Delucchi Manuel - S4803977
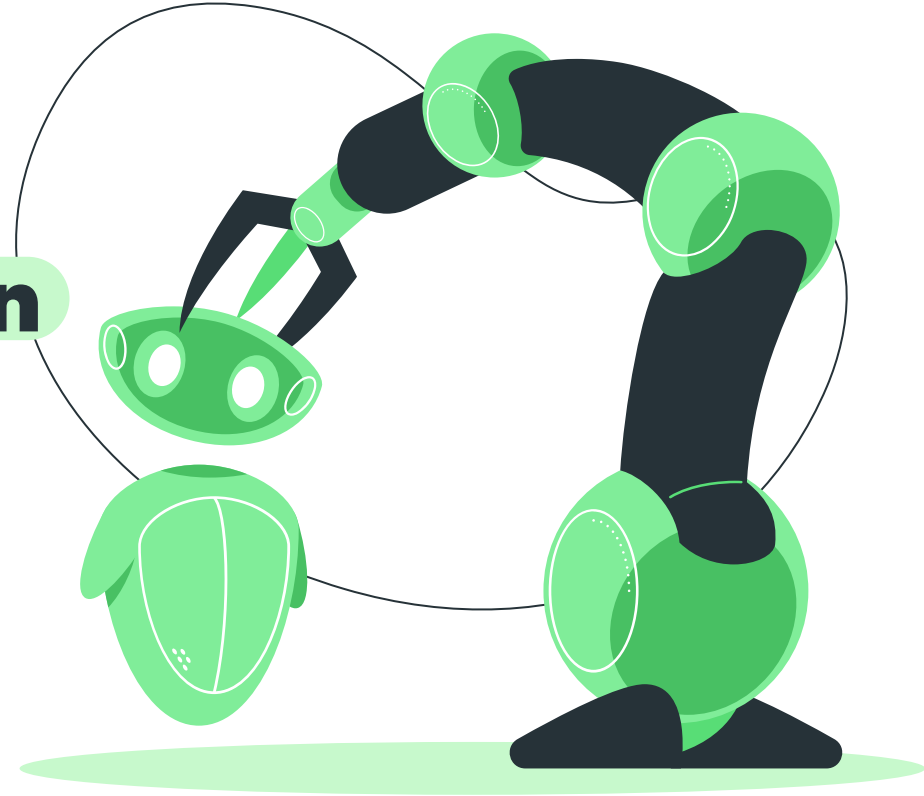Cappellini Matteo - S4822622

MYO Armband
Gesture Recognition

A recording of human hand muscle activity producing four different hand gestures

Project Description

MYO Armband is a wearable, non-invasive device used to acquire forearm stimuli corresponding to four distinct hand gestures.

It is equipped with an eight-channels sEMG sensor which designed to measures skin electromyography signals.
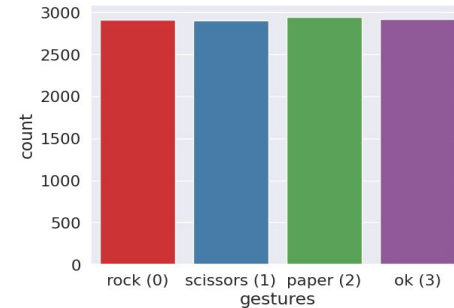
Dataset Description

The dataset can be found at: *https://www.kaggle.com/datasets/kyr7plus/emg-4/data*

The considered dataset includes 4 different .csv files, one for each different gesture recorded by the sensors on the MYO Armband.

The last column identifies with a number the gesture:

- *0 = rock*
- *1 = scissors*
- *2 = paper*
- *3 = ok*

The remaining 64 columns consist of sEMG datas, constituted of 8 consecutive readings from each of the 8 sensors.

