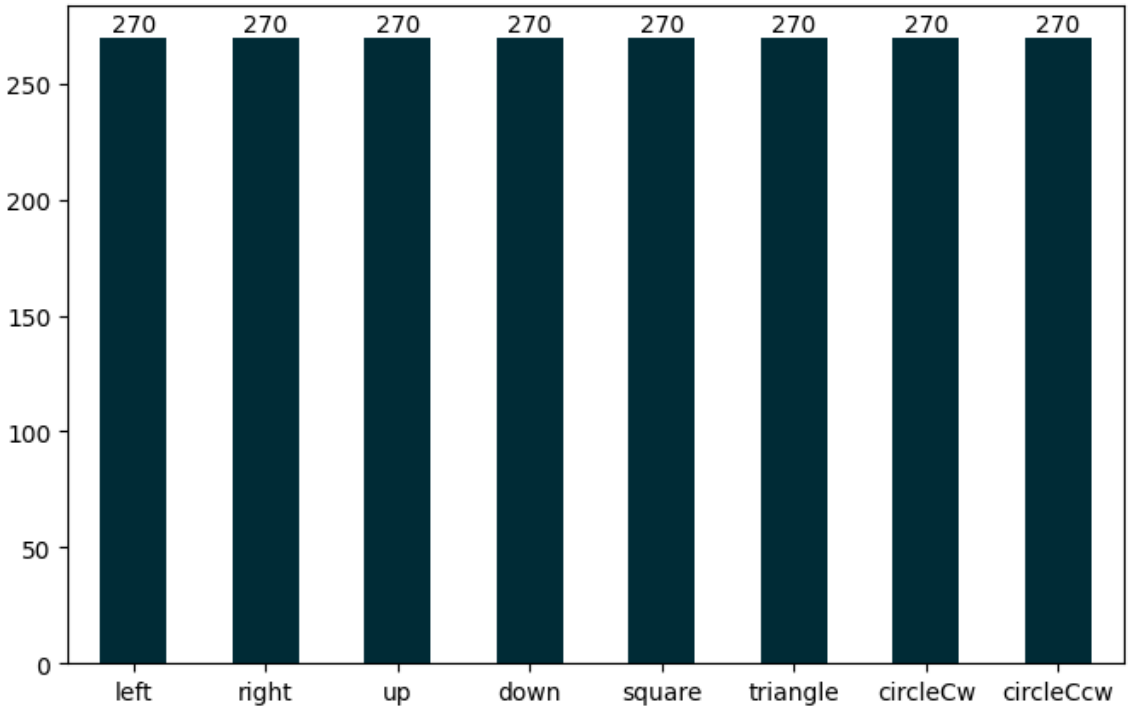
```
Index(['gesture', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8',
'X9', 'X10',
       'X11', 'X12', 'X13', 'X14', 'X15', 'X16', 'X17', 'X18', 'X19',
'X20',
       'Y1', 'Y2', 'Y3', 'Y4', 'Y5', 'Y6', 'Y7', 'Y8', 'Y9', 'Y10',
'Y11',
       'Y12', 'Y13', 'Y14', 'Y15', 'Y16', 'Y17', 'Y18', 'Y19', 'Y20',
'Z1',
       'Z2', 'Z3', 'Z4', 'Z5', 'Z6', 'Z7', 'Z8', 'Z9', 'Z10', 'Z11',
'Z12',
       'Z13', 'Z14', 'Z15', 'Z16', 'Z17', 'Z18', 'Z19', 'Z20'],
      dtype='object')
```

Data analysis

```
# extract with gestures are saved in dataset
gestures = df['gesture'].unique().tolist()

# print properties of dataset
print("There are "+str(len(df))+" samples with "+str(len(gestures))+" kinds of
        gestures: "+str(gestures))
```

```
There are 2160 samples with 8 kinds of gestures: ['left', 'right',
'up', 'down', 'square', 'triangle', 'circleCw', 'circleCcw']
```

```
# check if gestures are balanced across dataset
table = df['gesture'].value_counts()
ax = table.plot(kind='bar', rot=0, figsize=(8,5), color=pltcolor);
ax.bar_label(ax.containers[0]);
```



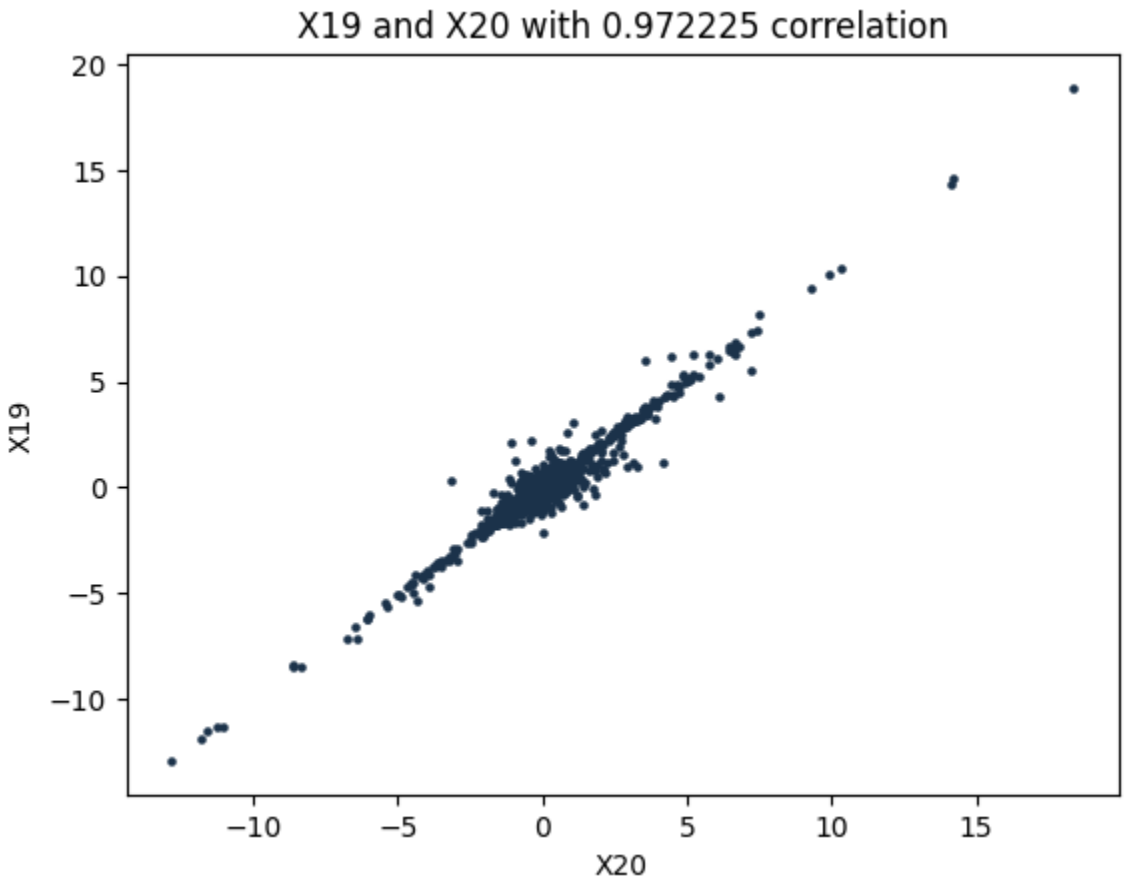
The plot shows that the classes are balanced and all include 270 samples.

```
# check correlation of features in dataset
df_corr = df
df_corr.corr().unstack().sort_values().drop_duplicates()
```

```
X5   Y6    -0.384871
     Y7    -0.383822
X6   Y7    -0.369919
X4   Y6    -0.355237
Y5   X4    -0.348510
               ...
X20  X18    0.911329
Z19  Z20    0.916037
X19  X18    0.953333
X20  X19    0.972225
X1   X1     1.000000
Length: 1771, dtype: float64
```

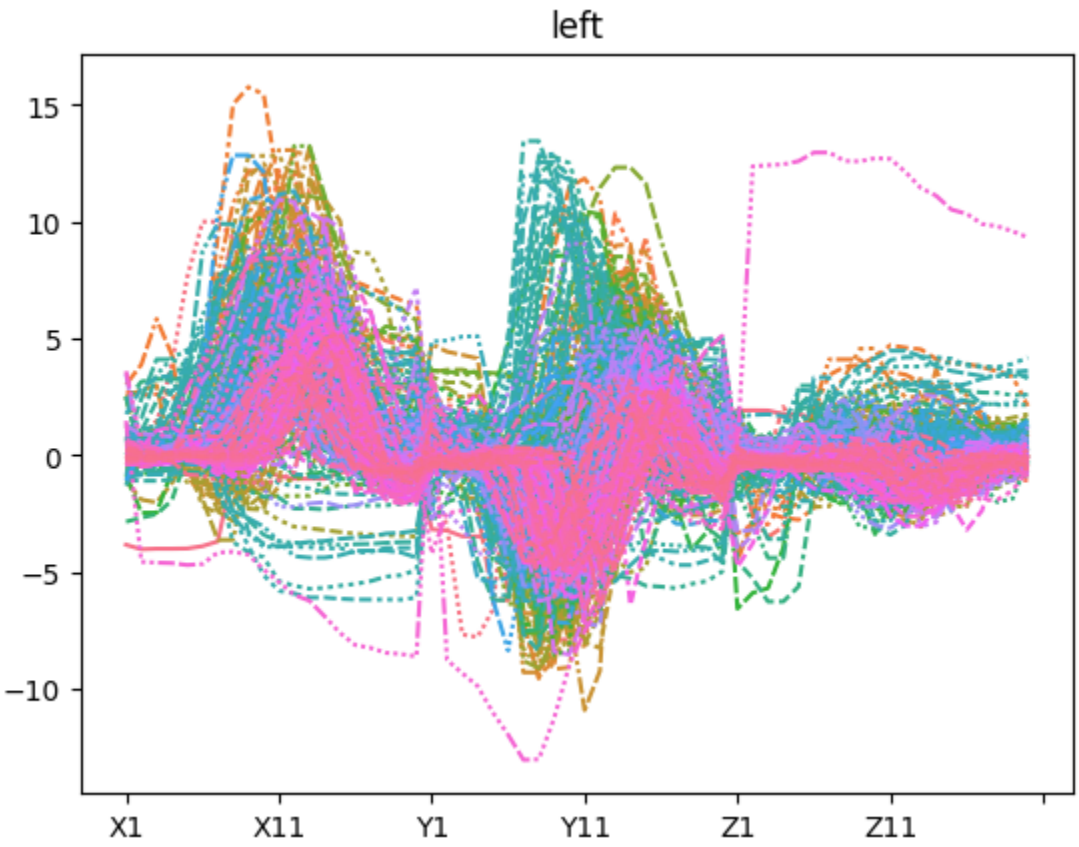
There are some highly correlated features such as X20 and X19, however there aren't any highly inverse correlated ones.

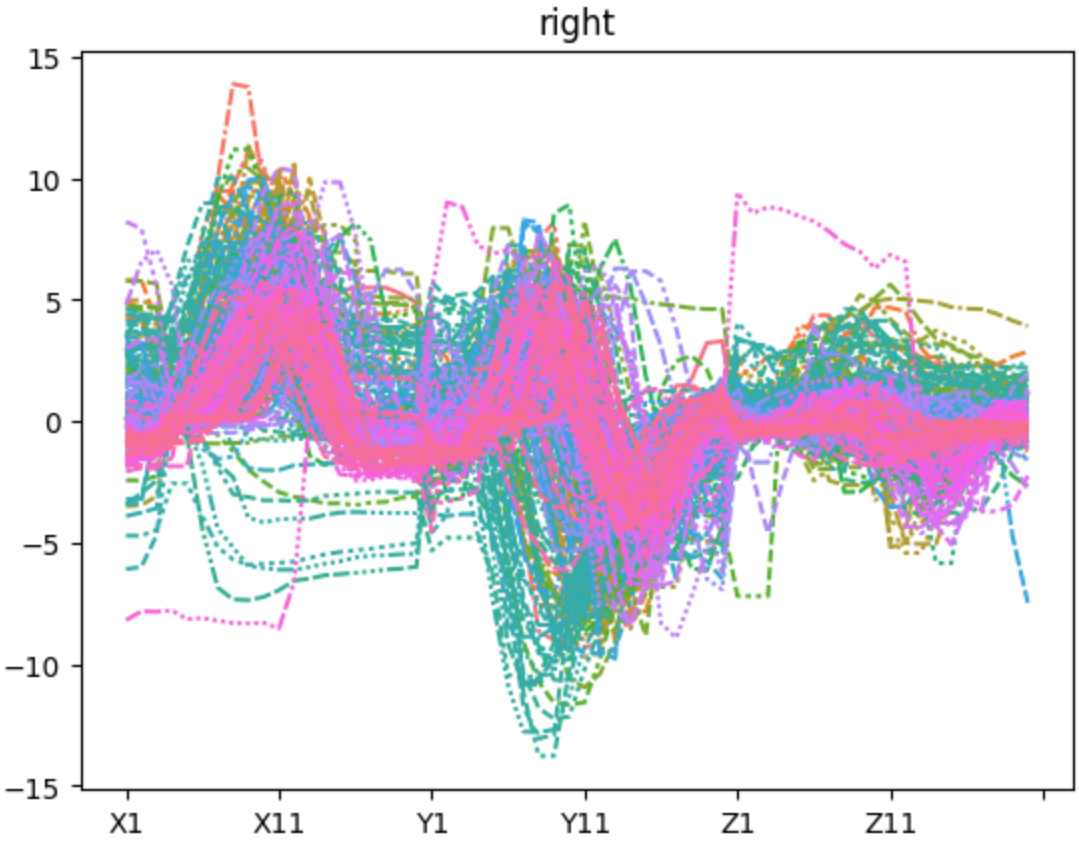
```
df.plot.scatter("X20", "X19", c="#1b324a", marker='.');
plt.title("X19 and X20 with 0.972225 correlation");
```

