

# gesture\_recognition

January 12, 2023

## 1 Gesture Recognition

### Dataset

- The dataset contains different arm gestures from different participants.
  - Recording device and wearing position: smartwatch (LG Watch G) running Android Wear, worn at right wrist.
  - Recorded data: 3 axes acceleration measured in G [ $9.81\text{m/s}^2$ ], with a target sampling rate of 50Hz.
  - Gestures, participants and samples: 8 gestures, 9 participants. Each participant performed each gesture 30 times, which results in a total of 240 samples per participants, 270 samples per gesture and 2160 samples in total in the data set.
  - The publication describing the dataset in more detail can be found here: [https://ambientintelligence.aalto.fi/team\\_old/findling/pdfs/publications/Kefer\\_16\\_ComparingP](https://ambientintelligence.aalto.fi/team_old/findling/pdfs/publications/Kefer_16_ComparingP)
- Provided csv files:
  - Acceleration recordings are split per axis into separate csv files (one file per axis). This implies that e.g. the Nth line of each file belong to the same gesture, participant and sample and represent the according x, y and z axis acceleration recordings.
  - Csv file columns are in this order:
    1. gesture
    2. participantNr
    3. sampleNr
    4. N acceleration values (different lengths per sample)

**Task** Goal is to build (a) successful gesture recognition model(s). The model should learn to distinguish between N gesture acceleration recording, then be able to decide for new recordings which gesture it shows. Load data, preprocess it, and extract features (see hints below). \* Do a gallery dependent data partitioning and train models using cross validation. Compare different models and feature extraction approaches by their cross validation results, and for the selected “best” model show at least the confusion matrix for heldback test partition over different gestures. Discuss what the best feature extraction/model configuration is you found, and if there are gestures that are harder to distinguish than others. \* *Bonus objective:* do gallery independent data partitioning instead using leave subject out cross validation (LSOCV, see slides).

### 1.1 Load data

- Load the `gesture_recognition_preprocessed_data.csv` file into a pandas dataframe for further processing
- Rename the columns to use X0-19, Y0-19, Z0-19 instead of X1-20, X1.1-20, X2.1-.20

- Separate person and sample columns into their own dataframe, as they are not needed yet

```
[111]: import pandas as pd

df = pd.read_csv("gesture_recognition_preprocessed_data.csv")
# rename the columns
rename = {old: chr(ord('X') + ((i - 3) // 20)) + ' ' + str((i - 3) % 20) if i >= 3
          else old for i, old in
          enumerate(df.columns.values)}
df.rename(columns=rename, inplace=True)

# separate the person and sample columns, as they are only needed for gallery
# independent
persons = df[["person", "sample"]]
df.drop(columns=["person", "sample"], inplace=True)
df.info()
```

<class 'pandas.core.frame.DataFrame'>			
RangeIndex: 2160 entries, 0 to 2159			
Data columns (total 61 columns):			
#	Column	Non-Null Count	Dtype
---	---	-----	----
0	gesture	2160 non-null	object
1	X0	2160 non-null	float64
2	X1	2160 non-null	float64
3	X2	2160 non-null	float64
4	X3	2160 non-null	float64
5	X4	2160 non-null	float64
6	X5	2160 non-null	float64
7	X6	2160 non-null	float64
8	X7	2160 non-null	float64
9	X8	2160 non-null	float64
10	X9	2160 non-null	float64
11	X10	2160 non-null	float64
12	X11	2160 non-null	float64
13	X12	2160 non-null	float64
14	X13	2160 non-null	float64
15	X14	2160 non-null	float64
16	X15	2160 non-null	float64
17	X16	2160 non-null	float64
18	X17	2160 non-null	float64
19	X18	2160 non-null	float64
20	X19	2160 non-null	float64
21	Y0	2160 non-null	float64
22	Y1	2160 non-null	float64
23	Y2	2160 non-null	float64
24	Y3	2160 non-null	float64
25	Y4	2160 non-null	float64

```
26    Y5        2160 non-null   float64
27    Y6        2160 non-null   float64
28    Y7        2160 non-null   float64
29    Y8        2160 non-null   float64
30    Y9        2160 non-null   float64
31    Y10       2160 non-null   float64
32    Y11       2160 non-null   float64
33    Y12       2160 non-null   float64
34    Y13       2160 non-null   float64
35    Y14       2160 non-null   float64
36    Y15       2160 non-null   float64
37    Y16       2160 non-null   float64
38    Y17       2160 non-null   float64
39    Y18       2160 non-null   float64
40    Y19       2160 non-null   float64
41    Z0        2160 non-null   float64
42    Z1        2160 non-null   float64
43    Z2        2160 non-null   float64
44    Z3        2160 non-null   float64
45    Z4        2160 non-null   float64
46    Z5        2160 non-null   float64
47    Z6        2160 non-null   float64
48    Z7        2160 non-null   float64
49    Z8        2160 non-null   float64
50    Z9        2160 non-null   float64
51    Z10       2160 non-null   float64
52    Z11       2160 non-null   float64
53    Z12       2160 non-null   float64
54    Z13       2160 non-null   float64
55    Z14       2160 non-null   float64
56    Z15       2160 non-null   float64
57    Z16       2160 non-null   float64
58    Z17       2160 non-null   float64
59    Z18       2160 non-null   float64
60    Z19       2160 non-null   float64
dtypes: float64(60), object(1)
memory usage: 1.0+ MB
```

```
[112]: persons.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2160 entries, 0 to 2159
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
--- 
 0   person   2160 non-null   int64  
 1   sample   2160 non-null   int64  
dtypes: int64(2)
```

memory usage: 33.9 KB

## 1.2 Data Preprocessing

- Additional preprocessing like filtering, outlier removal and scaling

### 1.2.1 Plot gestures

- First of all plot the unmodified gestures data by concatenation of the acceleration features of all axis

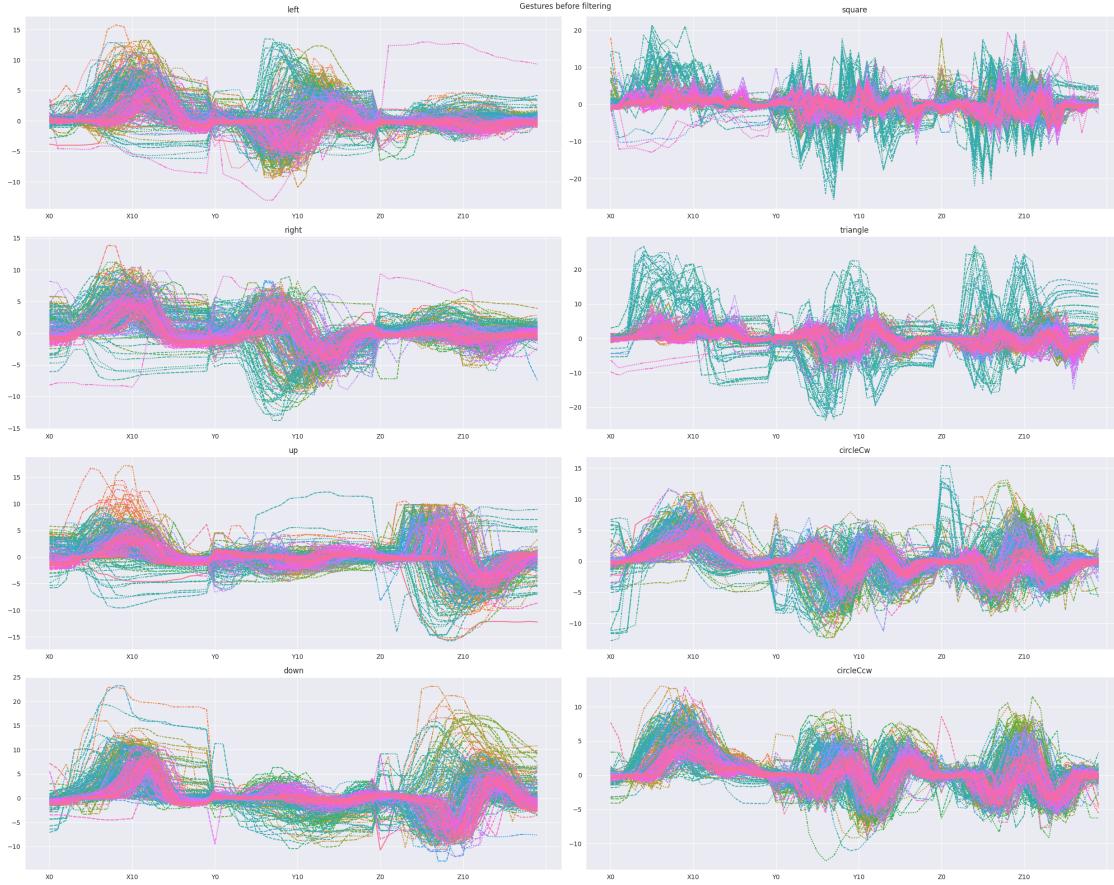
```
[113]: import seaborn as sns
from matplotlib import pyplot as plt

def plot_gestures(df, title):
    gestures = df["gesture"].unique()
    fig, ax = plt.subplots(4, 2, figsize=(25, 20))

    for i, gesture in enumerate(gestures):
        cur_ax = ax[i % 4][i // 4]
        cur_ax.set_title(gesture)
        cur_ax.set_xticks(range(0, 61, 10))
        sns.lineplot(df[df["gesture"] == gesture][df.columns.values[1:]].
                     transpose(), legend=False, ax=cur_ax)

    fig.suptitle(title)
    fig.tight_layout()
    plt.show()

plot_gestures(df, "Gestures before filtering")
```