Dataset Structure

	1_1	2_1	3_1	4_1	5_1	6_1	7_1	8_1	1_2	2_2	...	8_7	1_8	2_8	3_8	4_8	5_8	6_8	7_8	8_8	gestures
0	26.0	4.0	5.0	8.0	-1.0	-13.0	-109.0	-66.0	-9.0	2.0	...	-28.0	61.0	4.0	8.0	5.0	4.0	-7.0	-59.0	16.0	0
1	-47.0	-6.0	-5.0	-7.0	13.0	-1.0	35.0	-10.0	10.0	-4.0	...	-25.0	47.0	6.0	6.0	5.0	13.0	21.0	111.0	15.0	0
2	-19.0	-8.0	-8.0	-8.0	-21.0	-6.0	-79.0	12.0	0.0	5.0	...	-83.0	7.0	7.0	1.0	-8.0	7.0	21.0	114.0	48.0	0
3	2.0	3.0	0.0	2.0	0.0	22.0	106.0	-14.0	-16.0	-2.0	...	-38.0	-11.0	4.0	7.0	11.0	33.0	39.0	119.0	43.0	0
4	6.0	0.0	0.0	-2.0	-14.0	10.0	-51.0	5.0	7.0	0.0	...	38.0	-35.0	-8.0	2.0	6.0	-13.0	-24.0	-112.0	-69.0	0
11673	-3.0	-1.0	-1.0	-1.0	-28.0	20.0	5.0	0.0	-5.0	0.0	...	-3.0	1.0	4.0	3.0	4.0	-51.0	-49.0	5.0	-9.0	3
11674	-13.0	-5.0	-4.0	-3.0	-4.0	-24.0	-10.0	-8.0	20.0	9.0	...	6.0	-3.0	-3.0	-3.0	-5.0	-4.0	-45.0	-12.0	-15.0	3
11675	-1.0	-3.0	-1.0	1.0	30.0	38.0	-1.0	36.0	-10.0	1.0	...	14.0	-8.0	-4.0	-4.0	-4.0	-21.0	-29.0	-5.0	0.0	3
11676	1.0	4.0	4.0	5.0	9.0	-10.0	4.0	1.0	-2.0	-1.0	...	-16.0	-3.0	0.0	-3.0	-5.0	-36.0	-90.0	3.0	5.0	3
11677	-2.0	4.0	2.0	-4.0	12.0	3.0	-2.0	9.0	-8.0	-2.0	...	2.0	1.0	0.0	-1.0	-2.0	-30.0	64.0	11.0	5.0	3

The 4 .csv files are concatenated together to form a single dataset and the columns are labelled in the following way: *Sensor_MeasurementTime*

Dataset Shape: (11678, 65)

Data Preprocessing

```python
X = data.iloc[:, :-1]
Y = data.iloc[:, -1]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_state = 42)
X_val, X_test, Y_val, Y_test = train_test_split(X_test, Y_test, test_size = 1/3, random_state = 42)
sc = StandardScaler()
X_train_s = pd.DataFrame(sc.fit_transform(X_train))
X_test_s = pd.DataFrame(sc.transform(X_test))
X_val_s = pd.DataFrame(sc.transform(X_val))
```

X_train Shape: (8174, 64)   **Y_train Shape**: (8174, 1)
X_val Shape: (2336, 64)   **Y_val Shape**: (2336, 1)
X_test Shape: (1168, 64)   **Y_test Shape**: (1168, 1)

20%

10%

70%

Train   Validation   Test
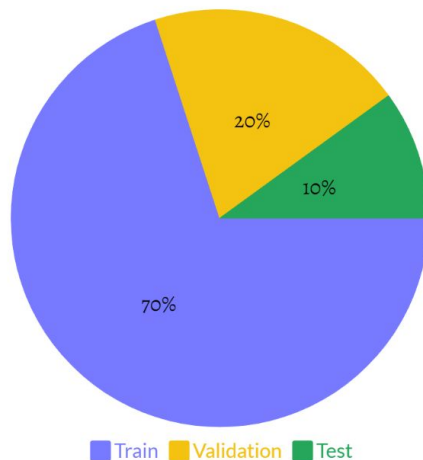
Classification

**Classification** refers to the machine's ability to assign instances to their correct groups.

- *Binary Classification*: the machine should classify an instance as only one of two classes (e.g. : yes/no, 1/0, true/false).
- *Multiclass Classification*: the machine should classify an instance as only one of three or more classes.

Our problem is a case of **Multiclass Classification**, that we solved by employing two different models:

- *Support Vector Machine (SVM)*
- *Random Forest*

SVMs – Support Vector Machines

SVMs are a supervised learning problem that aims to find an optimal decision boundary, or **hyperplane**, that maximizes the **margin** between two classes, which is the distance between the hyperplane and the **Support Vectors** (data points closest to the hyperplane).
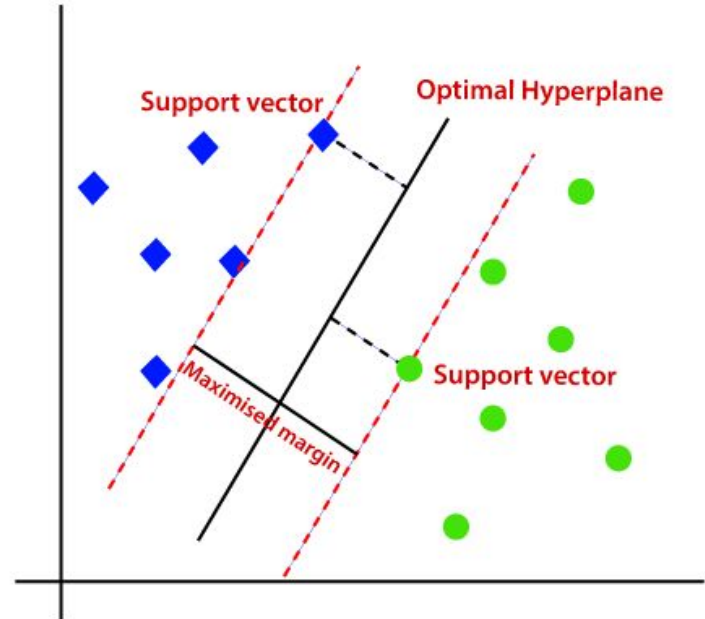
SVMs – Support Vector Machines

Linear SVM:
- Used for *linearly separable data*.
- The decision boundary is a straight line (or hyperplane in higher dimensions).
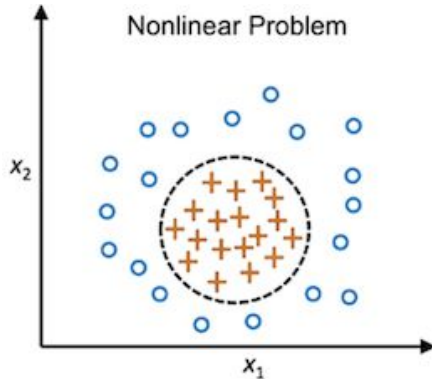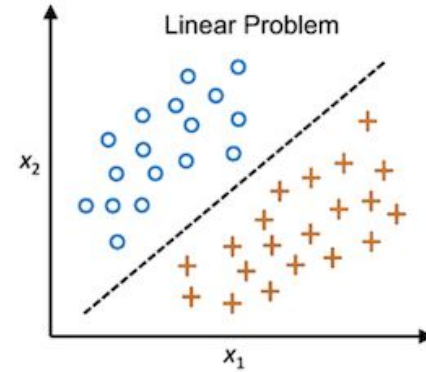
Non-Linear SVM:
- Used for *non-linearly separable data*, hence a hyperplane cannot separate the classes.
- *Strategy*: using *kernel functions*.

Kernel Function

The **kernel function** allows data to be mapped onto a higher-dimensional space where it becomes easier to find a hyperplane that can effectively separate different classes. It calculates the similarity between pairs of data points, allowing the SVM to work effectively in cases where the data are non-linearly separable.

Of the various kernel function options available, we have used the **Radial Basis Function (RBF)** kernel, also known as the *Gaussian kernel*, that is commonly employed in SVMs to capture non-linear patterns in the data efficiently.

kernel

Decision surface

$$K(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\gamma \cdot \|\mathbf{x}_1 - \mathbf{x}_2\|^2\right)$$
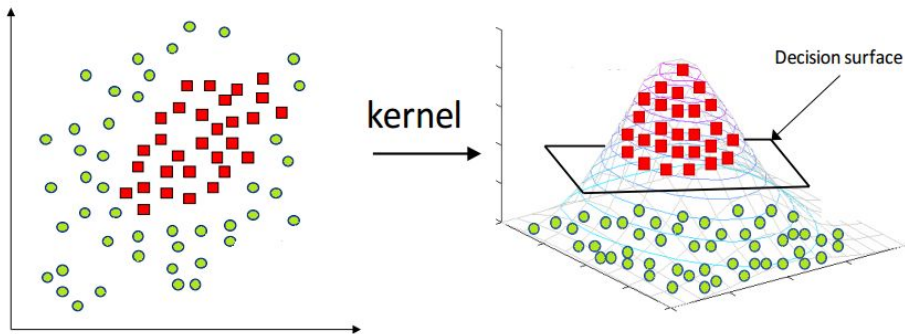
Hyperparameters Tuning

Gamma: Influences the shape of the decision boundary.

- *Small gamma*: decision boundary remains relatively smooth.
- *Large gamma*: decision boundary bends and twists more to correctly classify the training points (can lead to overfitting).