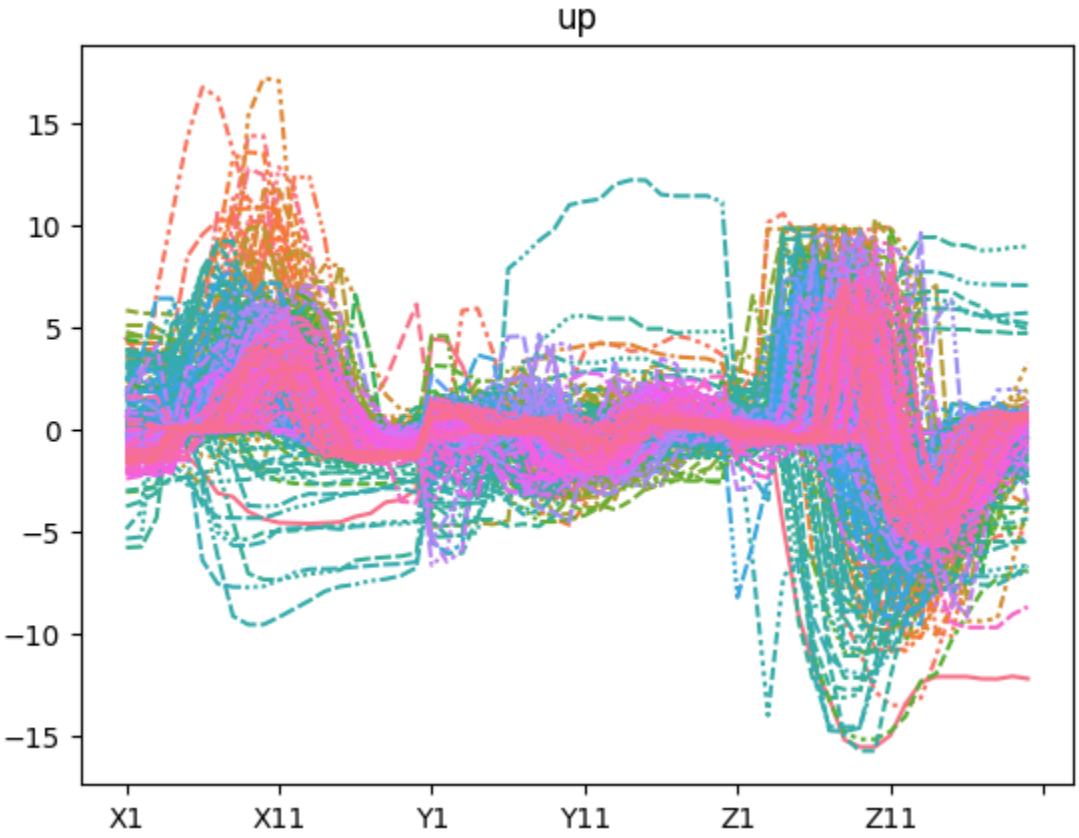


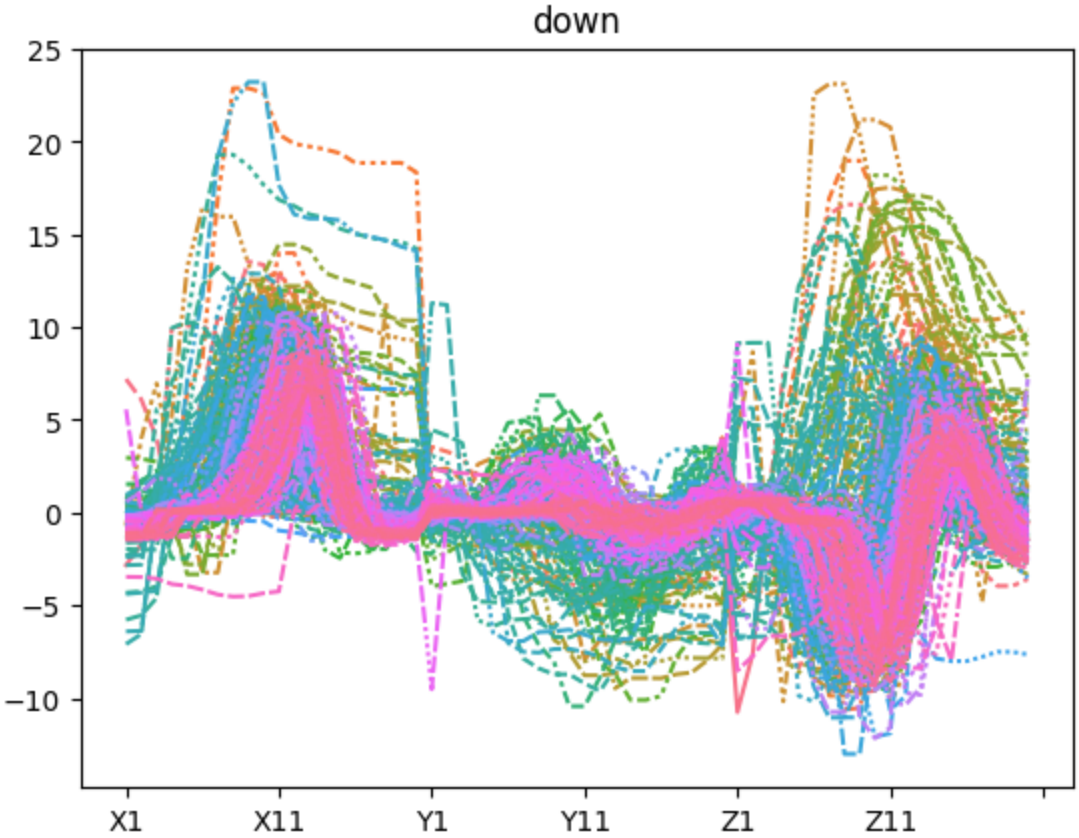
Visualization of gestures

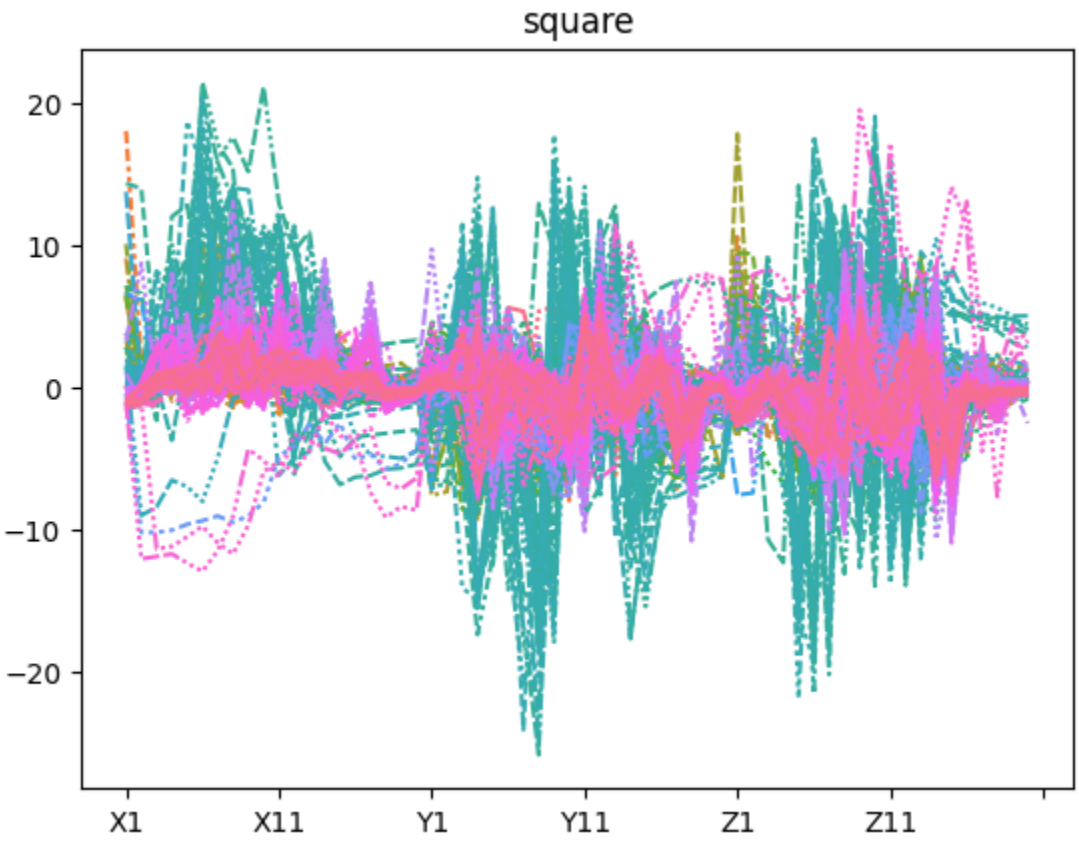
```
def visualize_gesture(gesture):
    df_temp = df.loc[df['gesture'] == gesture]
    del df_temp[df_temp.columns[0]]
    df_temp = df_temp.T

    sns.lineplot(df_temp, legend=False).set(
        xticks=range(0, len(df.columns), 10), title=gesture)

# plot all gestures with its 270 sample series in all 3 axes
for i, gest in enumerate(gestures):
    plt.figure(i)
    visualize_gesture(gest)
```