## 1.2.2 Filtering

We tried different approaches to filter the data, like `scipy medfilt` (median filter) and `wiener` filter. In the end we decided to stick with our own `normalize` filter as we thought it preserved the enough details, i.e. the outputs of the different gestures, look different enough

```
[114]: from scipy.signal import medfilt, wiener

def normalize(x):
    return (x - x.mean()) / x.std()

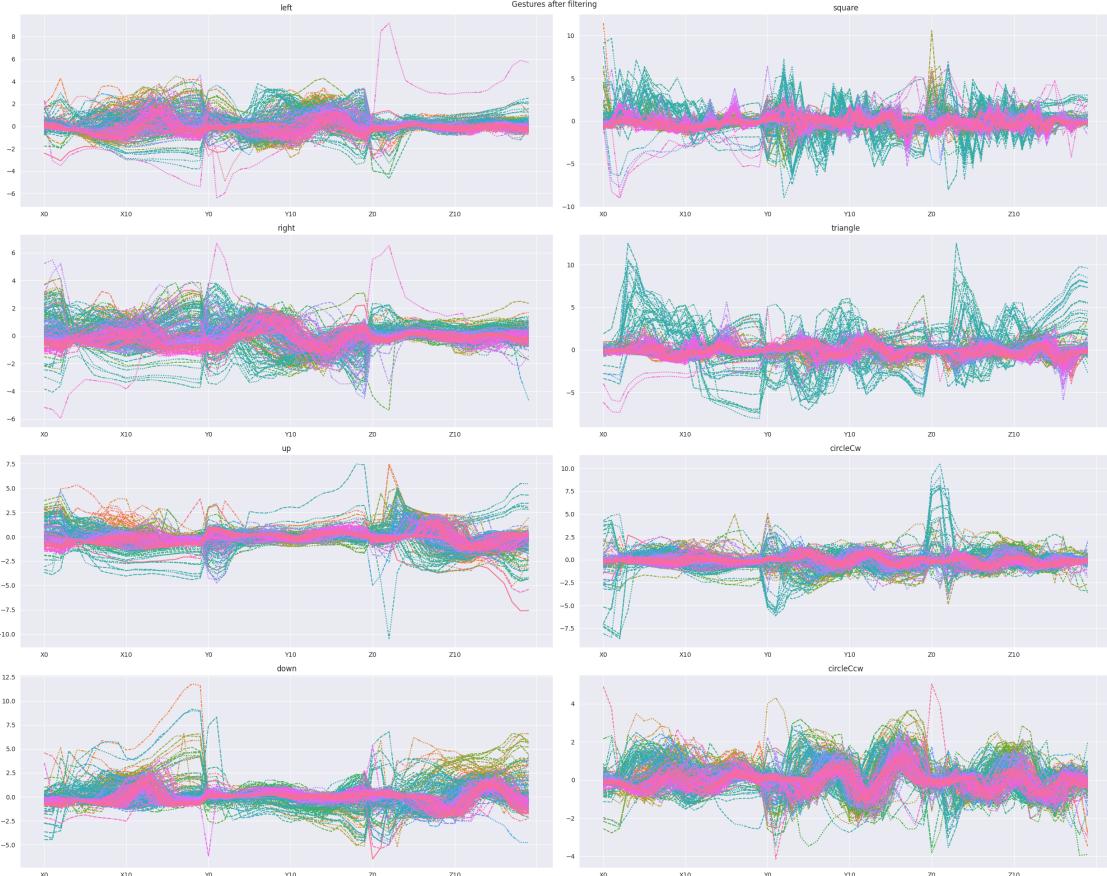
filtered_df = df[df.columns.values[1:]].apply(normalize)
# medfilt
# filtered_df = filtered_df[filtered_df.columns.values[1:]].apply(func=lambda x:
#     medfilt(x, kernel_size=9))
# wiener filter
# filtered_df = df[df.columns.values[1:]].apply(func=lambda x: wiener(x,
#     mysize=9))
```

```

filtered_df = pd.concat([df[["gesture"]], filtered_df], axis=1) # re-add the
    ↪gestures

plot_gestures(filtered_df, "Gestures after filtering")

```



### 1.2.3 Add length per sample

We thought it would help to add the length of each sample (number of values per sample) as a feature. For this we counted how many not null column were in each sample. Since the raw files should all have the same amount, we decided to use the information from the x-axis.

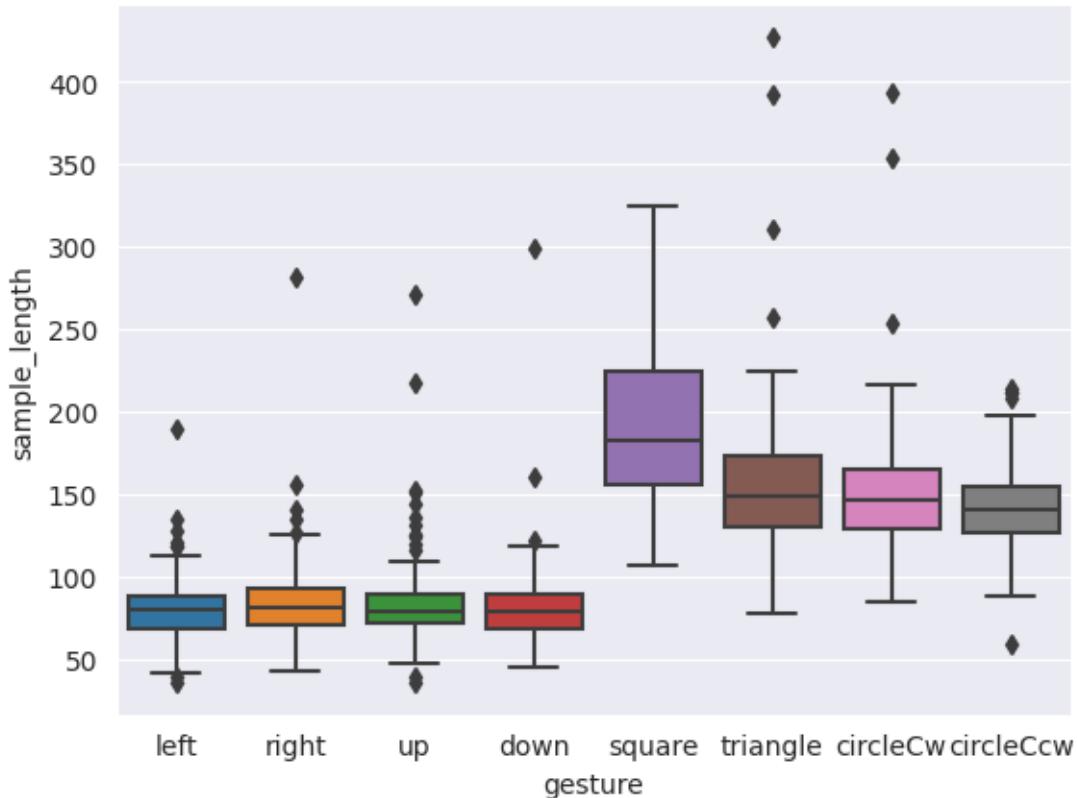
Since all will have at least 3 not-null columns we did not bother to exclude `gesture`, `person` and `sample` from the count.

```
[115]: wear_x = pd.read_csv("raw/wear/raw_data_wear_x.csv", header=None)
filtered_df[["sample_length"]] = wear_x.count(axis='columns')
```

**Boxplot** By plotting the `sample_length`, it is clear that it should at least be possible to separate long gestures from short gestures. We can also see that there are some outliers.

```
[116]: sns.boxplot(filtered_df, x="gesture", y="sample_length")
```

```
[116]: <AxesSubplot: xlabel='gesture', ylabel='sample_length'>
```



#### 1.2.4 Rename and separate target

Before scaling the data, use common X and y notation for the data and target dataframes.

```
[ ]: y = df["gesture"]
filtered_df.drop(columns=["gesture"], inplace=True)
X = filtered_df
# X.describe()
```

```
-----  
KeyError                                  Traceback (most recent call last)  
Cell In [125], line 2  
      1 y = df["gesture"]  
----> 2 filtered_df.drop(columns=["gesture"], inplace=True)  
      3 X = filtered_df
```

```

File ~/fh/ml/notebooks/venv/lib/python3.10/site-packages/pandas/util/_decorators.py:331, in deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*args, **kwargs)
      325 if len(args) > num_allow_args:
      326     warnings.warn(
      327         msg.format(arguments=_format_argument_list(allow_args)),
      328         FutureWarning,
      329         stacklevel=find_stack_level(),
      330     )
--> 331 return func(*args, **kwargs)

File ~/fh/ml/notebooks/venv/lib/python3.10/site-packages/pandas/core/frame.py:5388, in DataFrame.drop(self, labels, axis, index, columns, level, inplace, errors)
      5240 @deprecate_nonkeyword_arguments(version=None, allowed_args=["self", "labels"])
      5241 def drop(  # type: ignore[override]
      5242     self,
      5243     ...
      5249     errors: IgnoreRaise = "raise",
      5250 ) -> DataFrame | None:
      5251     """
      5252     Drop specified labels from rows or columns.
      5253
      5254     ...
      5255
      5256     weight    1.0      0.8
      5257     """
-> 5258     return super().drop(
      5259         labels=labels,
      5260         axis=axis,
      5261         index=index,
      5262         columns=columns,
      5263         level=level,
      5264         inplace=inplace,
      5265         errors=errors,
      5266     )
      5267

File ~/fh/ml/notebooks/venv/lib/python3.10/site-packages/pandas/util/_decorators.py:331, in deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*args, **kwargs)
      325 if len(args) > num_allow_args:
      326     warnings.warn(
      327         msg.format(arguments=_format_argument_list(allow_args)),
      328         FutureWarning,
      329         stacklevel=find_stack_level(),
      330     )
--> 331 return func(*args, **kwargs)

```

```

File ~/fh/ml/notebooks/venv/lib/python3.10/site-packages/pandas/core/generic.py
  ↪4505, in NDFrame.drop(self, labels, axis, index, columns, level, inplace, errors)
  ↪4506
4503 for axis, labels in axes.items():
4504     if labels is not None:
-> 4505         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
4507 if inplace:
4508     self._update_inplace(obj)

File ~/fh/ml/notebooks/venv/lib/python3.10/site-packages/pandas/core/generic.py
  ↪4546, in NDFrame._drop_axis(self, labels, axis, level, errors, only_slice)
4544     new_axis = axis.drop(labels, level=level, errors=errors)
4545 else:
-> 4546     new_axis = axis.drop(labels, errors=errors)
4547 indexer = axis.get_indexer(new_axis)
4549 # Case for non-unique axis
4550 else:

File ~/fh/ml/notebooks/venv/lib/python3.10/site-packages/pandas/core/indexes/
  ↪base.py:6975, in Index.drop(self, labels, errors)
6973 if mask.any():
6974     if errors != "ignore":
-> 6975         raise KeyError(f"list({labels[mask]}) not found in axis")
6976 indexer = indexer[~mask]
6977 return self.delete(indexer)

KeyError: "['gesture'] not found in axis"

```

## 1.2.5 Outlier removal

Before scaling the data, clip the most extreme outliers.