C (Regularization Parameter): Helps control the trade-off between maximizing the margin and minimizing the classification error..

- *Small C*: larger margin, some misclassifications are tolerated.
- *Large C*: smaller margin, tries to correctly classify more points.

```python
grid = {'C'      : [0.1, 1, 10, 100, 1000],
        'kernel' : ['rbf'],
        'gamma'  : [1, 0.1, 0.01, 0.001, 0.0001]
}
MS = GridSearchCV(estimator    = SVC(),
                  param_grid  = grid,
                  scoring     = 'accuracy',
                  cv          = 10,
                  verbose     = 2)
H = MS.fit(X_val_s, Y_val)
```
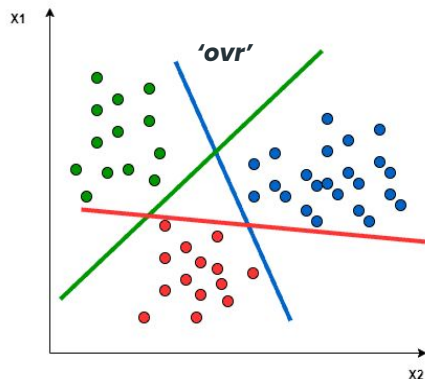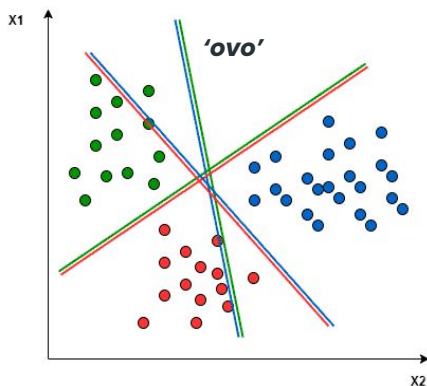
Best hyperparameters:
- *C*   = 10
- Gamma = 0.01

Decision Functions

SVM limit s that it works only with binary problem.
For multiclass classification the idea is to break down the problem into multiple binary classification problems.

- **One-vs-One (OvO)** approach: here the breakdown is set to a binary classifier per each pair of classes.

$$c\frac{c-1}{2}$$ - binary classification problems

- **One-vs-Rest (OvR)** approach: here the breakdown is set to a binary classifier per each class.

c - binary classification problems

Model

```
M = SVC(C                       = H.best_params_['C'],
        kernel                  = 'rbf',
        gamma                   = H.best_params_['gamma'],
        decision_function_shape = 'ovr')
M.fit(X_train_s, Y_train)
```
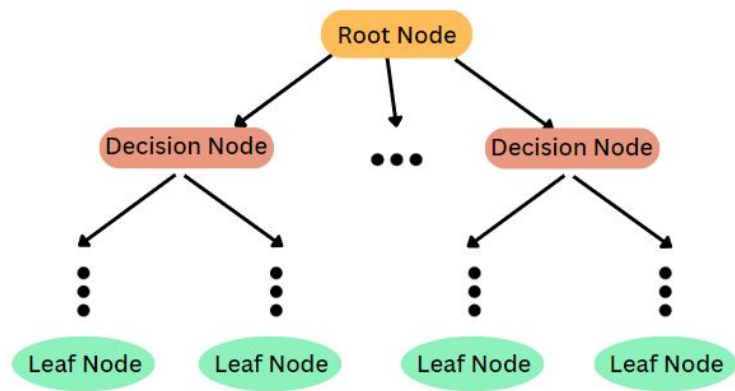
$$Accuracy = \frac{\sum_{i=1}^{n} \text{N}^{\circ} \text{ of Correctly Classified Instances for Class i}}{\text{Total Number of Instances}} * 100\% = 90.58\%$$