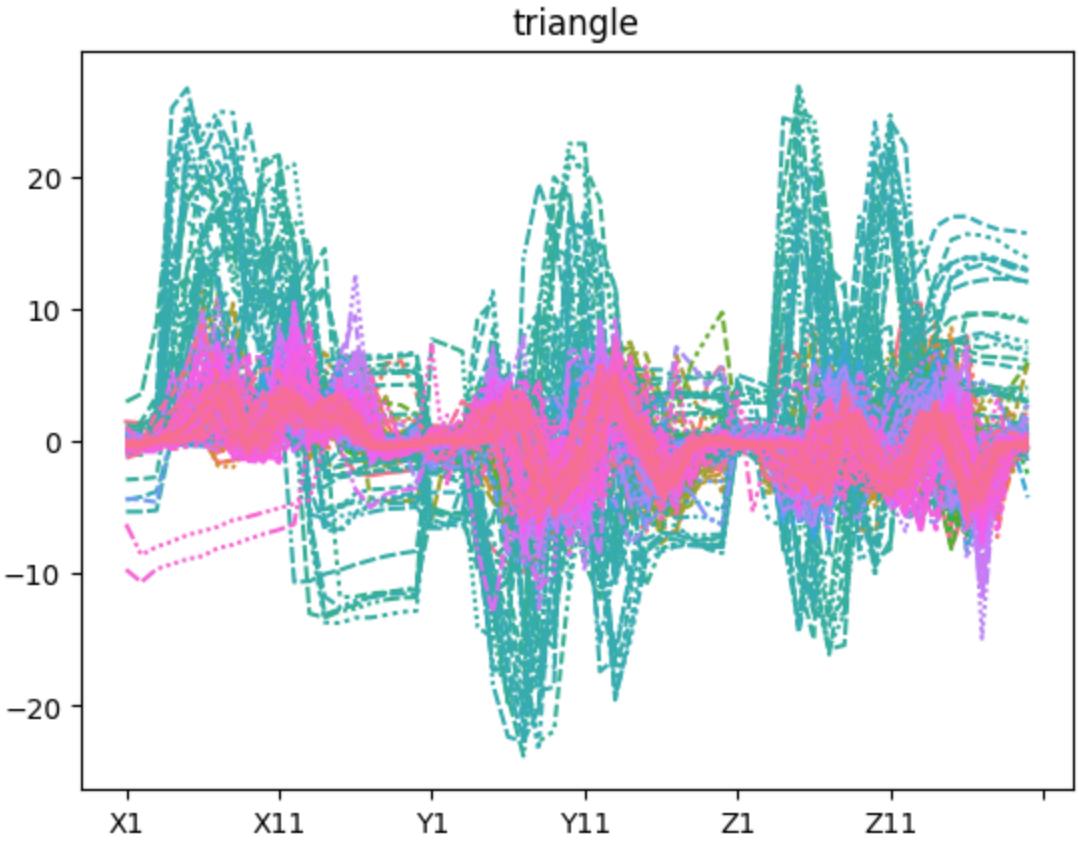

left
15
10
5
0
−5
−10
X1
X11
Y1
Y11
Z1
Z11

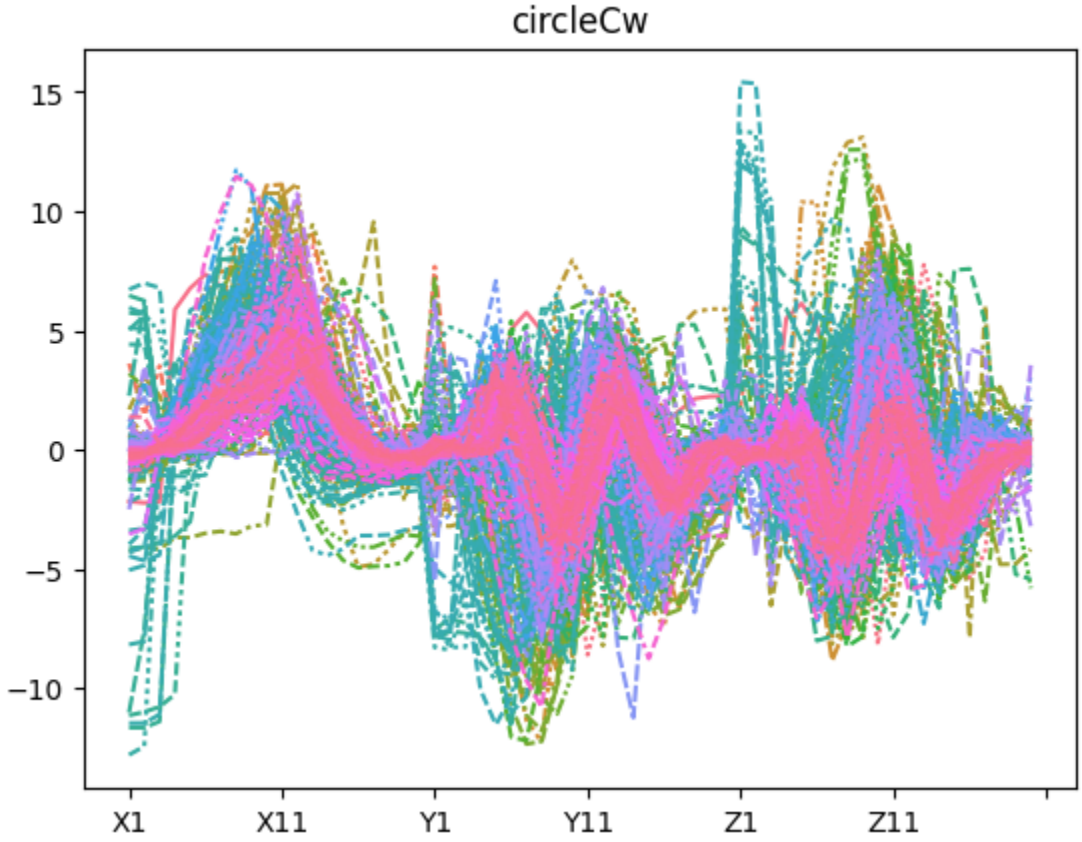

right
15
10
5
0
−5
−10
−15
X1
X11
Y1
Y11
Z1
Z11

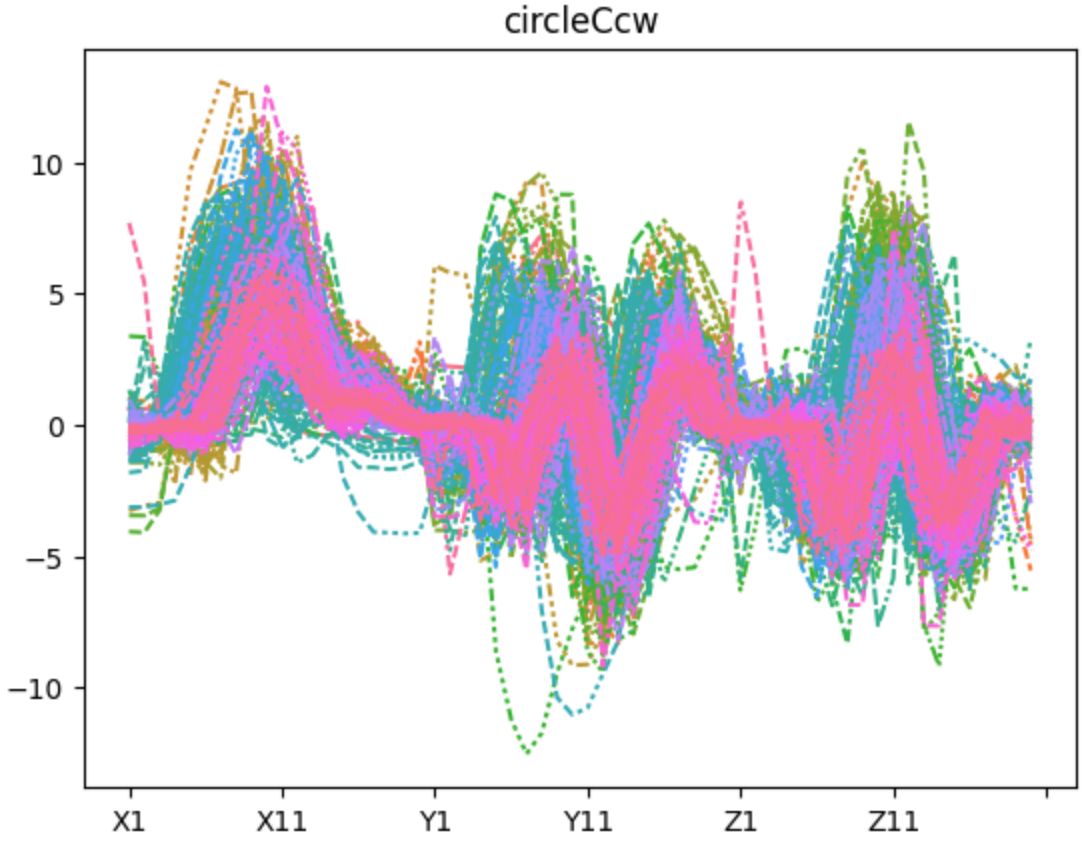

up
15
10
5
0
−5
−10
−15
X1
X11
Y1
Y11
Z1
Z11


down
25
20
15
10
5
0
−5
−10
X1
X11
Y1
Y11
Z1
Z11


square
20
10
0
−10
−20
X1
X11
Y1
Y11
Z1
Z11


triangle
20
10
0
−10
−20
X1
X11
Y1
Y11
Z1
Z11





Preprocessing

Filter

```
# create dataframe with only acceleration data
df_no_gesture = df.iloc[: , 1:]

# rolling mean -> NaN values
#df_filter = df_no_gesture.apply(func=lambda x: pd.Series(x).rolling(5).mean())

# rolling median -> NaN values
#df_filter = df_no_gesture.apply(func=lambda x: pd.Series(x).rolling(5).median())

# scipy savgol
# filter the data series with savgol filter to reduce noice
df_filter = df_no_gesture.apply(func=lambda x: savgol_filter(x, 5, 2))

# scipy medfilt: median filter to the input array
#df_filter = df_no_gesture.apply(func=lambda x: medfilt(x))

# scipy wiener
#df_filter = df_no_gesture.apply(func=lambda x: wiener(x))


# iir low pass filter -> also not really a gamechanger
#b, a = scipy.signal.iirfilter(4, Wn=2.5, fs=40, btype="low", ftype="butter")
#df_filter = df_no_gesture.apply(func=lambda x: scipy.signal.filtfilt(b, a,
        pd.Series(x)))

# check if any null values are now in dataframe
df_filter.isnull().values.ravel().sum()
```

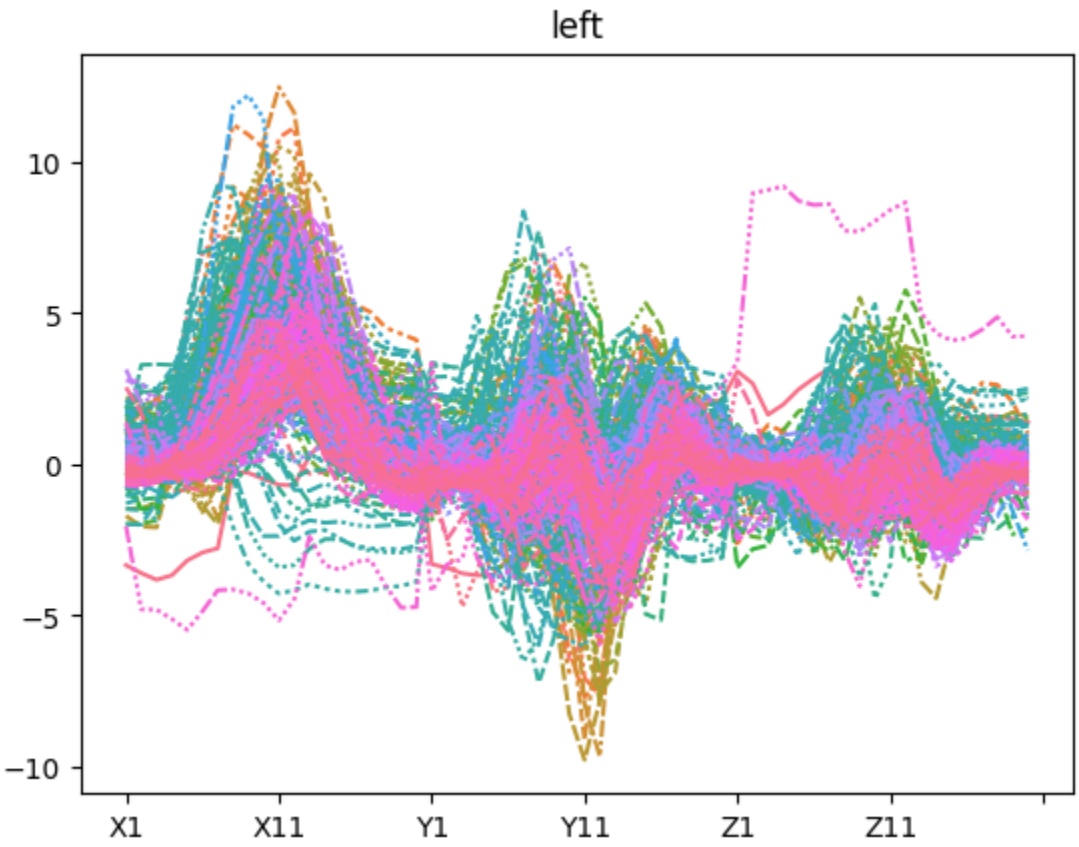
0

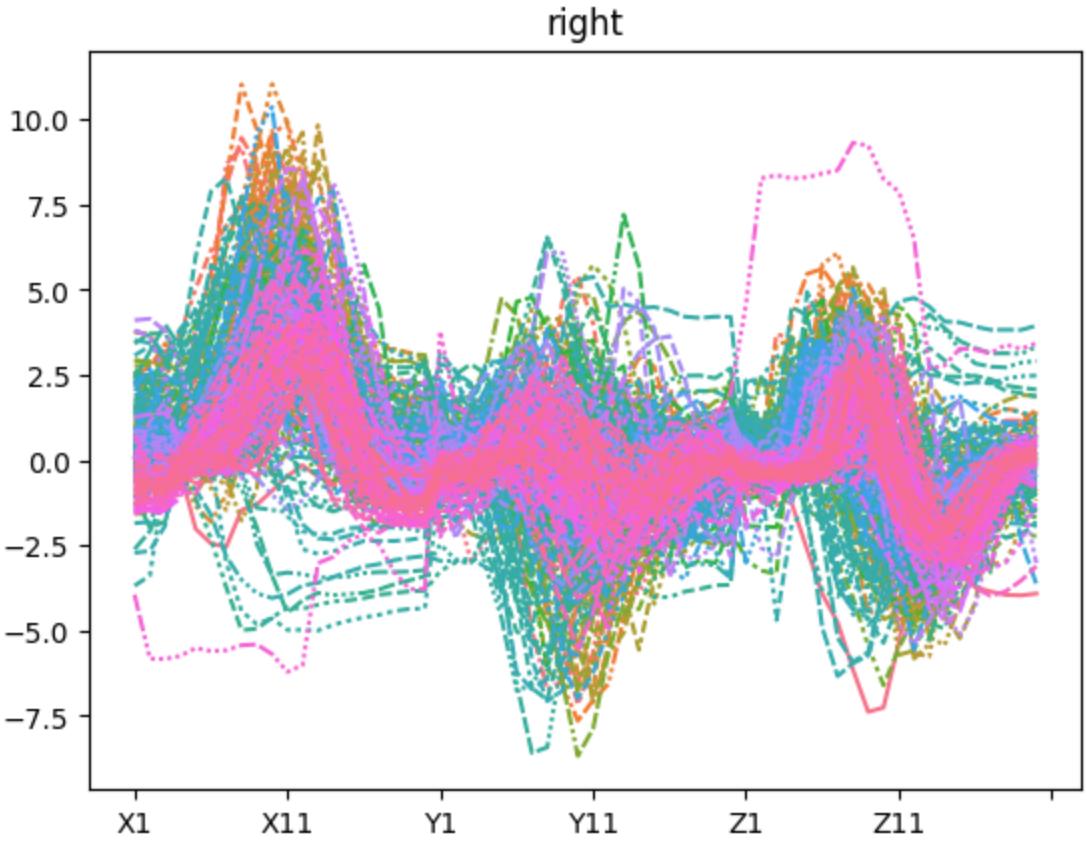
```
# add gesture names again to dataframe
df_filter.insert(loc=0, column='gesture', value=df['gesture'].to_numpy())
df = df_filter
#df.dropna() # for rolling mean or median
```

