Support Vector Machines

# Introduction

* SVMs are supervised learning models with associated learning algorithms that analyze data and recognize patterns and are used for regression and classification analysis.
* SVM uses training data assigned to specific classes to classify future data points. It represents the examples a map in point and space with as wide as possible gap between.
* The categories are separated by a hyper plane (a line that perfectly separates the two categories) along with a hyper plane margin that extends to the edge of the separate categories.
* The points in a category that touches the hyper plane margins are referred to as support vectors.
* The above applies to linearly separable categories. For mas that cannot be linearly separated, the option is to use ‘kernel trick’ which is essentially viewing the same data in a higher dimension or plane.

# SVM with python

* After the data cleaning and EDA, split the data to the training and testing sets.
* Import SVC from sklearn.svm
* Instantiate the model.
* Fit (train) the model.
* Predict with the model.
* Import and print classification report and confusion matrix.
* The above steps (using the default model parameters) usually return very poor results quite often and require using a method called grid search to find the suitable parameters such as C and gamma.

## Grid search

* It is used to find the best combination of parameters for a model.
* Import GridSearchCV from sklearn.grid\_search
* Create a dictionary of the parameters to test e.g., grid\_parameters = {‘C’:[a, b, c, d, …], ‘gamma’:[x, y, z, …]} where a, b,…x, y, z are numbers.
* The C and gamma are responsible for the bias and variance trade off.
* Instantiate the GridSearchCV e.g., grid = GridSearchCV (estimator = SVC(), param\_grid= grid\_parameters, verbose = d) where the verbose argument is used to show some details while the process is running and d is an integer.

Note: GridSearchCV has been moved into the sklearn family of model\_selection

* Fit (train) the model.
* You can check the best set of parameters with grid.best\_params\_, best estimator with grid.best\_estimator\_
* You can then rerun the prediction by calling predict off the grid object e.g., grid.predict(X\_test)
* Then print the confusion matrix and classification report again.
* Grid searches can take a really on time if the data set is large or you are running a number of parameters. In reality, it is better done by making sure your data is properly cleaned up, then, running the grid search with a few parameters. If this runs well, then, set your full parameters and run it while you take a break to go do something else.