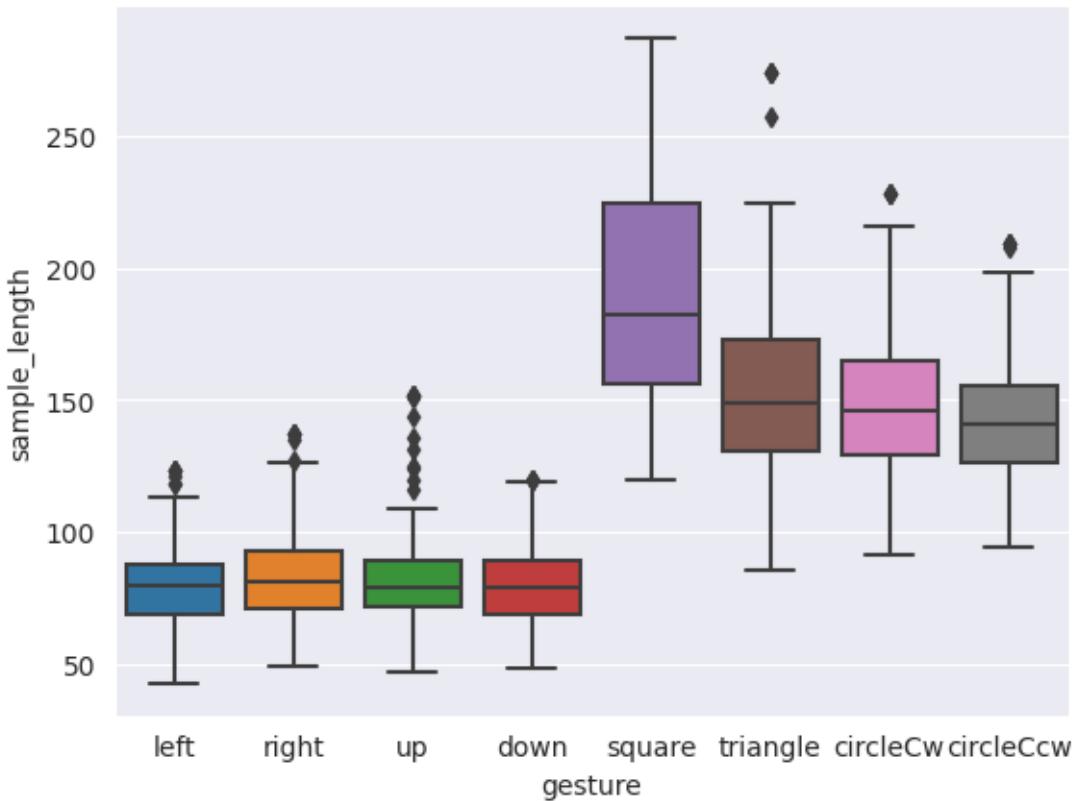
For that we need to make sure, the clipping only happens based on the gesture (group it by gesture)

```
[118]: # Old way without proper clipping
# q_max = X.quantile(.99)
# q_min = X.quantile(.01)
# X.clip(lower=q_min, upper=q_max, axis=1, inplace=True)
# sns.boxplot(pd.concat([y, X], axis=1), x="gesture", y="sample_length")

grouped_by_gesture = X.groupby(df["gesture"], as_index=False)
for name, group in grouped_by_gesture:
    q_max = group.quantile(.99)
    q_min = group.quantile(.01)
    mask = (df["gesture"] == name)
    X.loc[mask, group.columns] = group.clip(lower=q_min, upper=q_max, axis=1)

sns.boxplot(pd.concat([y, X], axis=1), x="gesture", y="sample_length")
```

```
[118]: <AxesSubplot: xlabel='gesture', ylabel='sample_length'>
```



## 1.2.6 Scaling

Scale the data by using the `StandardScaler` of `sklearn`

```
[119]: from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(X, y)

X_scaled = scaler.transform(X)

X_scaled = pd.DataFrame(X_scaled, columns=X.columns)
```

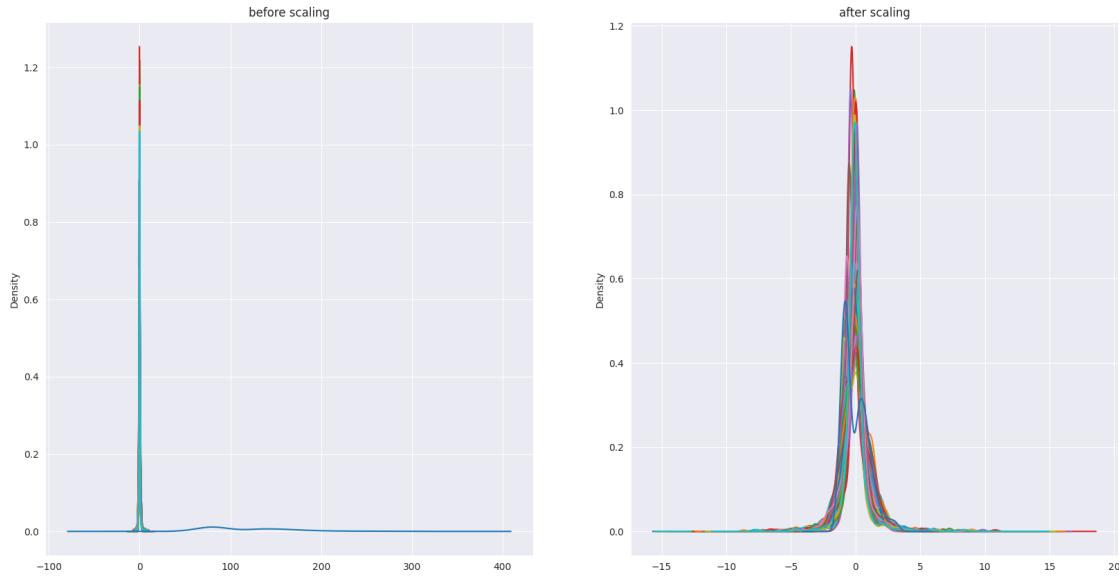
### Density plots before/after

```
[120]: fig, (ax1, ax2) = plt.subplots(ncols=2)
ax1.set_title("before scaling")

fig.set_size_inches(20, 10)
X.plot.density(ax=ax1, legend=False)
```

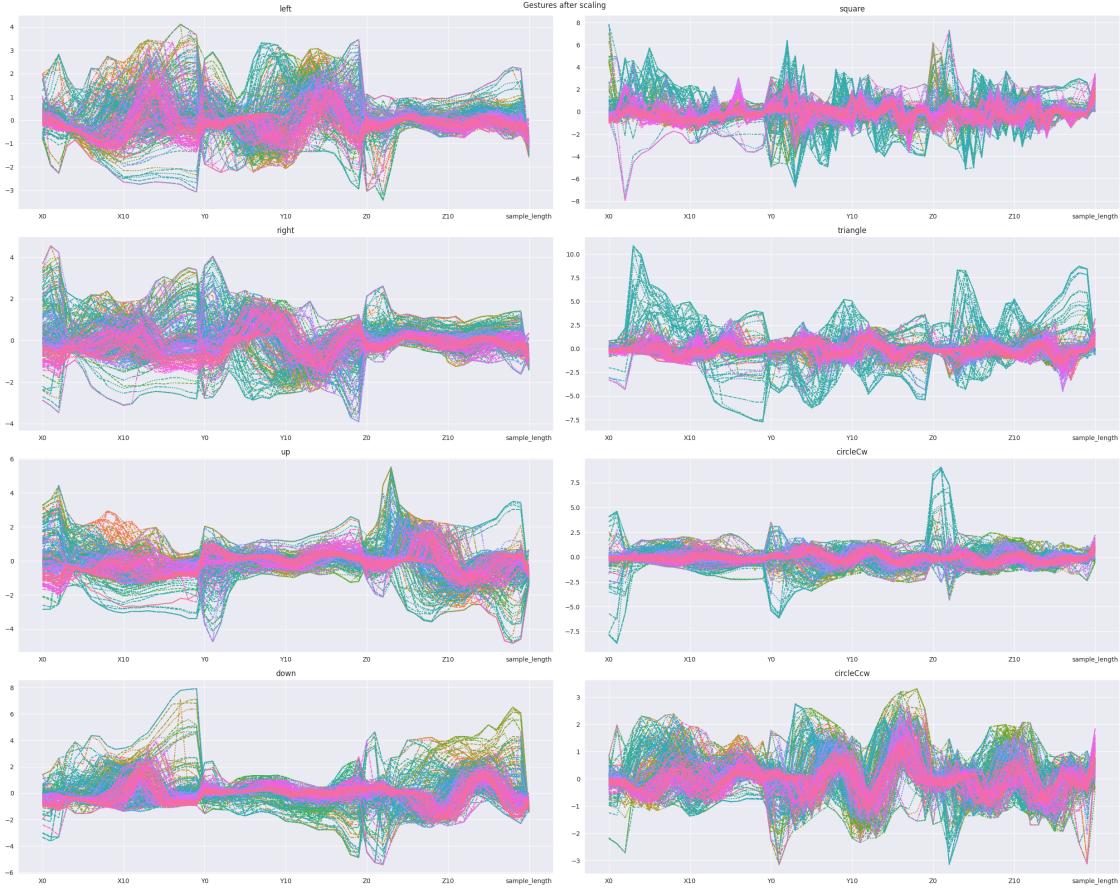
```
ax2.set_title("after scaling")
X_scaled.plot.density(ax=ax2, legend=False)
```

[120]: <AxesSubplot: title={'center': 'after scaling'}, ylabel='Density'>



### Gesture plots after scaling

[121]: `plot_gestures(pd.concat([y, X_scaled], axis=1), "Gestures after scaling")`



### 1.2.7 Adding arbitrary features

**Temporal domain features** We thought that maybe we could gain additional knowledge by adding the derivatives of the preprocessed data as features.

```
[122]: import numpy as np

x_columns = ["X" + str(i) for i in range(0, 20)]
y_columns = ["Y" + str(i) for i in range(0, 20)]
z_columns = ["Z" + str(i) for i in range(0, 20)]
# Calculate the first derivative (acceleration) of the acceleration data
der_1_x = np.diff(X_scaled[x_columns].values, axis=1)
der_1_y = np.diff(X_scaled[y_columns].values, axis=1)
der_1_z = np.diff(X_scaled[z_columns].values, axis=1)

# Calculate the second derivative (jerk) of the acceleration data
der_2_x = np.diff(der_1_x, axis=1)
der_2_y = np.diff(der_1_y, axis=1)
der_2_z = np.diff(der_1_z, axis=1)
```

```

# Add the acceleration and jerk data as new features to the DataFrame
temporal_domain_features = pd.concat([
    pd.DataFrame(der_1_x, columns=["x'" + str(i) for i in range(0, 19)]),
    pd.DataFrame(der_1_y, columns=["y'" + str(i) for i in range(0, 19)]),
    pd.DataFrame(der_1_z, columns=["z'" + str(i) for i in range(0, 19)]),
    pd.DataFrame(der_2_x, columns=["x''" + str(i) for i in range(0, 18)]),
    pd.DataFrame(der_2_y, columns=["y''" + str(i) for i in range(0, 18)]),
    pd.DataFrame(der_2_z, columns=["z''" + str(i) for i in range(0, 18)]),
], axis=1)