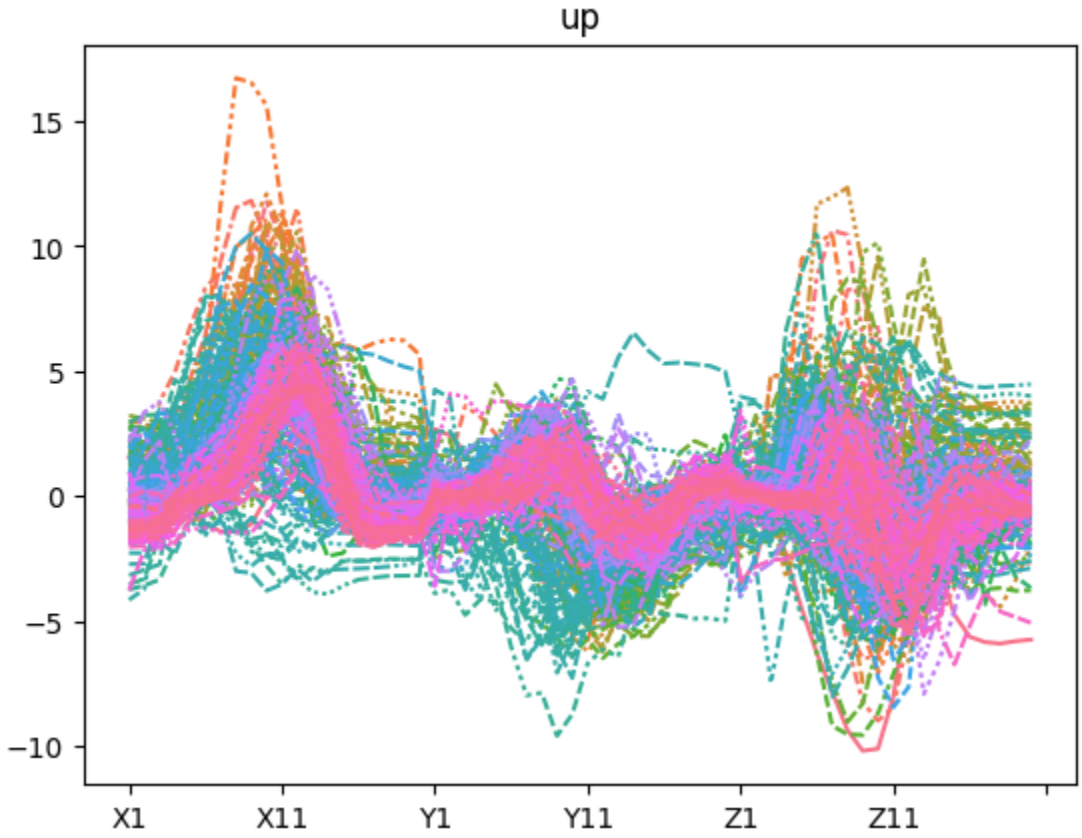
The savgol has proven to be the best option for the data as the gestures seem well distinguishable and it produces no problems when applying the filter.

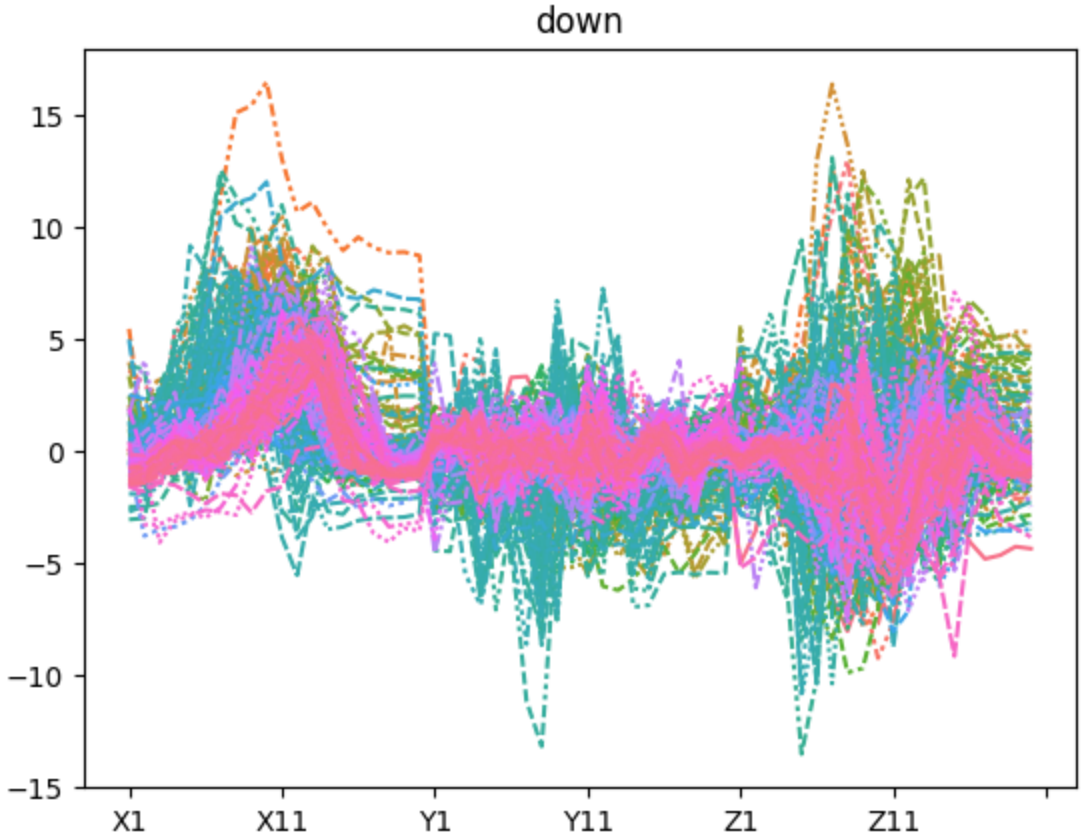
The filter with rolling mean and medium produced NaN values and therefore the number of samples would be reduced.

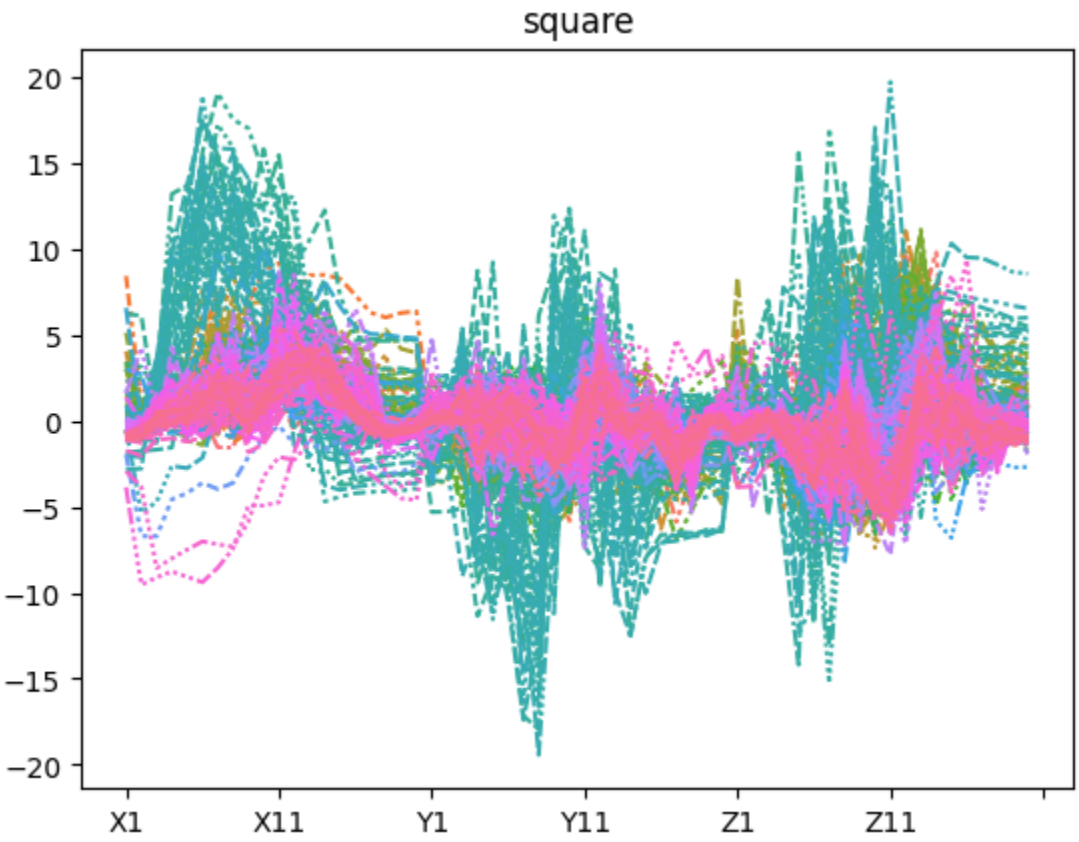
```
# plot filtered gestures
for i, gest in enumerate(gestures):
    plt.figure(i)
    visualize_gesture(gest)
```

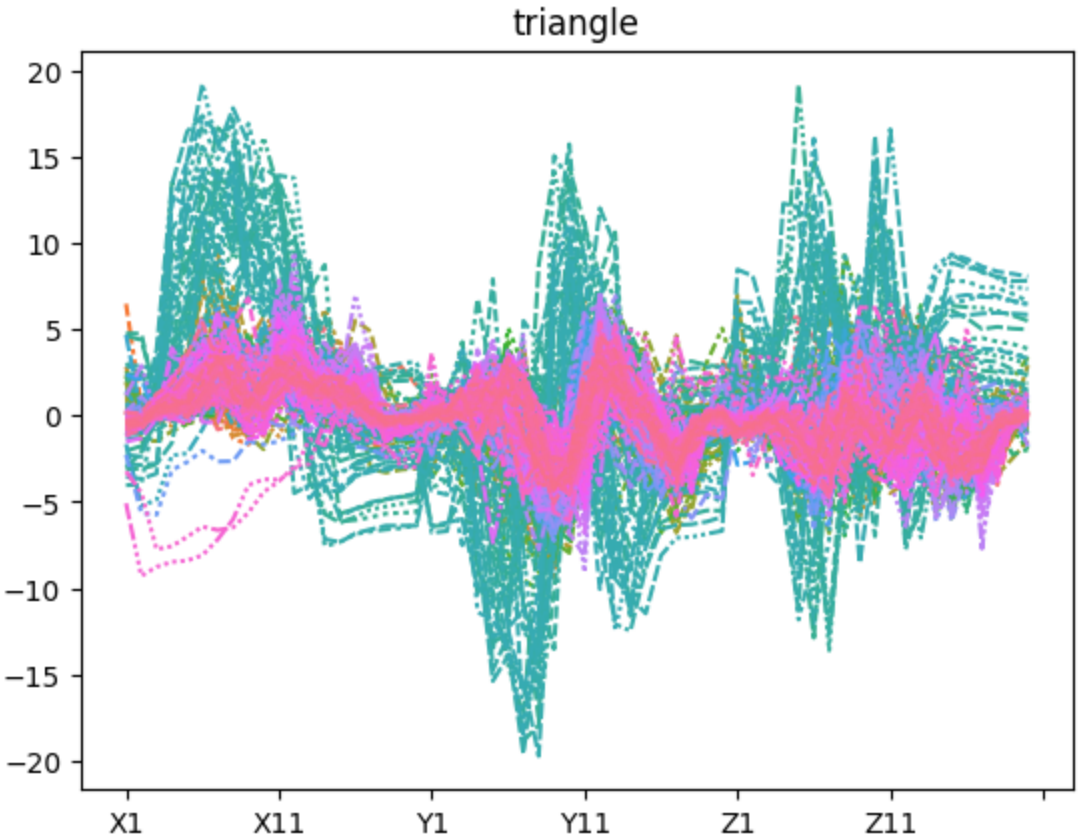

left
10
5
0
−5
−10
X1
X11
Y1
Y11
Z1
Z11


right
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
X1
X11
Y1
Y11
Z1
Z11


up
15
10
5
0
−5
−10
X1
X11
Y1
Y11
Z1
Z11


down
15
10
5
0
−5
−10
−15
X1
X11
Y1
Y11
Z1
Z11


square
20
15
10
5
0
−5
−10
−15
−20
X1
X11
Y1
Y11
Z1
Z11


triangle
20
15
10
5
0
−5
−10
−15
−20
X1
X11
Y1
Y11
Z1
Z11

Features

All the measured motion axes have the same length after preprocessing the data and are sampled with 20Hz. Because of that no features need to be interpolated to fit any desired length.