Confusion Matrix (SVC)
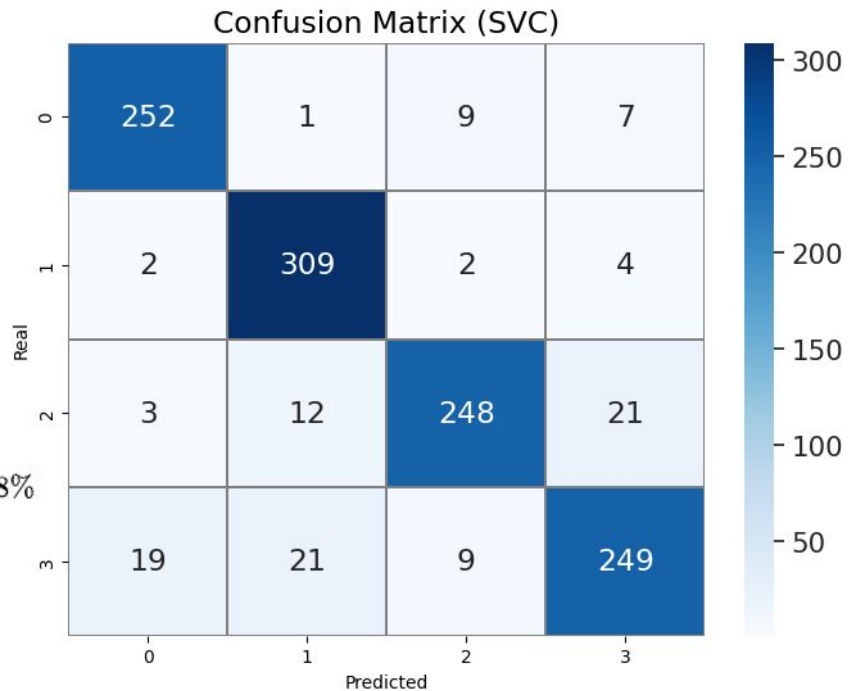
Decision Trees

A tree-shaped diagram that illustrates series of events leading to certain decisions

- Each node represents a test on an attribute and each branch is an outcome of that test

- Split the dataset in multiple sub-nodes, with the aim of maximizing the information gain and minimizing the entropy of the system

- Problems: **data bias** and **overfitting**

Random Forest

Random Forest is a **classification** algorithm defined as an **ensemble of decision trees**

Instead of relying on a single decision tree for predictions, this algorithm constructs multiple trees, each trained on different subsets of the training data.
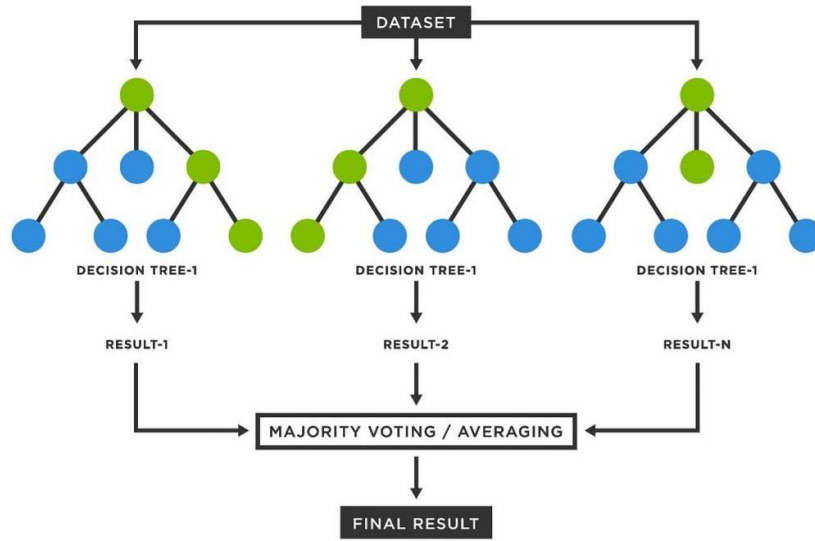
How does it work?

Bagging (Bootstrap Aggregating): Create multiple datasets for training each decision tree. This involves **randomly sampling the training data with replacement**, so that each data point can be selected multiple times or not at all, to create different subsets. Each decision tree is then trained on one of these subsets.

This rectify the overfitting problem of decision trees, reducing the variance and increasing the generalization of the model!

For **classification tasks** each decision tree in the forest independently predicts the class, then the final prediction is determined by a **majority vote among all the trees**.

A deeper tree can capture more details in the training data, but it might also overfit. By limiting the depth it is possible to achieve a better generalization.

Da quello che ho capito K-fold è importante per capire come è strutturato ogni albero su training e test ma idk

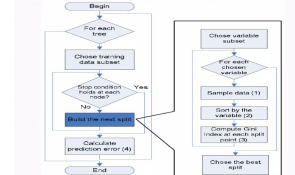For regression problems -> average value of all trees

the larger the better, but also the longer it will take to compute. Results will stop getting significantly better beyond a critical number of trees.