```

**Frequency domain features:** Next we tried to gain more knowledge by including features of the frequency domain such as information from the fft.

```

[123]: from scipy.fft import dct

# Calculate the fast Fourier transform (FFT) of the acceleration data
fft_x = np.fft.fft(X_scaled[x_columns].values, axis=1)
fft_y = np.fft.fft(X_scaled[y_columns].values, axis=1)
fft_z = np.fft.fft(X_scaled[z_columns].values, axis=1)

# Calculate the power spectral density (PSD) of the acceleration data
psd_x = np.abs(fft_x) ** 2
psd_y = np.abs(fft_y) ** 2
psd_z = np.abs(fft_z) ** 2

# Calculate the dominant frequency of the acceleration data
dominant_freq_x = np.argmax(psd_x, axis=1) / len(X_scaled)
dominant_freq_y = np.argmax(psd_y, axis=1) / len(X_scaled)
dominant_freq_z = np.argmax(psd_z, axis=1) / len(X_scaled)

frequency_domain_features = pd.concat([
    pd.DataFrame(psd_x, columns=["psd_x'" + str(i) for i in range(0, 20)]),
    pd.DataFrame(psd_y, columns=["psd_y'" + str(i) for i in range(0, 20)]),
    pd.DataFrame(psd_z, columns=["psd_z'" + str(i) for i in range(0, 20)]),
    pd.DataFrame(dominant_freq_x, columns=["dom_f_x"]),
    pd.DataFrame(dominant_freq_y, columns=["dom_f_y"]),
    pd.DataFrame(dominant_freq_z, columns=["dom_f_z"]),
], axis=1)

```

Through testing the models with various feature reduction methods, we came to the conclusion, that our additional features do not add any real value. Adding the `temporal_domain_features` makes the models perform way worse avg scores from 0.5 - 0.65. Adding the `frequency_domain_features` has the same effect, just not quite as harsh, as the avg score stick around 0.65 - 0.7.

So we concluded that we should not add these features, as they just harm the model performance.

```
[124]: X_scaled = pd.concat([
    X_scaled,
    # temporal_domain_features, # uncomment to see effect
    # frequency_domain_features # uncomment to see effect
], axis=1)
```

### 1.2.8 Create train test split (held-back test set)

**Gallery dependent** First we trained our model using gallery dependent cross validation. The held back test set was created using the `StratifiedShuffleSplit` of sklearn. Later on we decided to do gallery independent cross validation instead, so we commented out this code, but left it in for documentation.

```
[76]: from sklearn.model_selection import StratifiedShuffleSplit

# Gallery dependent held-back test set
# to change implementation uncomment this for stratified_split
# comment out the gallery independent code below, and remove the cv param from
# knn, tree, svm and knn GridSearchCVs
# stratified_split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
# X_train, X_test, y_train, y_test = [], [], [], []
# for train_index, test_index in stratified_split.split(X_scaled, persons["person"]):
#     X_train = X_scaled.loc[train_index]
#     X_test = X_scaled.loc[test_index]
#     y_train = y.loc[train_index]
#     y_test = y.loc[test_index]
```

**Gallery independent** To achieve gallery independent cross validation, we used the `LeavePGroupsOut` cross validator and leaving out 3 groups i.e. persons in our case. Here we used it to create a gallery independent held-back test set.

```
[77]: from sklearn.model_selection import LeavePGroupsOut

# Gallery independent held-back test set
# comment this out if you want to run stratified_split instead
lpo = LeavePGroupsOut(n_groups=3)
train_index, test_index = lpo.split(X_scaled, groups=persons["person"]).
    __next__()
X_train = X_scaled.loc[train_index]
X_test = X_scaled.loc[test_index]
y_train = y.loc[train_index]
y_test = y.loc[test_index]
persons = persons.loc[train_index]
```

### 1.2.9 Feature reduction

We decided to try different feature selection strategies and compare their results. Though this increased the training time massively. We decided to use kBest, RFE and PCA feature selection and reducing to 20 features.

#### kBest

```
[78]: from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
from sklearn.preprocessing import LabelEncoder

# Create an instance of SelectKBest
selector = SelectKBest(f_classif, k=20)
label_encoder = LabelEncoder()
label_encoder.fit(y)
train = label_encoder.transform(y_train)
# Fit the selector to the data
selector.fit(X_train, train)

# Get the selected features
kbest_X_train = selector.transform(X_train)
kbest_X_test = selector.transform(X_test)
```

#### RFE

```
[79]: from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

# Create an instance of LinearRegression
lr = LinearRegression()
label_encoder = LabelEncoder()
label_encoder.fit(y)
train = label_encoder.transform(y_train)
# Create an instance of RFE
rfe = RFE(lr, n_features_to_select=20)

# Fit the RFE to the data
rfe.fit(X_train, train)

# Get the selected features
rfe_X_train = rfe.transform(X_train)
rfe_X_test = rfe.transform(X_test)
```

#### PCA

```
[80]: from sklearn.decomposition import PCA
```

```

pca = PCA(n_components=20) # if features are added, change n_components >= 20
    ↵to achieve similar performance
pca.fit(X_train)
pca_X_train = pca.transform(X_train)
pca_X_test = pca.transform(X_test)

```

### 1.3 Model Training

We decided to use 4 different models: \* knn \* tree \* svm \* nn

Since we have 3 different feature selection strategies, we end up with 12 models.

Training is done via `GridSearchCV` to find the best parameters. To achieve gallery independent training we used the `LeavePGroupsOut` cross validator leaving out 3 groups (persons) for each cv run.

To cut down on processing time, we saved the best parameters to the `best_params` dictionary so this notebook can be run in a reasonable time. For this we defined 2 methods for each model type, one to give us the model by using grid search and one that simply fits the model using the best params.

#### KNN methods

```
[86]: from sklearn.model_selection import GridSearchCV
from sklearn import neighbors, neural_network

def knn_grid_search(X_train, y_train):
    knn = GridSearchCV(
        estimator=neighbors.KNeighborsClassifier(),
        param_grid=[{'n_neighbors': [3, 5, 7, 9], 'weights': ['uniform', ↵
        'distance'], 'leaf_size': [10, 15, 20]}],
        scoring="accuracy",
        cv=lpo.split(X_train, groups=persons["person"]),
        return_train_score=True
    )
    knn.fit(X_train, y_train)
    print(knn.best_params_)
    return knn, knn.best_params_

def knn_with_best_params(X_train, y_train, params):
    knn = neighbors.KNeighborsClassifier(n_neighbors=params["n_neighbors"], ↵
    weights=params["weights"],
                                         leaf_size=params["leaf_size"])
    knn.fit(X_train, y_train)
    return knn, params
```

#### Tree methods

```
[87]: from sklearn.tree import DecisionTreeClassifier

def tree_grid_search(X_train, y_train):
    tree = GridSearchCV(
        estimator=DecisionTreeClassifier(),
        param_grid=[{
            'splitter': ['best', 'random'],
            'max_depth': [10, 100, 1000],
            'criterion': ['gini', 'entropy', 'log_loss'],
            'class_weight': ['balanced']}],
        scoring="accuracy",
        cv=lpo.split(X_train, groups=persons["person"]),
        n_jobs=-1
    )

    tree.fit(X_train, y_train)
    print(tree.best_params_)
    return tree, tree.best_params_

def tree_with_best_params(X_train, y_train, params):
    tree = DecisionTreeClassifier(splitter=params["splitter"], max_depth=params["max_depth"],
                                  criterion=params["criterion"], class_weight=params["class_weight"])
    tree.fit(X_train, y_train)
    return tree, params
```

## SVM methods

```
[88]: from sklearn.utils.fixes import loguniform
from sklearn import svm as subvector

def svm_grid_search(X_train, y_train):
    svc = GridSearchCV(
        estimator=subvector.SVC(),
        param_grid=[{
            'C': loguniform(0.1, 1, 100, 1000).rvs(20),
            'class_weight': ['balanced'],
            'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
            'gamma': loguniform(0.000035, 0.000245).rvs(20)
        }],
        cv=lpo.split(X_train, groups=persons["person"]),
        n_jobs=-1
    )
```

```

    svc.fit(X_train, y_train)
    print(svc.best_params_)
    return svc, svc.best_params_

def svm_with_best_params(X_train, y_train, params):
    svc = subvector.SVC(C=params["C"], class_weight=params["class_weight"], ↴
    ↴kernel=params["kernel"],
                           gamma=params["gamma"])
    svc.fit(X_train, y_train)
    return svc, params

```

### Neural Network methods

```
[89]: def nn_grid_search(X_train, y_train):
    nn = GridSearchCV(
        estimator=neural_network.MLPClassifier(max_iter=10000),
        param_grid=[{
            'hidden_layer_sizes': [6, 9, 12],
            'activation': ['identity', 'logistic', 'relu'],
            'solver': ['lbfgs', 'sgd'],
            'learning_rate': ['constant', 'adaptive']
        }],
        cv=lpo.split(X_train, groups=persons["person"]),
        n_jobs=-1
    )
    nn.fit(X_train, y_train)
    print(nn.best_params_)
    return nn, nn.best_params_

def nn_with_best_params(X_train, y_train, params):
    nn = neural_network.MLPClassifier(max_iter=10000, ↴
    ↴hidden_layer_sizes=params["hidden_layer_sizes"], ↴
                           activation=params["activation"], ↴
    ↴solver=params["solver"], ↴
                           learning_rate=params["learning_rate"])
    nn.fit(X_train, y_train)
    return nn, params