Original length of recording as features

The length of the original recording should be added as a feature. The original data is available in the raw data files for each axis.

```
raw_x = pd.read_csv('raw_data_wear_x.csv', header=None)
raw_y = pd.read_csv('raw_data_wear_y.csv', header=None)
raw_z = pd.read_csv('raw_data_wear_z.csv', header=None)
print('Number of columns in raw data files: x='+str(len(raw_x.columns)) +
      ", y="+str(len(raw_y.columns))+", z="+str(len(raw_z.columns)))
```

```
Number of columns in raw data files: x=427, y=427, z=427
```

As all raw data files include the same number of columns, the length of the recordings are taken from raw_data_wear_x. The columns for gesture, person and sample are not relevant for the recording length and therefore removed from the count.

```
# add new column containng the length of the original recording
df["recording_length"] = raw_x.count(axis='columns')-3
```

Outliers

The outliers were capped to the lower or upper limit instead of removing them to keep more information and improve the model performance.

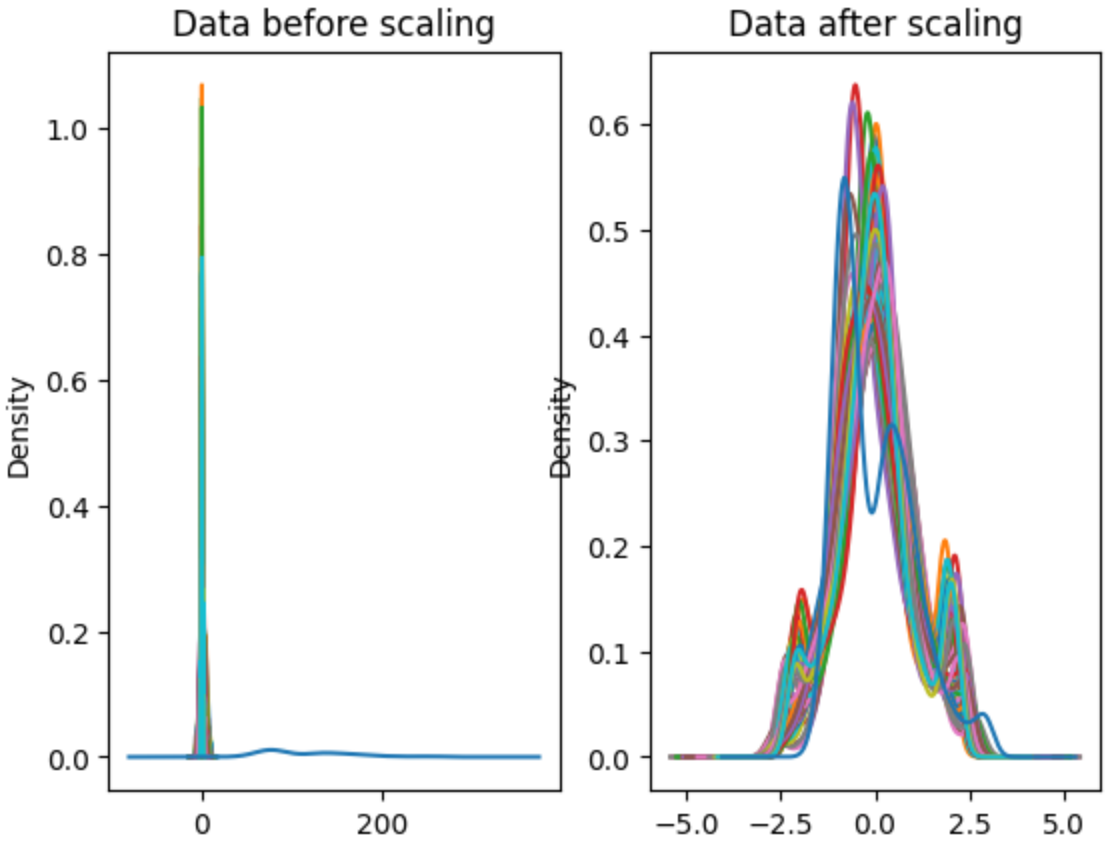
```
cols = df.columns.tolist()
cols.remove('gesture')

for col in cols:
    # calculate interquartile range (iqr)
    q25 = df[col].quantile(0.25)
    q75 = df[col].quantile(0.75)
    iqr = q75-q25

    # calculate lower and upper whisker
    q_hi = q75+(1.5*iqr)
    q_low = q25-(1.5*iqr)

    # cap data at higher and lower bounds
    df.loc[df[col] < q_low, col] = q_low
    df.loc[df[col] > q_hi, col] = q_hi
    df_filtered = df[(df[col] < q_hi) & (df[col] > q_low)]

print(str(len(df)-len(df_filtered)) +
      ' rows with outliers detected. Outliers were set to the lower or upper limit.')
```

```
28 rows with outliers detected. Outliers were set to the lower or
upper limit.
```

Scaling

The data gets scaled with the StandardScaler to change the distribution by which removing the mean and scaling each feature to unit variance.

```
# uniformly scale data
scaler = StandardScaler()
y_gestures = df['gesture']
X_transform = scaler.fit_transform(df.iloc[:, 1:].values)

# show data before and after uniform scaling
fig, (ax1, ax2) = plt.subplots(ncols=2)
ax1.set_title("Data before scaling")
ax2.set_title("Data after scaling")

df.plot.density(ax=ax1, legend=False)
df_scaled = pd.DataFrame(X_transform, columns=df.iloc[:, 1:].columns)
df_scaled.plot.density(ax=ax2, legend=False);
df = df_scaled
df.insert(loc=0, column='gesture', value=y_gestures)
```



Other features

New features are created using the magnitudes of the acceleromation data

```
x_data = df.iloc[:, 1:21]
y_data = df.iloc[:, 21:41]
z_data = df.iloc[:, 41:61]

axis_data = [{"name": "x", "data": x_data}, {
    "name": "y", "data": y_data}, {"name": "z", "data": z_data}]
```

Magnitude for each step

```
mag_data = pd.DataFrame()
for i in range(20):
    mag_list = []
    for j in range(len(x_data)):
        x = x_data.iloc[j].values[i]
        y = y_data.iloc[j].values[i]
        z = z_data.iloc[j].values[i]
        mag_list.append(np.sqrt(x**2 + y**2 + z**2))
    mag_data["MAG"+str(i+1)] = mag_list
```

Mean Magnitude

```
df['mean_mag'] = mag_data.mean(axis=1)
```

Minimum Magnitude

```
df['min_mag'] = mag_data.min(axis=1)
```

Maximum Magnitude

```
df['max_mag'] = mag_data.max(axis=1)
```

Range of Magnitude

```
df['max_mag'] = abs(mag_data.max(axis=1) - mag_data.min(axis=1))
```

Mean of the acceleration data

```
mean_x = df.iloc[:, 1:21].mean(axis=1)
mean_y = df.iloc[:, 21:41].mean(axis=1)
mean_z = df.iloc[:, 41:61].mean(axis=1)
df['mean_x'] = mean_x
df['mean_y'] = mean_y
df['mean_z'] = mean_z
```

Mean Absolute Deviation of the acceleration data

```
df['mad_x'] = df.iloc[:, 1:21].mad(axis=1)
df['mad_y'] = df.iloc[:, 21:41].mad(axis=1)
df['mad_z'] = df.iloc[:, 41:61].mad(axis=1)
```

Root Mean Square of the acceleration data

```
df['rms_x'] = df.iloc[:, 1:21].apply(func=lambda x: np.sqrt(np.mean(x**2)), axis=1)
df['rms_y'] = df.iloc[:, 21:41].apply(func=lambda x: np.sqrt(np.mean(x**2)), axis=1)
df['rms_z'] = df.iloc[:, 31:61].apply(func=lambda x: np.sqrt(np.mean(x**2)), axis=1)
```

Variance features

```
for axis in axis_data:
    variance_arr = axis.get('data').var(axis=1)
    df['variance_'+axis.get('name')] = variance_arr

df['mean_variance'] = df.iloc[:,-3:].mean(axis=1)
```

Standard deviation features

```
for axis in axis_data:
    sd_arr = axis.get('data').var(axis=1)
    df['sd_'+axis.get('name')] = sd_arr

df['mean_sd'] = df.iloc[:,-3:].mean(axis=1)
df['mag_sd'] = mag_data.var(axis=1)
```

Peaks

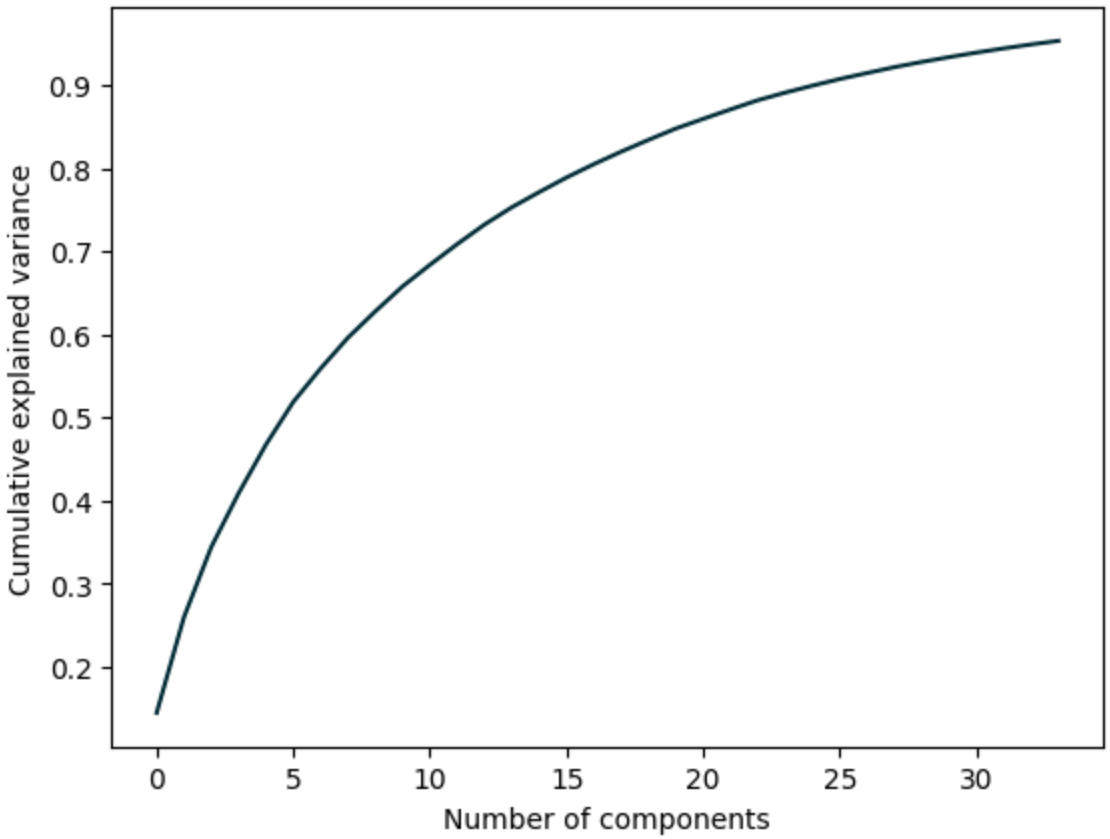
The times the acceleration data went over the mean of the axis

```
mean_x = df.iloc[:, 1:21].mean(axis=1)
mean_y = df.iloc[:, 21:41].mean(axis=1)
mean_z = df.iloc[:, 41:61].mean(axis=1)

x = df.iloc[:, 1:21].T
y = df.iloc[:, 21:41].T
z = df.iloc[:, 41:61].T

df['peaks_x'] = x[x > mean_x].count()
df['peaks_y'] = y[y > mean_y].count()
df['peaks_z'] = z[z > mean_z].count()

df['peaks_mag'] = mag_data[mag_data > df['mean_mag']].count(axis=1)
```

PCA

This plot shows the explained variance ratio when 95% of variance is kept. 95% was chosen because otherwise the model performance would decrease too much. Feature reduction with PCA wil be applied later when training the models using a pipeline.

```
pca = PCA(n_components=0.95)
X_pca = pca.fit(X_transform)
plt.plot(np.cumsum(pca.explained_variance_ratio_), color=pltcolor)
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
plt.show()
```