Boostrapping: ensures that we are not using the same data for every tree. -> model less sensitive to original training data

Random feature selection: reduce correlation between trees, if we use every feature then most trees will have the same decision nodes -> similarity, increase variance
Balance between bad prediction trees

https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/

Random forest algorithm(flow chart):

Hyperparameters Tuning

Max_features: the maximum size of features to consider when splitting a node. Can be equal to either:

- n (number of features)
- Sqrt(n)
- Log2(n)

Max_samples: determines what fraction of the original dataset is given to any individual tree in order to avoid the overfitting problem

```python
grid_search = {'max_features' : ['log2', 'sqrt', None],
               'max_samples' : [0.3, 0.5, 0.7]}
GS = GridSearchCV(estimator   = RandomForestClassifier(),
                  param_grid  = grid_search,
                  scoring     = 'accuracy',
                  cv          = 10,
                  verbose = 2)
H = GS.fit(X_val_s, Y_val)
```
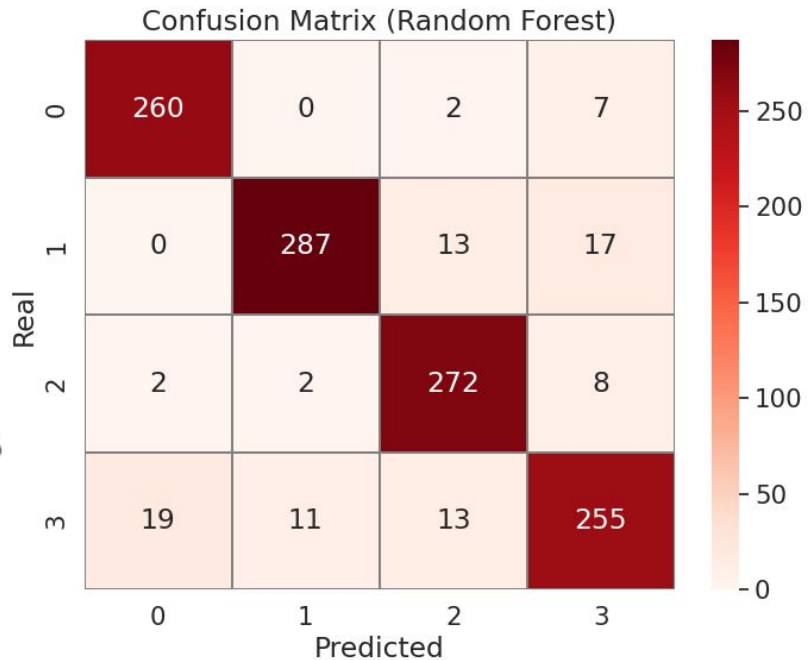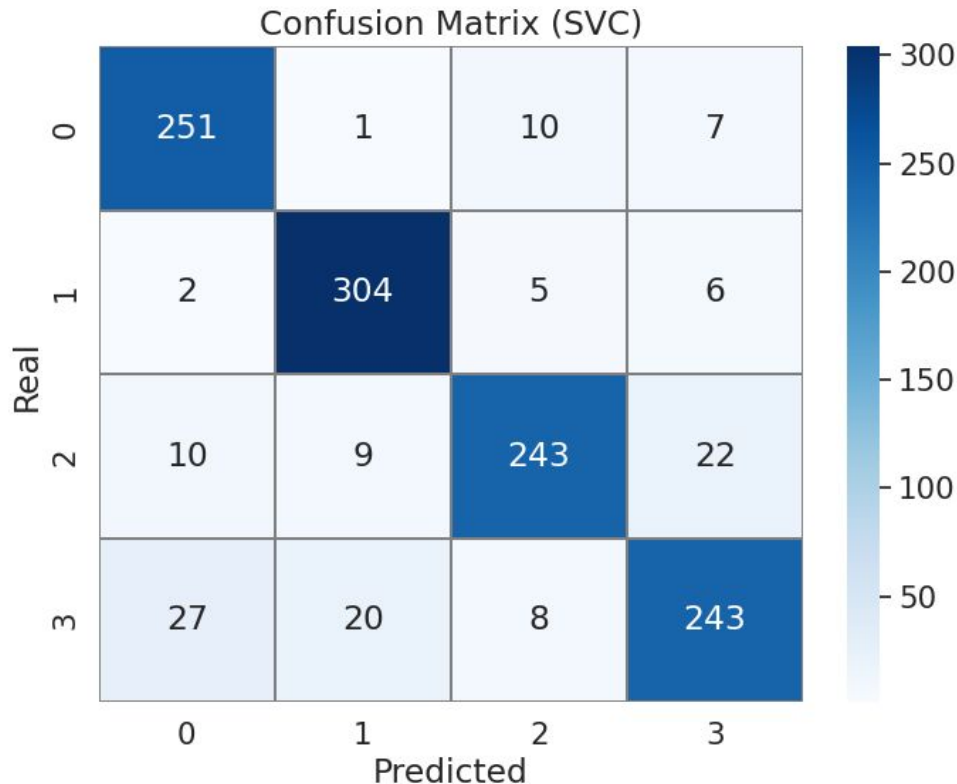
Best hyperparameters:
- Max_features = log2
- Max_samples = 0.5