```

#### 1.3.1 Training models

Here we train all 12 models. The `best_params` dictionary contains the best params of the run used to complete the exercise. This is done, so that the notebook can be executed faster. The gridsearch initially took around 4h per strategy, so 12h in total. This was reduced to around 1h on a 8 core 16 thread cpu by setting the `n_job` param of the `GridSearchCV` to `-1`, so the grid search uses all the available cores.

```
[108]: models = {}
# save best_params from GridSearchCV
# to recalc best params remove / comment out the initialization of best_params
# but note that each loop takes around 4 hours to complete so about 12h in total
best_params = {
    # stratified kFold
    # "kBest": {
        # "knn": {'leaf_size': 10, 'n_neighbors': 3, 'weights': 'distance'},
        # "tree": {'class_weight': 'balanced', 'criterion': 'entropy', ↴
        'max_depth': 1000, 'splitter': 'best'},
        # "svm": {'C': 696.6186904262564, 'class_weight': 'balanced', 'gamma': ↴
        0.0002171766176033019, 'kernel': 'rbf'},
        # "nn": {'activation': 'relu', 'hidden_layer_sizes': 12, ↴
        'learning_rate': 'adaptive', 'solver': 'sgd'}
    },
    # "rfe": {
        # "knn": {'leaf_size': 10, 'n_neighbors': 3, 'weights': 'distance'},
        # "tree": {'class_weight': 'balanced', 'criterion': 'entropy', ↴
        'max_depth': 1000, 'splitter': 'best'},
        # "svm": {'C': 439.8789649591561, 'class_weight': 'balanced', 'gamma': ↴
        0.00022595356458860057, 'kernel': 'rbf'},
        # "nn": {'activation': 'relu', 'hidden_layer_sizes': 12, ↴
        'learning_rate': 'adaptive', 'solver': 'sgd'},
    },
    # "pca": {
        # "knn": {'leaf_size': 10, 'n_neighbors': 3, 'weights': 'distance'},
        # "tree": {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': ↴
        1000, 'splitter': 'best'},
        # "svm": {'C': 671.3171466128516, 'class_weight': 'balanced', 'gamma': ↴
        0.000219343943890147, 'kernel': 'rbf'},
        # "nn": {'activation': 'relu', 'hidden_layer_sizes': 12, ↴
        'learning_rate': 'constant', 'solver': 'sgd'}
    }
}

# LeavePGroupsOut
'kBest': {
    'knn': {'leaf_size': 10, 'n_neighbors': 3, 'weights': 'distance'},
    'tree': {'class_weight': 'balanced', 'criterion': 'log_loss', ↴
    'max_depth': 100, 'splitter': 'best'},
    'svm': {'C': 239.11122411579947, 'class_weight': 'balanced', 'gamma': 7. ↴
    028366450401378e-05, 'kernel': 'rbf'},
    'nn': {'activation': 'relu', 'hidden_layer_sizes': 12, 'learning_rate': ↴
    'constant', 'solver': 'sgd'}
},
'rfe': {
    'knn': {'leaf_size': 10, 'n_neighbors': 3, 'weights': 'distance'},
```

```

        'tree': {'class_weight': 'balanced', 'criterion': 'log_loss', ↴
↳ 'max_depth': 100, 'splitter': 'best'},
        'svm': {'C': 992.8479466901772, 'class_weight': 'balanced', 'gamma': 0. ↴
↳ 00023077983300935984, 'kernel': 'rbf'},
        'nn': {'activation': 'relu', 'hidden_layer_sizes': 12, 'learning_rate': ↴
↳ 'adaptive', 'solver': 'sgd'}
    },
    'pca': {
        'knn': {'leaf_size': 10, 'n_neighbors': 3, 'weights': 'distance'},
        'tree': {'class_weight': 'balanced', 'criterion': 'log_loss', ↴
↳ 'max_depth': 1000, 'splitter': 'best'},
        'svm': {'C': 451.32633762305517, 'class_weight': 'balanced', 'gamma': 0. ↴
↳ 0002120012988488178, 'kernel': 'rbf'},
        'nn': {'activation': 'relu', 'hidden_layer_sizes': 12, 'learning_rate': ↴
↳ 'adaptive', 'solver': 'sgd'}
    }
}

for m, X_train, X_test in [("kBest", kbest_X_train, kbest_X_test),
                           ("rfe", rfe_X_train, rfe_X_test),
                           ("pca", pca_X_train, pca_X_test)]:
    params = best_params.get(m)
    if params is None:
        knn = knn_grid_search(X_train, y_train)
        tree = tree_grid_search(X_train, y_train)
        svm = svm_grid_search(X_train, y_train)
        nn = nn_grid_search(X_train, y_train)
    else:
        knn = knn_with_best_params(X_train, y_train, params.get("knn"))
        tree = tree_with_best_params(X_train, y_train, params.get("tree"))
        svm = svm_with_best_params(X_train, y_train, params.get("svm"))
        nn = nn_with_best_params(X_train, y_train, params.get("nn"))
    models[m] = (X_test, {"knn": knn[0], "tree": tree[0], "svm": svm[0], "nn": ↴
↳ nn[0]})
    best_params[m] = {"knn": knn[1], "tree": tree[1], "svm": svm[1], "nn": ↴
↳ nn[1]}
    print(m, "done")

```

kBest done  
rfe done  
pca done

[109]: # uncomment to print and copy output into best\_params dictionary for future use  
# best\_params

## 1.4 Performance evaluation

Here we print the classification\_reports and confusion matrices of all models, to decide on which model we want to use.

```
[91]: from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

gestures = df["gesture"].unique()

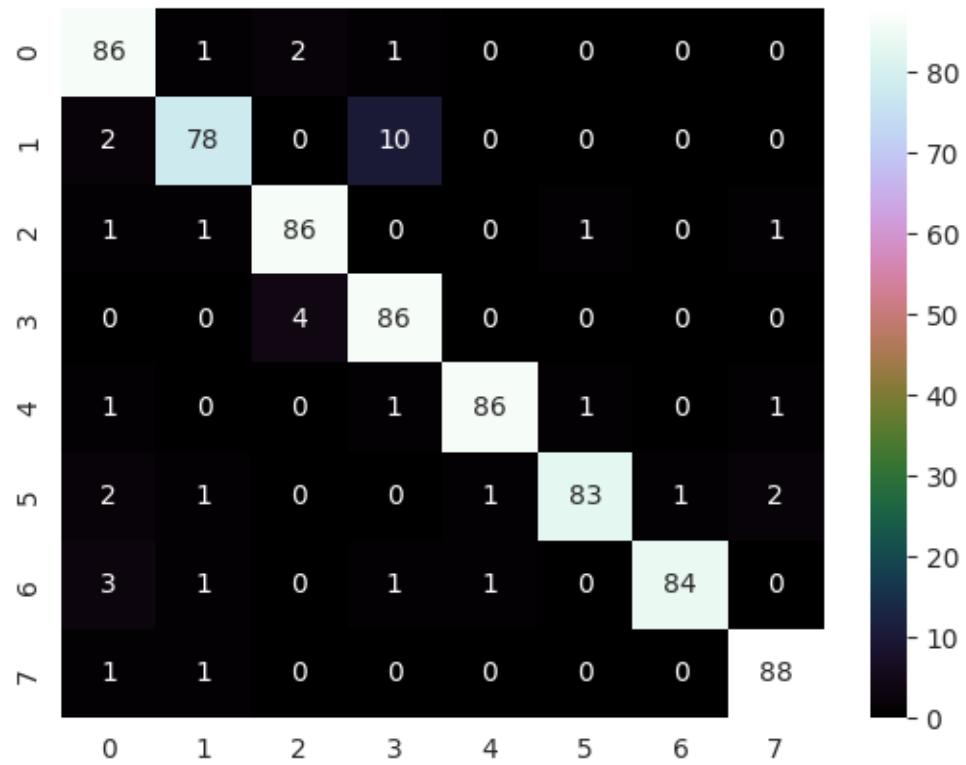
def print_classification_reports(model_name, model, X_test, y_test):
    print(model_name)
    y_pred = model.predict(X_test)
    print(classification_report(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred, labels=gestures)
    sns.heatmap(cm, annot=True, cmap="cubehelix")
    plt.show()

for key, val in models.items():
    print(key + " feature selection")
    X_test = val[0]
    print_classification_reports("KNN", val[1]["knn"], X_test, y_test)
    print_classification_reports("Tree", val[1]["tree"], X_test, y_test)
    print_classification_reports("SVM", val[1]["svm"], X_test, y_test)
    print_classification_reports("NN", val[1]["nn"], X_test, y_test)
```

kBest feature selection

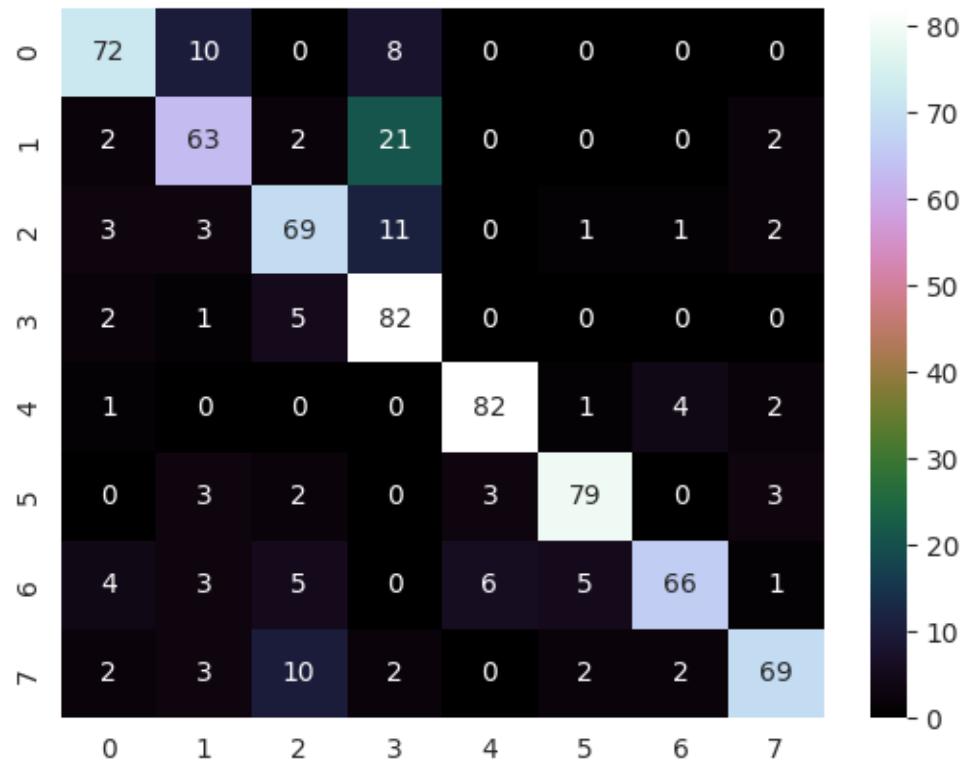
KNN

	precision	recall	f1-score	support
circleCcw	0.96	0.98	0.97	90
circleCw	0.99	0.93	0.96	90
down	0.87	0.96	0.91	90
left	0.90	0.96	0.92	90
right	0.94	0.87	0.90	90
square	0.98	0.96	0.97	90
triangle	0.98	0.92	0.95	90
up	0.93	0.96	0.95	90
accuracy			0.94	720
macro avg	0.94	0.94	0.94	720
weighted avg	0.94	0.94	0.94	720