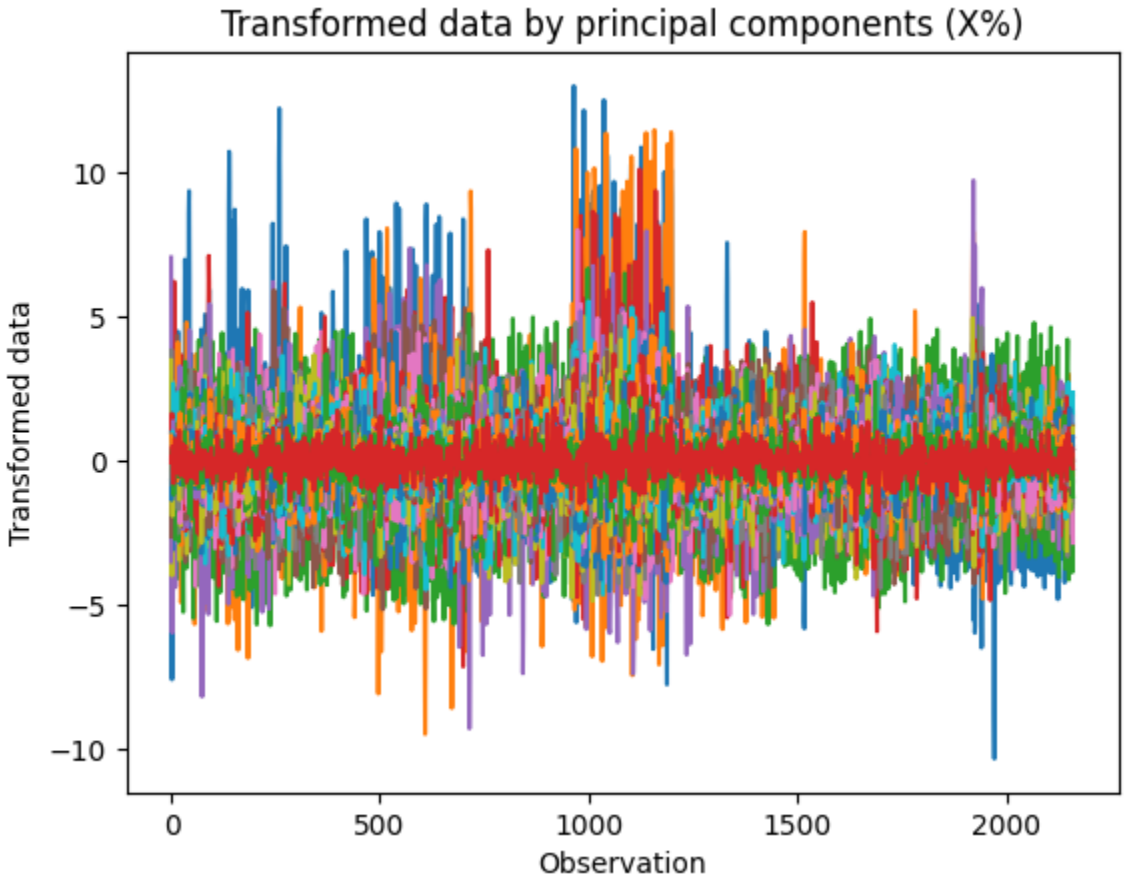


PCA with 0.95% variance means 34 components in this case.

```
#print shape of transformed PCA
X_pca = pca.transform(X_transform)
X_pca.shape
```

```
(2160, 34)
```

```
plt.plot(X_pca)
plt.xlabel('Observation')
plt.ylabel('Transformed data')
plt.title('Transformed data by principal components (95%)');
```



Models

Feature Selection

We tried to use lesser Features, but the models get significantly lower accuracy(5-10%) after the feature selection. Because of this we used all features.

The data is split into training and test data with a 80/20 split

```
# split dataset in train and test data
X = df.values[:,1:]
Y = df.values[:,0:1]


X_train, X_test, y_train, y_test = train_test_split(
    X, y_gestures, test_size=0.2)

# function to print scores of a model according to the predicted labels from the grid
        search
def print_model_result(predicted_labels):
    print("Recall: ", recall_score(y_test, predicted_labels, average=None))
    print("Recall Average: ", recall_score(
        y_test, predicted_labels, average="micro"))
    print("Precision: ", precision_score(
        y_test, predicted_labels, average=None))
    print("Precision Average: ", precision_score(
```

```
        y_test, predicted_labels, average="micro"))
    print("F1-Score: ", f1_score(y_test, predicted_labels, average=None))
    print("Accuracy: %.2f  ," % accuracy_score(y_test, predicted_labels,
          normalize=True), accuracy_score(y_test, predicted_labels, normalize=False))

    print("Number of samples:", y_test.shape[0])
    cmd = ConfusionMatrixDisplay(confusion_matrix(
        y_test, predicted_labels), display_labels=gestures)
    fig, ax = plt.subplots(figsize=(8,8));
    cmd.plot(cmap="Blues", ax=ax)
```

KNN

```
knn_hyperparameters = list(range(1, 31))

knn = KNeighborsClassifier()
param_grid = dict(n_neighbors=knn_hyperparameters)

# defining parameter range
grid = GridSearchCV(KNeighborsClassifier(), param_grid, cv=10,
                    scoring='accuracy',
                    return_train_score=False, verbose=1, n_jobs=-1)

# fitting the model for grid search
grid_search = grid.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 30 candidates, totalling 300 fits
```

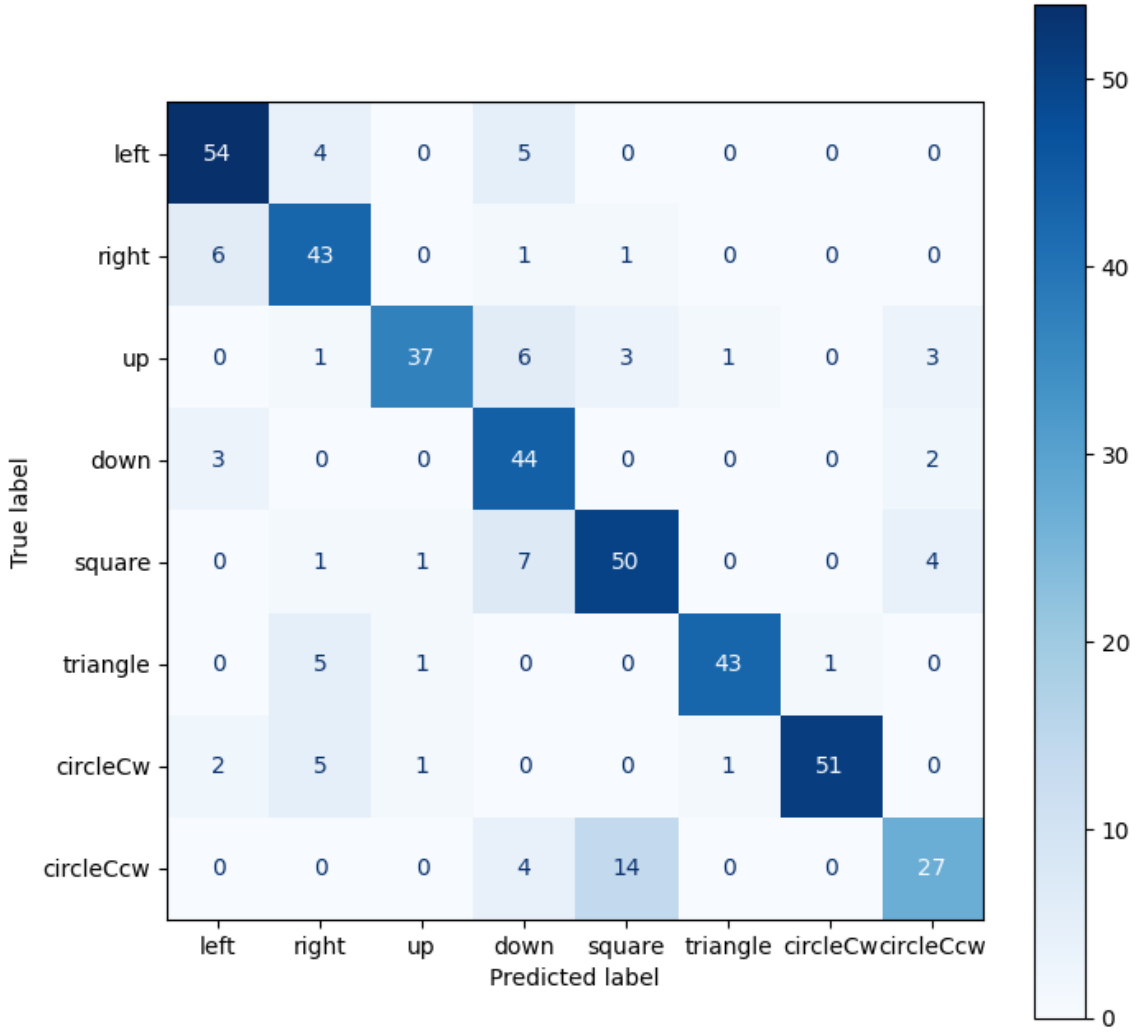
```
accuracy = grid_search.best_score_ * 100
print("Best params: "+str(grid_search.best_params_))
print("Accuracy for our training dataset with tuning is : {:.2f}%"\
    .format(accuracy))
```

```
Best params: {'n_neighbors': 8}
Accuracy for our training dataset with tuning is : 82.12%
```

```
predicted_labels = grid_search.best_estimator_.predict(X_test)
predicted_labels_knn = predicted_labels

best_knn_model = grid_search.best_estimator_
gs_knn = grid_search

print_model_result(predicted_labels)
```

```
Recall:  [0.85714286 0.84313725 0.7254902  0.89795918 0.79365079 0.86
 0.85       0.6        ]
Recall Average:  0.8078703703703703
Precision:  [0.83076923 0.72881356 0.925      0.65671642 0.73529412
0.95555556
 0.98076923 0.75       ]
Precision Average:  0.8078703703703703
```

```
F1-Score:  [0.84375    0.78181818 0.81318681 0.75862069 0.76335878
0.90526316
 0.91071429 0.66666667]
Accuracy: 0.81  , 349
Number of samples: 432
```



Random Forest

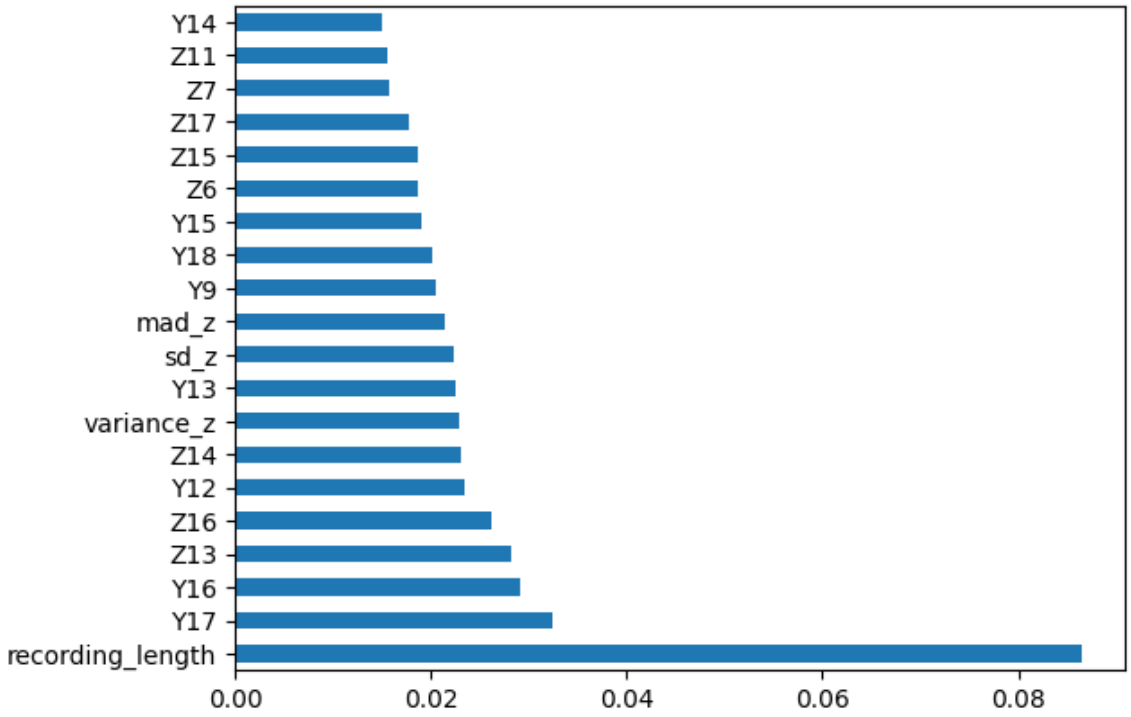
```
tuning_params_rf = {
    'max_depth': [30, 40, 50, 60],
    'max_features': ['sqrt'],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10],
    'n_estimators': [200, 400, 600]}


param_grid_rf = {}

for key, value in tuning_params_rf.items():
    hyperparam_key = "classify__" + key
    param_grid_rf[hyperparam_key] = value
```

```
gs = GridSearchCV(RandomForestClassifier(), param_grid=tuning_params_rf,
                  cv=10, scoring="accuracy", n_jobs=-1)
gs.fit(X_train, y_train);

print('Best parameters: '+str(gs.best_params_))
print('Best accuracy score: '+str(gs.best_score_))
print('Importances of the features: ', gs.best_estimator_.feature_importances_)
```

```
Best parameters: {'max_depth': 30, 'max_features': 'sqrt',
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 400}
Best accuracy score: 0.8836705202312138
Importances of the features:  [0.00761207 0.00756906 0.00643513
0.00723247 0.00586779 0.00770059
 0.00711186 0.00627177 0.01206717 0.0124174  0.00737755 0.00881104
 0.0080765  0.00723709 0.0087985  0.00932863 0.0081802  0.01007655
 0.00755369 0.00610772 0.00513028 0.00612469 0.00801022 0.00568806
 0.01159603 0.01082797 0.00862614 0.01388689 0.02045729 0.01018871
 0.00706929 0.02341036 0.02260073 0.01501592 0.01910401 0.02912235
 0.0325294  0.020183   0.00590431 0.0055877  0.00791164 0.00542977
 0.00557694 0.00635017 0.01402247 0.01873498 0.01571413 0.01229961
 0.00789628 0.0139069  0.01556611 0.01187612 0.02815165 0.02307978
 0.01865262 0.02627153 0.01776298 0.00569766 0.00566075 0.0066827
 0.08656836 0.00748056 0.00517129 0.00423192 0.00729051 0.00556883
 0.00659572 0.00561917 0.01055348 0.02142456 0.0080493  0.00850668
 0.0079481  0.00640261 0.01067185 0.02287474 0.0067203  0.00546591
 0.0105214  0.02240343 0.00706651 0.00450429 0.00292432 0.00242058
 0.00287464 0.        ]
```

