## TO START

```
# IMPORT DATA LIBRARIES
import pandas as pd
import numpy as np
# IMPORT VIS LIBRARIES
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# IMPORT MODELLING LIBRARIES
from sklearn.model_selection import train_tes-
t_split
from sklearn.svm import SVC
from sklearn.metrics import classification_rep-
ort,confusion_matrix
```

## TRAIN MODEL

### SPLIT DATASET

| | |
|---|---|
| X = df[['col1','col2',etc.]] | create df features |
| y = df['col'] | create df var to predict |
| X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.3) | split df in train and test df |

### FIT THE MODEL

| | |
|---|---|
| svc= SVC() | instatiate model |
| svc.fit(X_train,y_train) | train/fit the model |

### MAKE PREDICTIONS

```
pred = svm.predict(X_test)
```

### EVAUATE MODEL

```
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

## GRID SEARCH EXPLANATION

Finding the right parameters (like what C or gamma values to use) is a tricky task! But luckily, we can be a little lazy and just try a bunch of combinations and see what works best! This idea of creating a 'grid' of parameters and just trying out all the possible combinations is called a Gridsearch, this method is common enough that Scikit-learn has this functionality built-in with GridSearchCV! The CV stands for cross-validation which is the GridSearchCV takes a dictionary that describes the parameters that should be tried and a model to train. The grid of parameters is defined as a dictionary, where the keys are the parameters and the values are the settings to be tested..

==========

**C** is the parameter for the soft margin cost function, which controls the influence of each individual support vector; this process involves trading error penalty for stability. C is the **cost of misclassification of training examples** against the simplicity of the decision surface. A **large C** gives low bias and high variance. Low bias because you penalize the cost of missclasification a lot. A **small C** gives you higher bias and lower variance.

**Gamma** is the parameter of a **Gaussian Kernel** (to handle non-linear classification). Gamma **controls the shape of the "peaks"** where you raise the points. A small gamma gives a pointed bump in the higher dimensions, a large gamma gives a softer, broader bump. So a **small gamma** will give you low bias and high variance while a **large gamma** will give you higher bias and low variance. You usually find the best C and Gamma hyper-parameters using Grid-Search.

**Kernel** will decide the hyperplane you will use to divide the points.

==========

**Refit** an estimator using the best-found parameters on the whole dataset.

**Verbose** controls the verbosity: the higher, the more messages.

## SVM parameters

Following table should sum up the explanation for an SVM-

| | Large Gamma | Small Gamma | Large C | Small C |
|---|---|---|---|---|
| Variance | Low | High | High | Low |
| Bias | High | Low | Low | High |

The art is to choose a model with optimum variance and bias. Therefore you need to choose the values of C and Gamma accordingly.

## GRID SEARCH

| | |
|---|---|
| **from sklearn.model_selection import GridSearchCV** | import GridSearch |
| param_grid = {<br>'C': [0.1,1, 10, 100, 1000],<br>'gamma': [1,0.1,0.01,0.001,0.0001],<br>'kernel': ['rbf']} | parameters, see info |
| grid = GridSearchCV(<br>SVC(),<br>param_grid,<br>refit=True,<br>verbose=3) | parameters, see info |
| grid.fit(X_train,y_train) | |
| grid.best_params_ | |
| grid.best_estimator_ | |
| grid_predictions = grid.predict(X_test) | |
| print(confusion_matrix(y_test,grid_predictions)) | |
| print(classification_report(y_test,grid_predictions)) | |