Model

```python
M = RandomForestClassifier(max_features  =
H.best_params_['max_features'],
                           max_depth       =
H.best_params_['max_depth'])
M.fit(X_train_s, Y_train)
```

$$Accuracy = \frac{\sum_{i=1}^{n} \text{N}^\circ \text{ of Correctly Classified Instances for Class i}}{\text{Total Number of Instances}} * 100\% = 91.95\%$$

Confusion Matrix (Random Forest)

Confusion Matrix

Confusion Matrix (SVC)

- **TP**: Correctly predicted instances for each class (along the diagonal).
- **FP**: Negative value incorrectly predicted as positive.
- **FN**: Positive value incorrectly predicted as negative.
- **TN**: Instances of each class that are correctly predicted as negative.

Y_test Shape: (1168, 1)

Ex: Class 0 (Rock)
- TP = 251
- FP = 2+10+27 = 39
- FN = 1+10+7 = 18
- TN = 1168 – (TP+FP+FN) = 860

Confusion Matrices Comparison

Confusion Matrix (SVC)

Confusion Matrix (Random Forest)
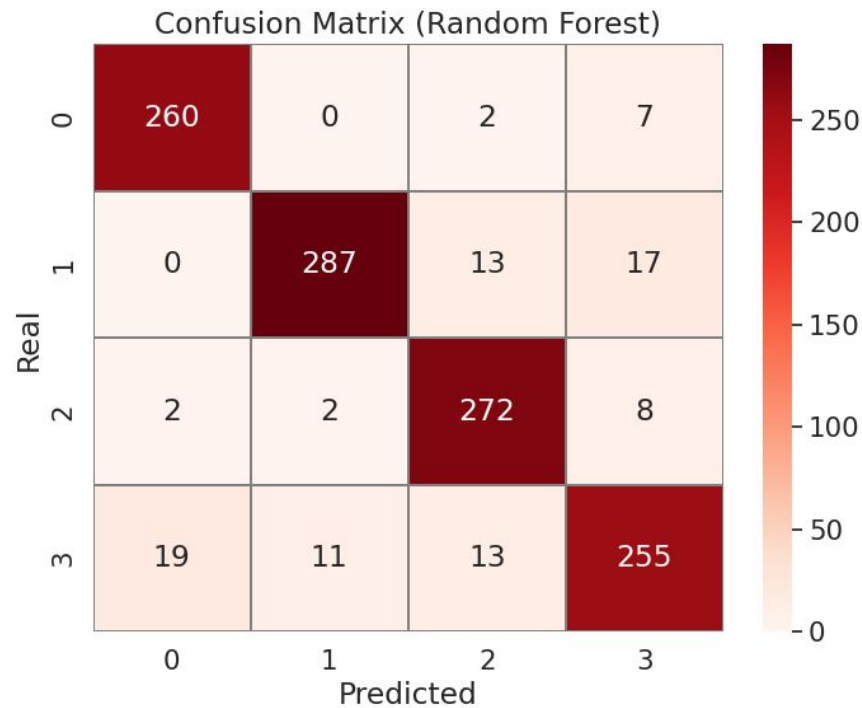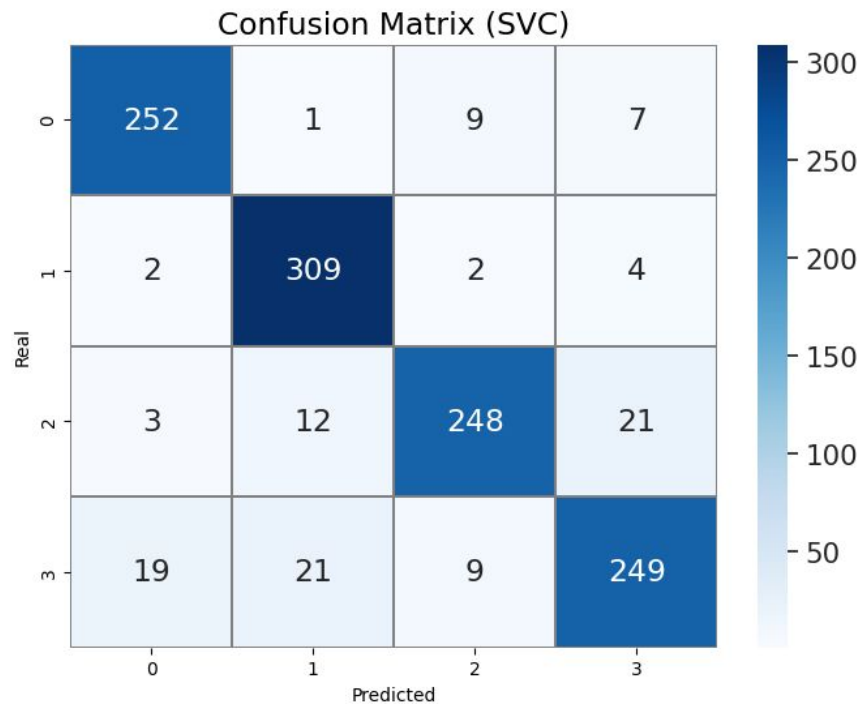
Y_test Shape: (1168, 1)