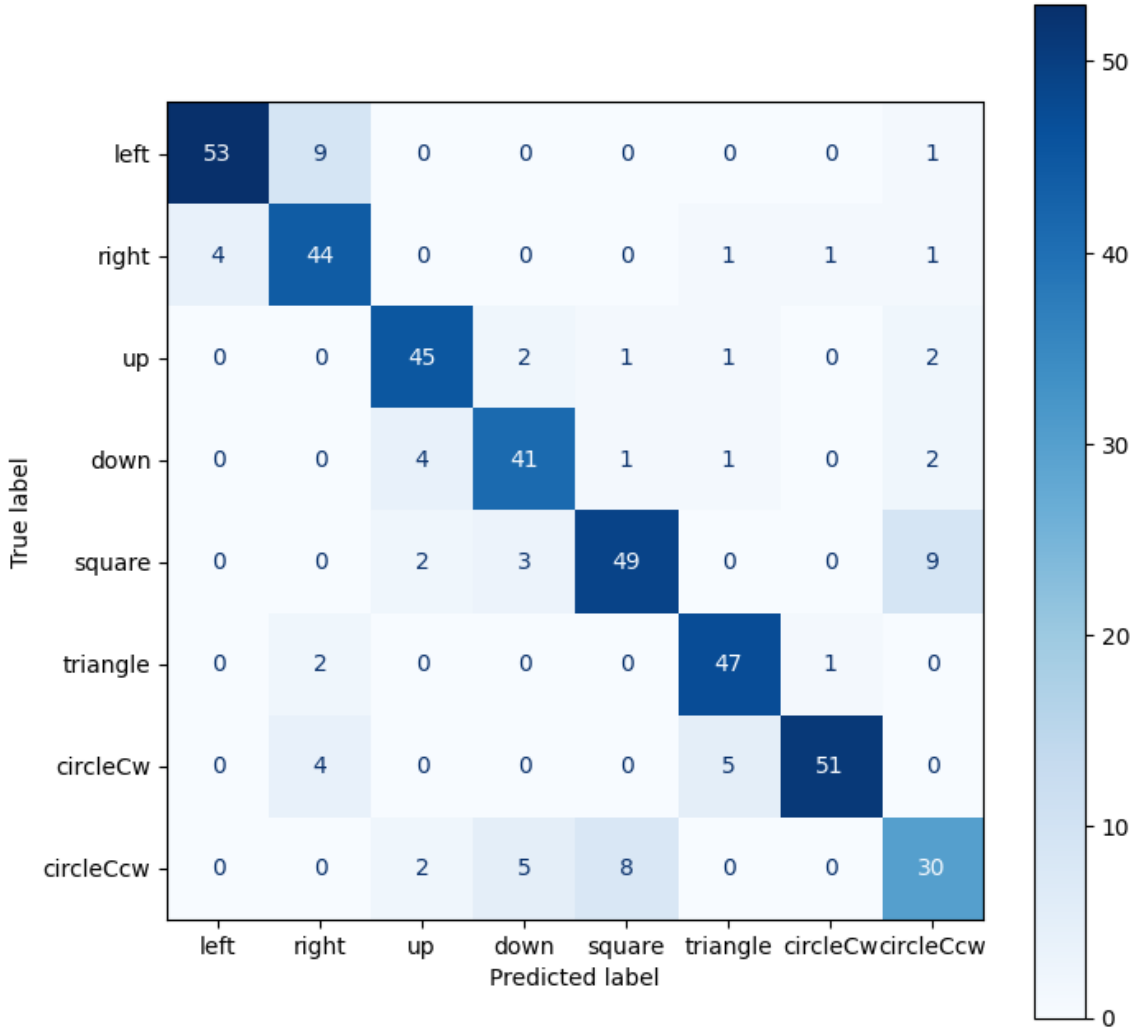
20 most important features

```
feature_importances = pd.Series(gs.best_estimator_.feature_importances_,
        index=df.iloc[:, 1:].columns)
feature_importances.nlargest(20).plot(kind='barh');
```



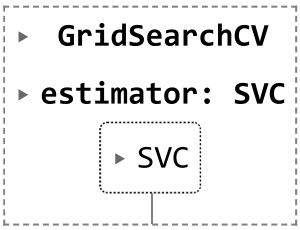

```
predicted_labels = gs.best_estimator_.predict(X_test)

best_rf_model = grid_search.best_estimator_
predicted_labels_rf = predicted_labels
gs_rf = gs

print_model_result(predicted_labels)
```

```
Recall:  [0.84126984 0.8627451  0.88235294 0.83673469 0.77777778 0.94
 0.85       0.66666667]
Recall Average:  0.8333333333333334
Precision:  [0.92982456 0.74576271 0.8490566  0.80392157 0.83050847
0.85454545
 0.96226415 0.66666667]
Precision Average:  0.8333333333333334
F1-Score:  [0.88333333 0.8        0.86538462 0.82       0.80327869
0.8952381
 0.90265487 0.66666667]
Accuracy: 0.83  , 360
Number of samples: 432
```



SVM

```
tuning_params_svm = {"C": [0.001, 0.01, 0.1, 1, 10],
                     "gamma": [0.001, 0.01, 0.1, 1, 10], "kernel": ['rbf']}
param_grid_svm = {}

for key, value in tuning_params_svm.items():
    hyperparam_key = "classify__" + key
    param_grid_svm[hyperparam_key] = value

gs = GridSearchCV(SVC(), param_grid=tuning_params_svm,
                  cv=10, scoring="accuracy", n_jobs=-1)
gs.fit(X_train, y_train)
```

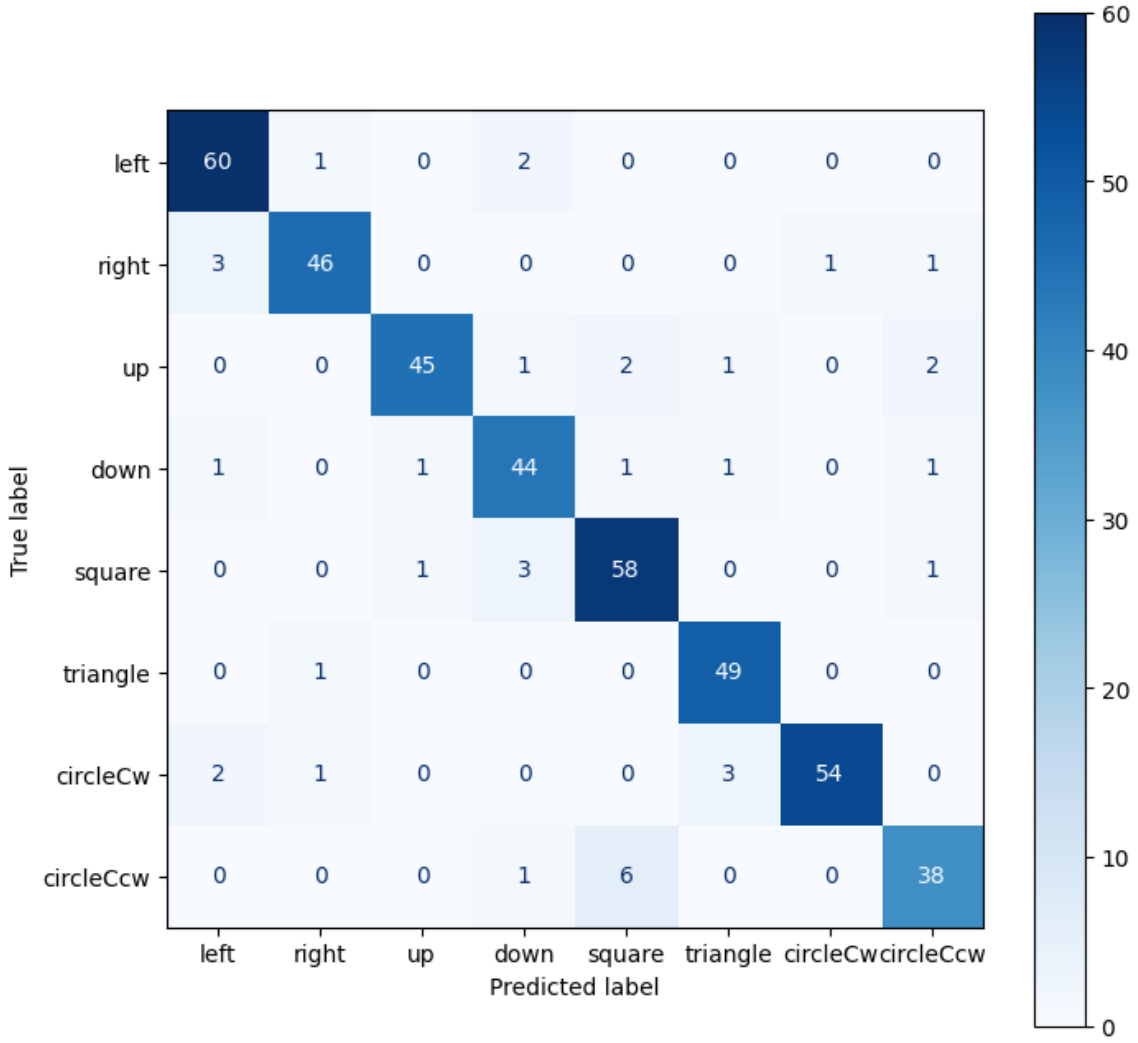



```
print('Best parameters: '+str(gs.best_params_))
print('Best accuracy score: '+str(gs.best_score_))
```

```
Best parameters: {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
Best accuracy score: 0.9091342922435812
```

```
predicted_labels = gs.best_estimator_.predict(X_test)
predicted_labels_svm = predicted_labels
gs_svm = gs

best_svm_model = gs.best_estimator_

print_model_result(predicted_labels)
```

```
Recall:  [0.95238095 0.90196078 0.88235294 0.89795918 0.92063492 0.98
 0.9        0.84444444]
Recall Average:  0.9120370370370371
Precision:  [0.90909091 0.93877551 0.95744681 0.8627451  0.86567164
0.90740741
 0.98181818 0.88372093]
Precision Average:  0.9120370370370371
F1-Score:  [0.93023256 0.92       0.91836735 0.88       0.89230769
0.94230769
 0.93913043 0.86363636]
Accuracy: 0.91  , 394
Number of samples: 432
```



PCA

```
# Initialze the estimators
clf1 = KNeighborsClassifier()
clf2 = RandomForestClassifier()
clf3 = SVC()

# hyperparameters for each dictionary
param1 = {}
param1['classifier__n_neighbors'] = knn_hyperparameters
param1['classifier'] = [clf1]

param2 = {}
param2['classifier__max_depth'] = tuning_params_rf.get('max_depth')
param2['classifier__max_features'] = tuning_params_rf.get('max_features')
param2['classifier__min_samples_leaf'] = tuning_params_rf.get(
    'min_samples_leaf')
param2['classifier__min_samples_split'] = tuning_params_rf.get(
    'min_samples_split')
param2['classifier__n_estimators'] = tuning_params_rf.get('n_estimators')
param2['classifier'] = [clf2]

param3 = {}
param3['classifier__C'] = tuning_params_svm.get('C')
```