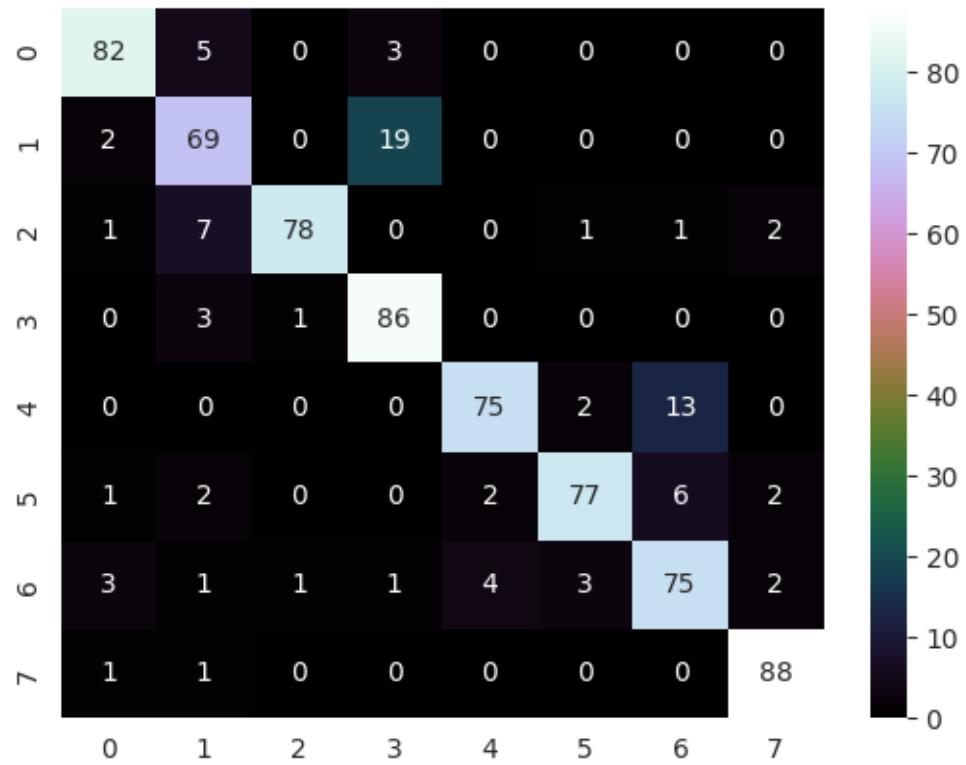
### Tree

	precision	recall	f1-score	support
circleCcw	0.87	0.77	0.82	90
circleCw	0.90	0.73	0.81	90
down	0.66	0.91	0.77	90
left	0.84	0.80	0.82	90
right	0.73	0.70	0.72	90
square	0.90	0.91	0.91	90
triangle	0.90	0.88	0.89	90
up	0.74	0.77	0.75	90
accuracy			0.81	720
macro avg	0.82	0.81	0.81	720
weighted avg	0.82	0.81	0.81	720



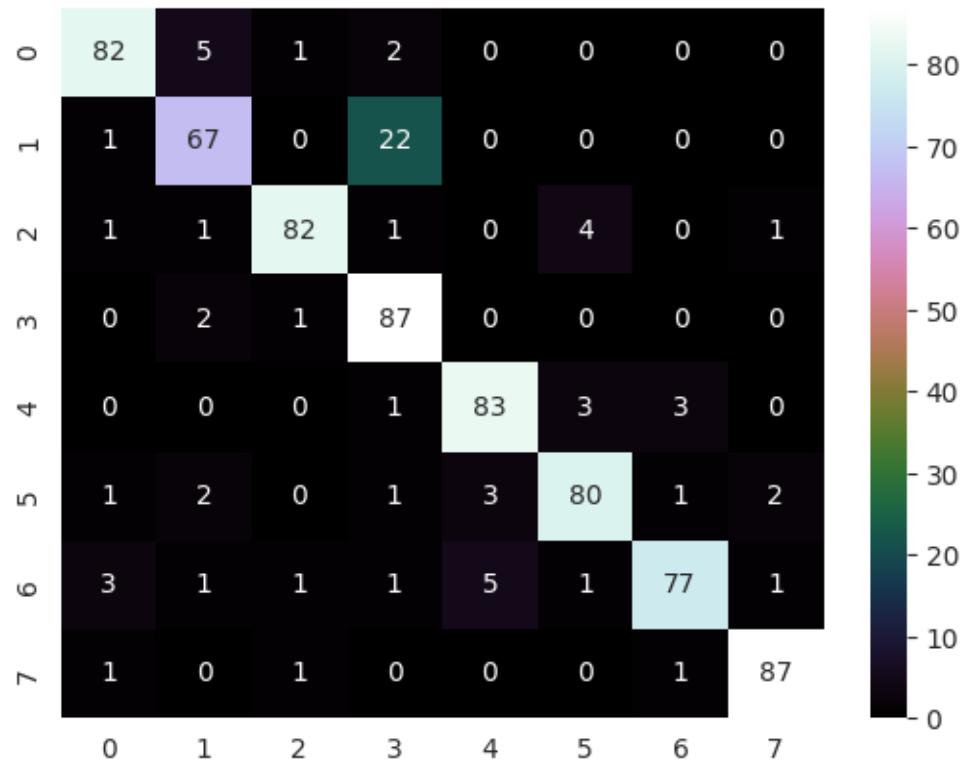
## SVM

	precision	recall	f1-score	support
circleCcw	0.94	0.98	0.96	90
circleCw	0.79	0.83	0.81	90
down	0.79	0.96	0.86	90
left	0.91	0.91	0.91	90
right	0.78	0.77	0.78	90
square	0.93	0.83	0.88	90
triangle	0.93	0.86	0.89	90
up	0.97	0.87	0.92	90
accuracy			0.88	720
macro avg	0.88	0.88	0.88	720
weighted avg	0.88	0.88	0.88	720

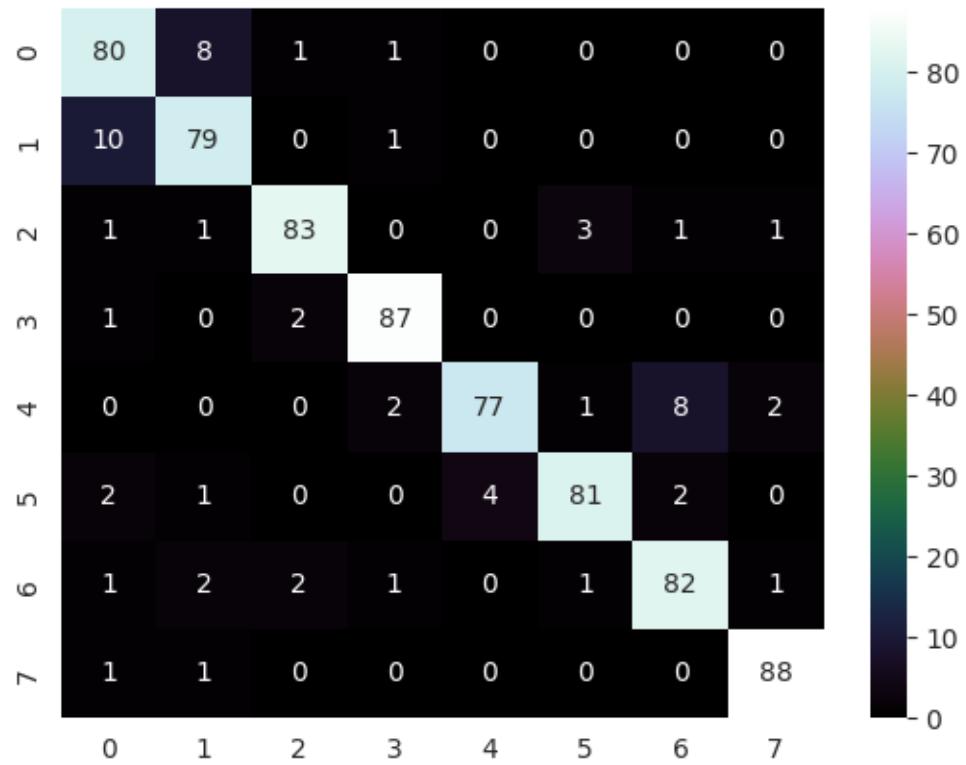


NN

	precision	recall	f1-score	support
circleCcw	0.96	0.97	0.96	90
circleCw	0.94	0.86	0.90	90
down	0.76	0.97	0.85	90
left	0.92	0.91	0.92	90
right	0.86	0.74	0.80	90
square	0.91	0.92	0.92	90
triangle	0.91	0.89	0.90	90
up	0.95	0.91	0.93	90
accuracy			0.90	720
macro avg	0.90	0.90	0.90	720
weighted avg	0.90	0.90	0.90	720