Classification Report

SVC

	precision	recall	f1-score	support
0	0.91	0.94	0.92	269
1	0.90	0.97	0.94	317
2	0.93	0.87	0.90	284
3	0.89	0.84	0.86	298
accuracy			0.91	1168

Random Forest

	precision	recall	f1-score	support
0	0.93	0.97	0.95	269
1	0.96	0.91	0.93	317
2	0.91	0.96	0.93	284
3	0.89	0.86	0.87	298
accuracy			0.92	1168

Precision: number of true positive outcomes out of all positive predictions

$$Prec = \frac{TP}{TP + FP}$$

Recall: ratio of true positives predictions wrt the sum of true positives and false negatives

$$Rec = \frac{TP}{TP + FN}$$

F1 score: weighted harmonic mean between precision and recall

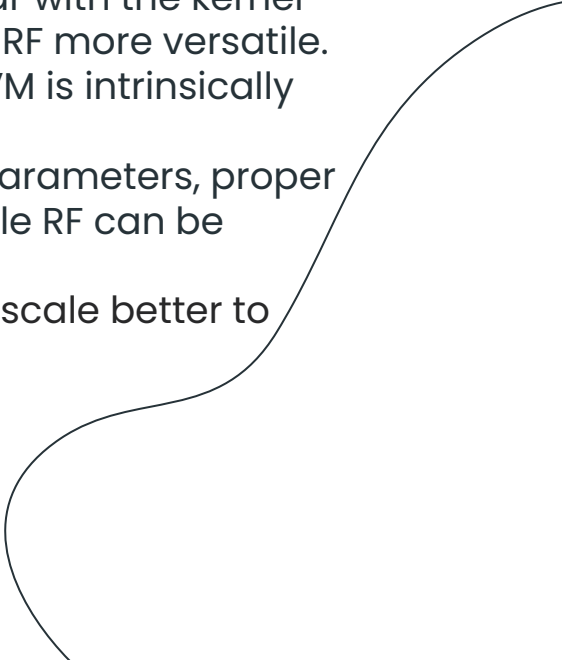$$f1 = 2\frac{Prec * Rec}{Prec + Rec}$$

Accuracy: proportion of correctly classified data points out of all the data points

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}$$

Results comparison

	SVC	RF
Precision	0.905870	0.919886
Recall	0.905822	0.919521
F1 Score	0.905846	0.919703
Accuracy	0.905822	0.919521

SVM vs Random Forest

- RF has been proven more accurate in our case.
- SVM is a linear model that can be extended to non-linear with the kernel trick, while RF is an ensemble non-linear model, making RF more versatile.
- RF is intrinsically suited for multiclass problems, while SVM is intrinsically two-class.
- SVM's performance can be highly sensitive to its hyperparameters, proper tuning is crucial for achieving optimal performance, while RF can be considered more robust and less dependant to them.
- SVMs can struggle with very large datasets, whereas RF scale better to large datasets.

Thanks!