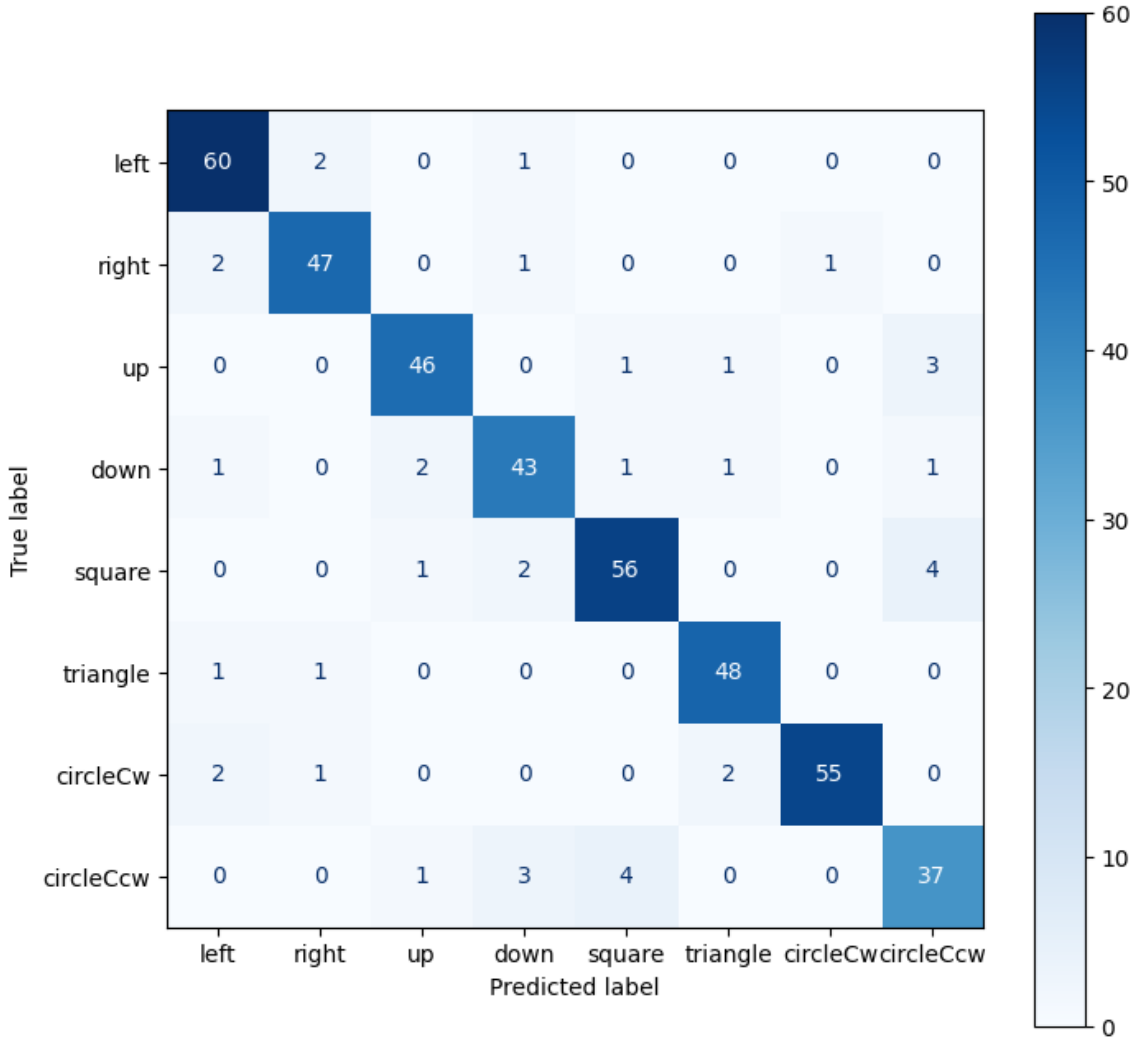
```
param3['classifier__gamma'] = tuning_params_svm.get('gamma')
param3['classifier__kernel'] = tuning_params_svm.get('kernel')
param3['classifier'] = [clf3]

pipe = Pipeline([
    ('normalization', StandardScaler()),
    ('pca', PCA(n_components=0.95)),
    ('classifier', clf1)
])
params = [param1, param2, param3]

# pca grid search
gs_pca = GridSearchCV(pipe, params, cv=10, n_jobs=-1,
                      scoring='accuracy').fit(X_train, y_train)

print('Best classifier and parameters: '+str(gs_pca.best_params_))
print('Best accuracy score: '+str(gs_pca.best_score_))
print('Best estimator explained variance: ' +
      str(gs_pca.best_estimator_.steps[1][1].explained_variance_))
```

```
Best classifier and parameters: {'classifier': SVC(C=10, gamma=0.01),
'classifier__C': 10, 'classifier__gamma': 0.01, 'classifier__kernel':
'rbf'}
Best accuracy score: 0.9114598736389301
Best estimator explained variance: [16.47429336  8.60806875
5.99262717  5.22713177  4.42380286  3.41282762
  3.37868863  2.60300385  2.28992552  2.07012873  1.91911976
1.86770835
  1.64156598  1.56254581  1.45229852  1.39045369  1.30537951
1.12263519
  1.08593707  1.07866656  0.99763421  0.95486961  0.91251062
0.82391312
  0.79115343  0.76210515  0.73569325  0.70678641  0.67911027
0.64770503
  0.56231921  0.50651356  0.48775875  0.46280398  0.43175662
0.40791765
  0.37386691  0.34091765  0.31988791]
```

```
predicted_labels_pca = gs_pca.best_estimator_.predict(X_test)
best_pca_model = gs_pca.best_estimator_
print_model_result(predicted_labels_pca)
```

```
Recall:  [0.95238095 0.92156863 0.90196078 0.87755102 0.88888889 0.96
 0.91666667 0.82222222]
Recall Average:  0.9074074074074074
Precision:  [0.90909091 0.92156863 0.92       0.86       0.90322581
0.92307692
 0.98214286 0.82222222]
Precision Average:  0.9074074074074074
```

```
F1-Score:  [0.93023256 0.92156863 0.91089109 0.86868687 0.896
0.94117647
 0.94827586 0.82222222]
Accuracy: 0.91  , 392
Number of samples: 432
```



Final Result

Model comparison

```
model_names = [
    'KNN',
    'Random Forest',
    'SVM',
    'PCA SVM'
]
```

Best score after training

```
model_gs = [
    gs_knn,
```

```
    gs_rf,
    gs_svm,
    gs_pca
]

for i, model in enumerate(model_gs):
    print(model_names[i]+": Best accuracy: "+str(model.best_score_))
```

```
KNN: Best accuracy: 0.8240623739749966
Random Forest: Best accuracy: 0.8836705202312138
SVM: Best accuracy: 0.9091342922435812
PCA SVM: Best accuracy: 0.9114598736389301
```

Model evaluation with the best hyperparameters

```
best_model_predictions = [
    predicted_labels_knn,
    predicted_labels_rf,
    predicted_labels_svm,
    predicted_labels_pca
]

for i, pred in enumerate(best_model_predictions):
    print(model_names[i])
    print(classification_report(y_test, pred))
```

```
KNN
              precision    recall  f1-score   support

   circleCcw       0.83      0.86      0.84        63
    circleCw       0.73      0.84      0.78        51
        down       0.93      0.73      0.81        51
        left       0.66      0.90      0.76        49
       right       0.74      0.79      0.76        63
      square       0.96      0.86      0.91        50
    triangle       0.98      0.85      0.91        60
          up       0.75      0.60      0.67        45

    accuracy                           0.81       432
   macro avg       0.82      0.80      0.81       432
weighted avg       0.82      0.81      0.81       432

Random Forest
              precision    recall  f1-score   support

   circleCcw       0.93      0.84      0.88        63
    circleCw       0.75      0.86      0.80        51
```

down	0.85	0.88	0.87	51
left	0.80	0.84	0.82	49
right	0.83	0.78	0.80	63
square	0.85	0.94	0.90	50
triangle	0.96	0.85	0.90	60
up	0.67	0.67	0.67	45
accuracy			0.83	432
macro avg	0.83	0.83	0.83	432
weighted avg	0.84	0.83	0.83	432

SVM

	precision	recall	f1-score	support
circleCcw	0.91	0.95	0.93	63
circleCw	0.94	0.90	0.92	51
down	0.96	0.88	0.92	51
left	0.86	0.90	0.88	49
right	0.87	0.92	0.89	63
square	0.91	0.98	0.94	50
triangle	0.98	0.90	0.94	60
up	0.88	0.84	0.86	45
accuracy			0.91	432
macro avg	0.91	0.91	0.91	432
weighted avg	0.91	0.91	0.91	432

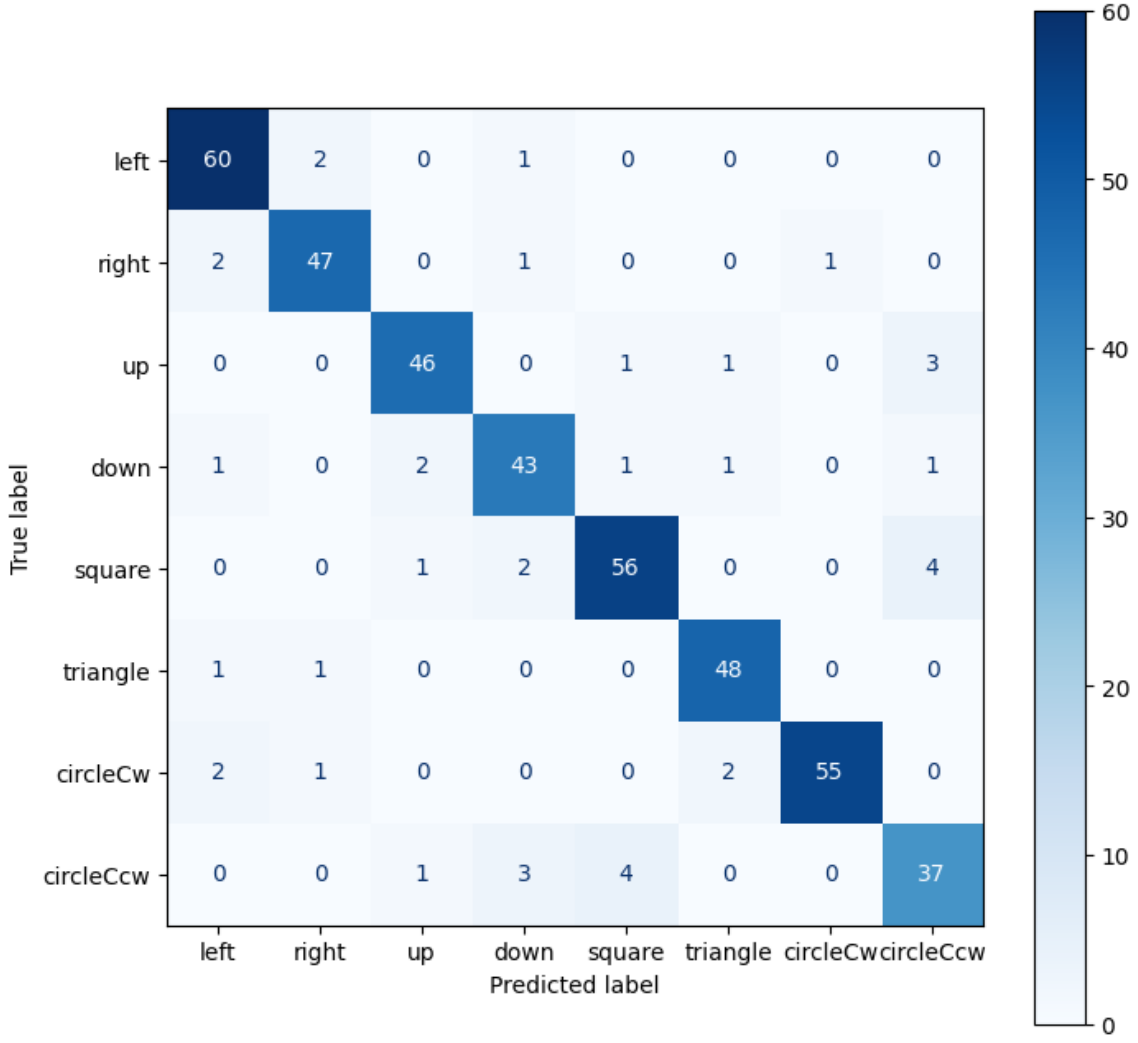
PCA SVM

	precision	recall	f1-score	support
circleCcw	0.91	0.95	0.93	63
circleCw	0.92	0.92	0.92	51
down	0.92	0.90	0.91	51
left	0.86	0.88	0.87	49
right	0.90	0.89	0.90	63
square	0.92	0.96	0.94	50
triangle	0.98	0.92	0.95	60
up	0.82	0.82	0.82	45
accuracy			0.91	432
macro avg	0.91	0.91	0.90	432
weighted avg	0.91	0.91	0.91	432

Final decision

The model we choose is the SVM with PCA because it shows the most accuracy among all gestures.

```
# predict on validation data
results = gs_pca.predict(X_test);
# show result confusion matrix
cmd = ConfusionMatrixDisplay(confusion_matrix(y_test, results),
                             display_labels=df['gesture'].unique())
fig, ax = plt.subplots(figsize=(8,8))
cmd.plot(ax=ax, cmap="Blues");
```



The confusion matrix shows that the model provides really good prediction results. The SVM predicts almost all labels accurately and the errors are spread evenly accross all gestures.