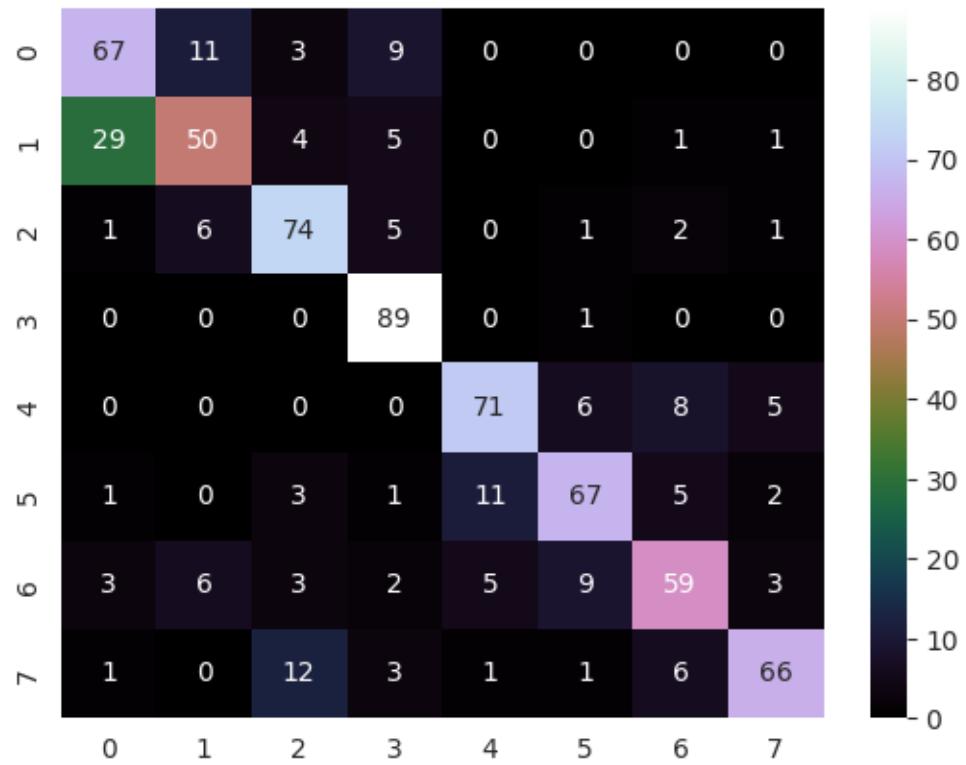


rfe feature selection				
KNN				
	precision	recall	f1-score	support
circleCcw	0.96	0.98	0.97	90
circleCw	0.88	0.91	0.90	90
down	0.95	0.97	0.96	90
left	0.83	0.89	0.86	90
right	0.86	0.88	0.87	90
square	0.95	0.86	0.90	90
triangle	0.94	0.90	0.92	90
up	0.94	0.92	0.93	90
accuracy			0.91	720
macro avg	0.91	0.91	0.91	720
weighted avg	0.91	0.91	0.91	720



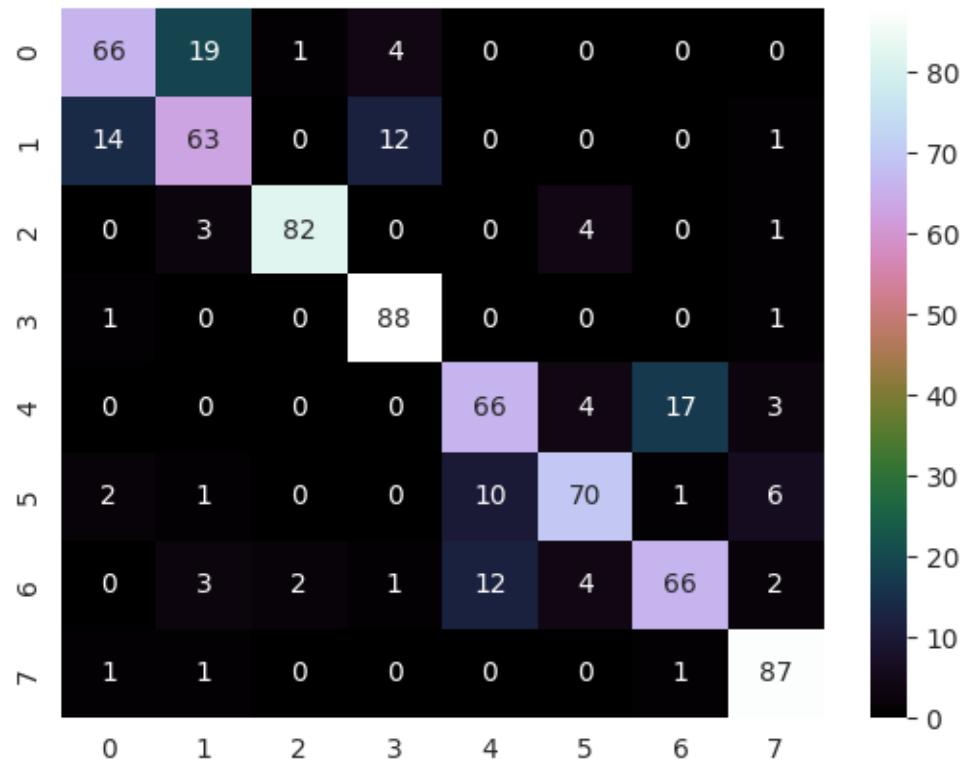
### Tree

	precision	recall	f1-score	support
circleCcw	0.85	0.73	0.79	90
circleCw	0.73	0.66	0.69	90
down	0.78	0.99	0.87	90
left	0.66	0.74	0.70	90
right	0.68	0.56	0.61	90
square	0.81	0.79	0.80	90
triangle	0.79	0.74	0.77	90
up	0.75	0.82	0.78	90
accuracy			0.75	720
macro avg	0.75	0.75	0.75	720
weighted avg	0.75	0.75	0.75	720



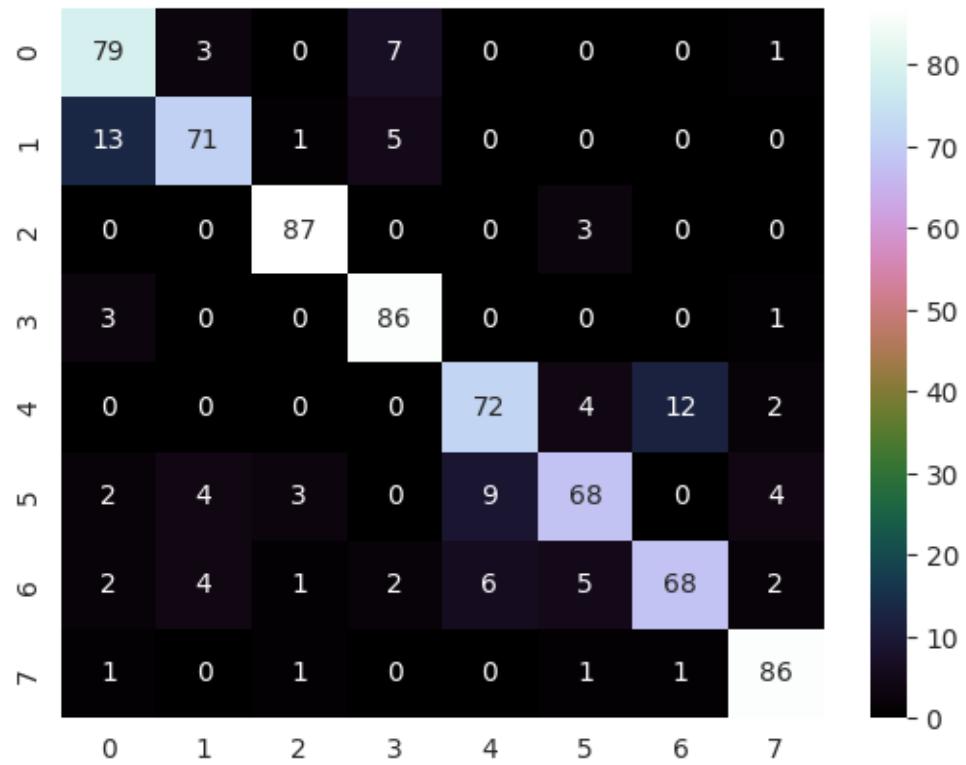
## SVM

	precision	recall	f1-score	support
circleCcw	0.86	0.97	0.91	90
circleCw	0.78	0.73	0.75	90
down	0.84	0.98	0.90	90
left	0.79	0.73	0.76	90
right	0.70	0.70	0.70	90
square	0.75	0.73	0.74	90
triangle	0.85	0.78	0.81	90
up	0.96	0.91	0.94	90
accuracy			0.82	720
macro avg	0.82	0.82	0.81	720
weighted avg	0.82	0.82	0.81	720

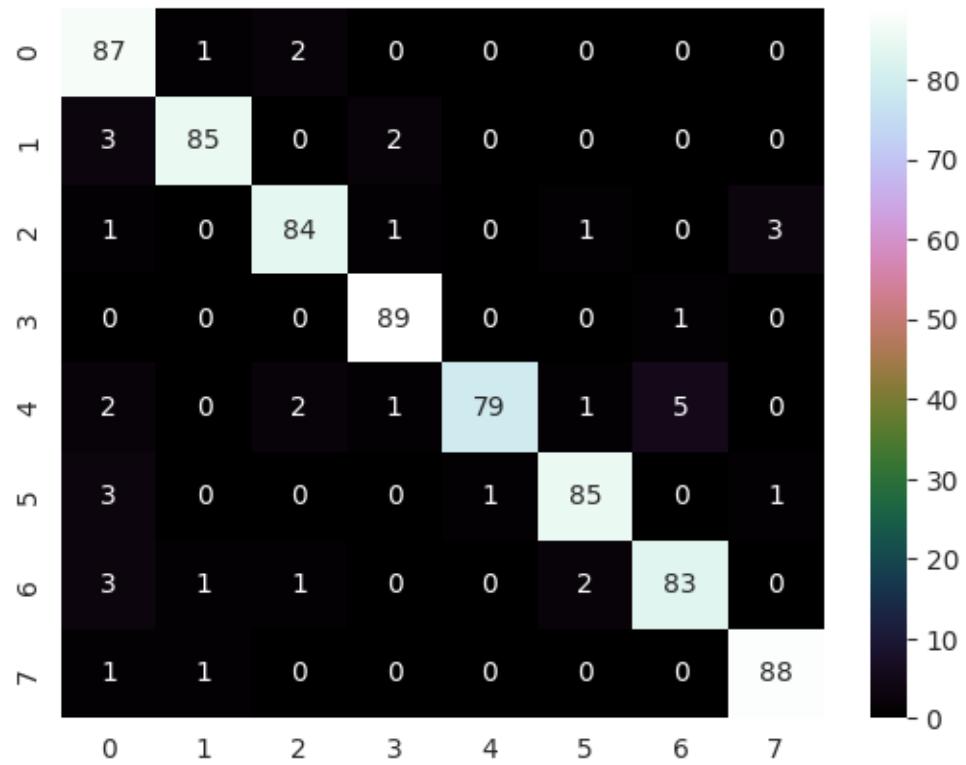


NN

	precision	recall	f1-score	support
circleCcw	0.90	0.96	0.92	90
circleCw	0.84	0.76	0.80	90
down	0.86	0.96	0.91	90
left	0.79	0.88	0.83	90
right	0.87	0.79	0.83	90
square	0.83	0.80	0.81	90
triangle	0.84	0.76	0.80	90
up	0.94	0.97	0.95	90
accuracy			0.86	720
macro avg	0.86	0.86	0.86	720
weighted avg	0.86	0.86	0.86	720

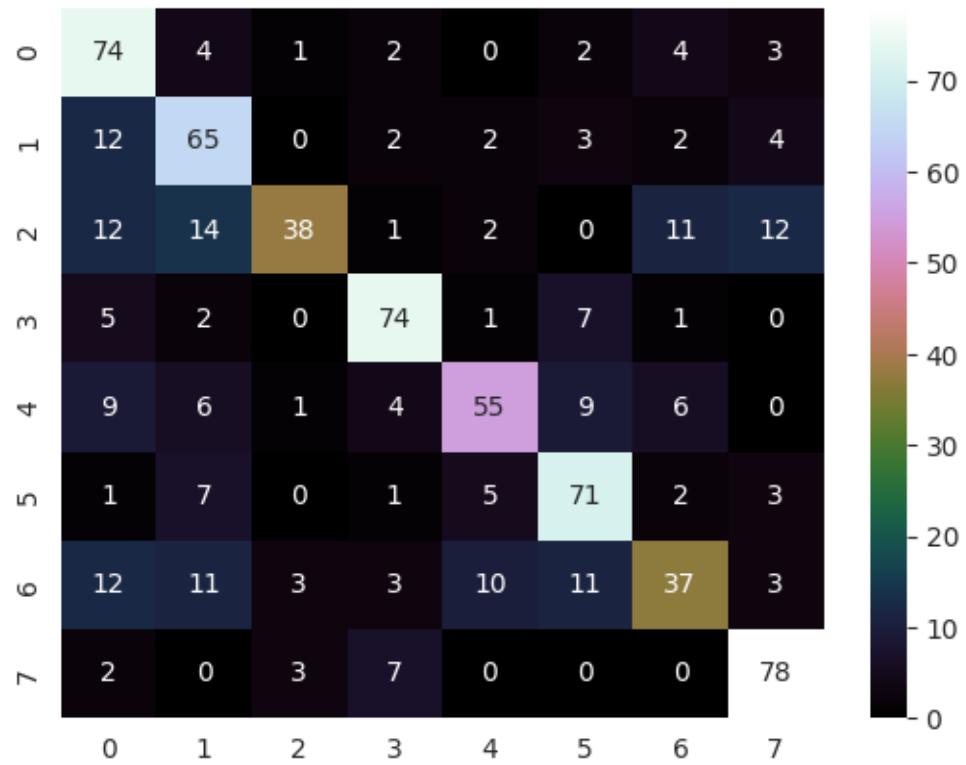


pca feature selection KNN				
	precision	recall	f1-score	support
circleCcw	0.96	0.98	0.97	90
circleCw	0.93	0.92	0.93	90
down	0.96	0.99	0.97	90
left	0.87	0.97	0.92	90
right	0.97	0.94	0.96	90
square	0.99	0.88	0.93	90
triangle	0.96	0.94	0.95	90
up	0.94	0.93	0.94	90
accuracy			0.94	720
macro avg	0.95	0.94	0.94	720
weighted avg	0.95	0.94	0.94	720



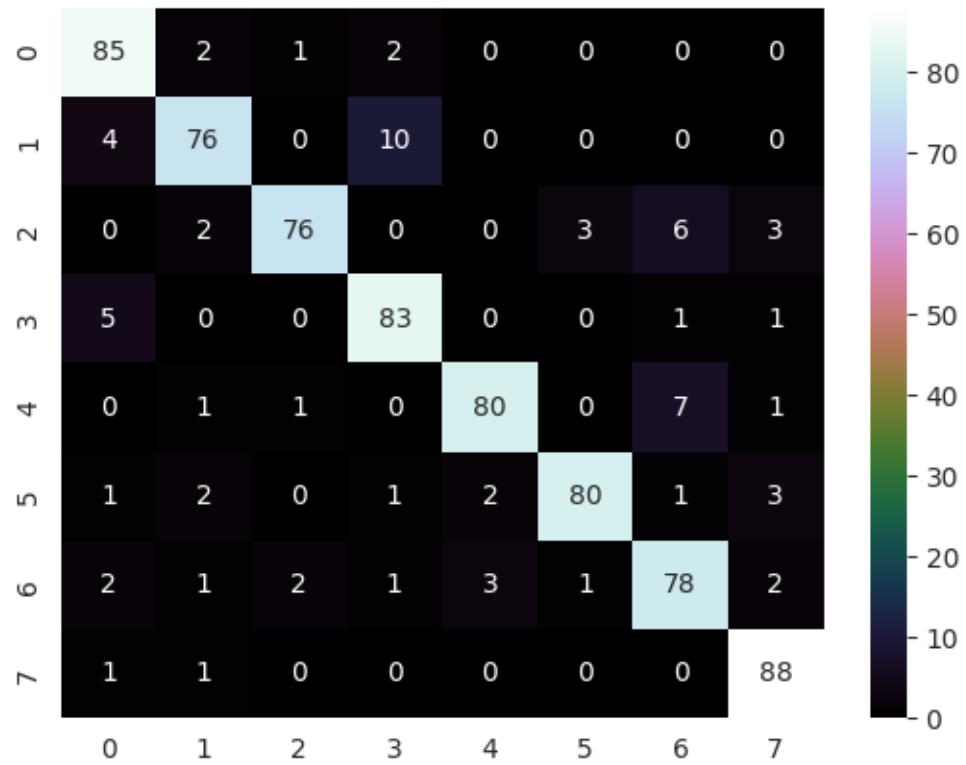
### Tree

	precision	recall	f1-score	support
circleCcw	0.76	0.87	0.81	90
circleCw	0.59	0.41	0.48	90
down	0.79	0.82	0.80	90
left	0.58	0.82	0.68	90
right	0.60	0.72	0.65	90
square	0.73	0.61	0.67	90
triangle	0.69	0.79	0.74	90
up	0.83	0.42	0.56	90
accuracy			0.68	720
macro avg	0.69	0.68	0.67	720
weighted avg	0.69	0.68	0.67	720



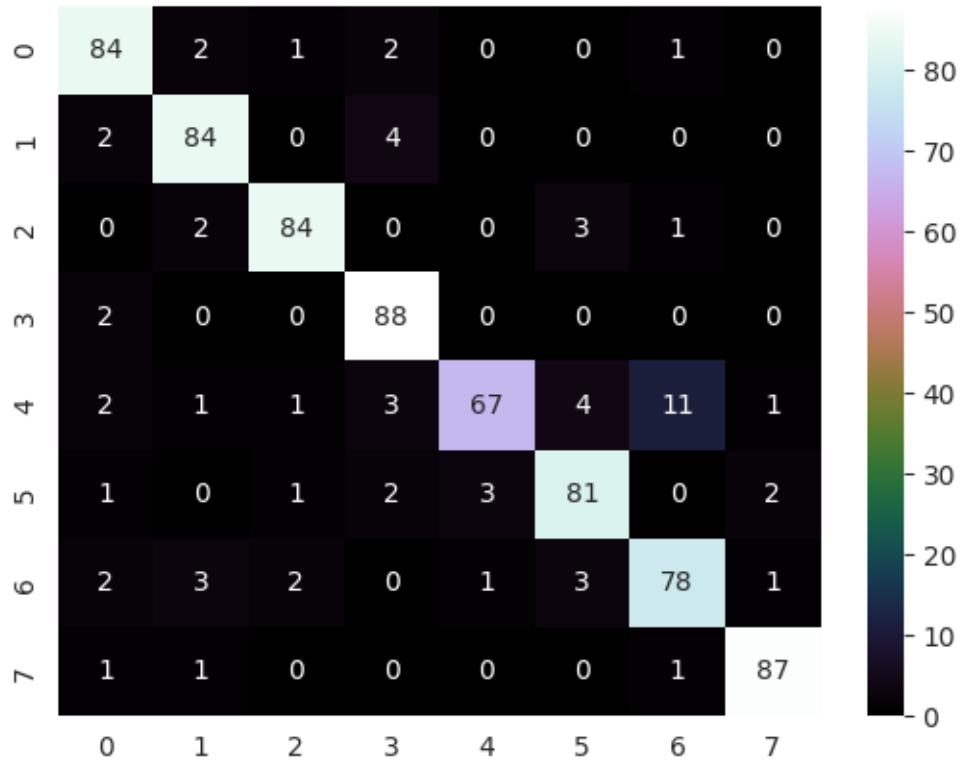
## SVM

	precision	recall	f1-score	support
circleCcw	0.90	0.98	0.94	90
circleCw	0.84	0.87	0.85	90
down	0.86	0.92	0.89	90
left	0.87	0.94	0.90	90
right	0.89	0.84	0.87	90
square	0.94	0.89	0.91	90
triangle	0.95	0.89	0.92	90
up	0.95	0.84	0.89	90
accuracy			0.90	720
macro avg	0.90	0.90	0.90	720
weighted avg	0.90	0.90	0.90	720



NN

	precision	recall	f1-score	support
circleCcw	0.96	0.97	0.96	90
circleCw	0.85	0.87	0.86	90
down	0.89	0.98	0.93	90
left	0.89	0.93	0.91	90
right	0.90	0.93	0.92	90
square	0.94	0.74	0.83	90
triangle	0.89	0.90	0.90	90
up	0.94	0.93	0.94	90
accuracy			0.91	720
macro avg	0.91	0.91	0.91	720
weighted avg	0.91	0.91	0.91	720



## 1.5 Model selection

As the `classification_reports` show, the feature selection strategy does not affect the model performances by much, as the performances all follow the same trend: 1. knn has the highest avg accuracy, 2. nn has the second-highest avg accuracy, 3. svm is in second to last place in avg accuracy, 4. tree is always the last in avg accuracy and especially underperformed when using pca as feature selection strategy.

In order to avoid over-fitting, we decided choose a model that has a below 90% avg accuracy. We decided to opt for the svm model that used kBest as feature selection.

We also saved the model to disk using `joblib`.

```
[107]: import joblib

model = models["kBest"][1]["svm"]

# Save the model to a file
# depending on whether grid search was performed or not
joblib.dump(model.best_estimator_, 'model.joblib')
# joblib.dump(model, 'model.joblib')
```

```
[107]: ['model.joblib']
```

Model can now be imported from disk for other uses such as production

```
[ ]: loaded_model = joblib.load('model.joblib')
```