**Classification Preferable Methods for Executing in Python**

* K Nearest Neighbours
* Decision Tree Classifier
* Logistic Regression

**Packages**

**KNN:** from sklearn.neighbors import KNeighborsClassifier

**Decision Tree Classifier:** from sklearn.tree import DecisionTreeClassifier

**Logistic Regression:** from sklearn.linear\_model import LogisticRegression

**Important steps in executing any algorithm in our Dataset**

* Importing suitable packages for particular algorithm like KNN, Logistic regression etc.

**from sklearn.neighbors import KNeighborsClassifier**

* Instantiate the variable with function of that algorithm like

**knn = KNeighborsClassifier(<parameters>)**

* Fit the features and the target to the variable using fit function

**knn.fit(X, Y)**

**Where X – Features records, Y – Target**

Scores calculation differs from each algorithm

* Now the **knn** variable has memorised the training set or in other words, learned the dataset. Now it is time to test on our testing data with only features to predict the target or labels based on training of training set.
* Testing can be done by using syntax

**knn.predict(testing)**

* The above code will display the labels for all the testing data. But the labels prediction will differ from algorithm to algorithm and will differ from different parameter value in same algorithm

**For example**: knn= KNeighborsClassifier (n\_neighbors=3) and knn1 = KNeighborsClassifier (n\_neighbors=5) may produce different prediction results on same testing set.

And also, KNN algorithm and Logistic Regression algorithm will produce different results on predictions

**How to find out which one algorithm with specific parameter is predicting results correctly in classification ?**

There are some evaluation methods that are used for find the algorithm which performs prediction better on out of sample data or testing data.

* Training and Testing on same dataset
* Train/Test Split

**Training and Testing on same dataset**

In this method, training and testing is performed on the same training dataset and accuracy is determined for each cases using metrics library

For example:

**# Package : from sklearn.neighbors import KNeighborsClassifier**

**knn = KNeighborsClassifier(n\_neighbors=5)**

**knn.fit (training[feature\_names], target)**

**prdknntrain = knn.predict(training[feature\_names])**

**# Package : from sklearn.metrics import accuracy\_score**

**accuracy\_score (target, prdknntrain)**

This gives the accuracy score on training dataset. The point in doing predictions on training set itself is that you know the exact label for the features as you are using training dataset. Calculating predictions for training set and comparing it with labels already available will provide how well it performs on training set and we can give assurance that it will do the same in the testing set.

Therefore, the one which gets the maximum accuracy score will be chosen as a model for this particular dataset.

But the **drawback in using this method** is that increased accuracy score will lead to over complex models, thus rewarding with the term “**Over fitting**”.

**Train/Test Split**

In this method, training set is itself split into training and testing samples. Generally, about 60% of dataset is taken as training data and remaining 40% is allocated as testing data.

This is also possible by using library **cross-validation** from sklearn

**from sklearn.cross\_validation import train\_test\_split**

The syntax for Train/Test split is

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=4)**

where total dataset ‘X’ is divided into 4 namely **X\_train, X\_test, y\_train, y\_test**

**X\_train, X\_test –** Features data of training and testing samples respectively.

**y\_train, y\_test –** Label (Output) data of training and testing samples respectively

You can specify how much of data can be used for testing by parameter called **test\_size** inside train\_test\_split function.

If **test\_size = 0.4,** it means 40 % data is to be taken for testing. Remaining for training.

**Random state** parameter is used in such a way that if you don’t mention random state in your syntax, every time you run this code, it takes different samples of training and testing for prediction

So, it is preferred to allocate random state with some value so that it doesn’t take different sets of training and testing data each time.

And then, the usual routine process

**knn = KNeighborsClassifier(n\_neighbors=5)**

**knn.fit(X\_train, y\_train)**

**y\_pred = knn.predict(X\_test)**

**print(metrics.accuracy\_score(y\_test, y\_pred))**

Now the algorithm which produces maximum accuracy is selected as the best algorithm for solving the problem. Training and testing different samples of same dataset will be more efficient than doing both on same data.

**But the drawback** is that **different random state values** produce different accuracy scores.

**Now what can we do?**

The answer is **K-Fold Cross Validation**.

K-Fold cross validation divides the data into K folds and in that 1 fold is treated as testing data and remaining K-1 folds are treated as testing data and prediction process run as usual based on algorithm flow.

**Looks like Train/Test split?**

Not at all, this process lasts for just 1 iteration. In the next iteration, a new fold is treated as testing data and remaining as training data. Likewise, all the K folds will once be testing data in K-1 iterations. Therefore all folds of data are thoroughly trained and tested and mean value for score produced in each iteration will be efficiency score of that particular algorithm.

**Implementation of K-Fold Cross Validation**

**from sklearn.cross\_validation import cross\_val\_score**

**knn = KNeighborsClassifier(n\_neighbors=5)**

**scores = cross\_val\_score(knn, X, y, cv=10, scoring='accuracy')**

**print(scores)**

Here **cv** represents number of folds the data has to be divided.

Printing scores will display accuracy scores for 10 different folds. The mean of these will give me one perfect accuracy score for particular algorithm

This is for 10 folds. Different value of cv will produce different accuracy score. To find which value of cv produces maximum accuracy, draw graph for different values of cv and their mean score and find which value produces maximum score for one particular algorithm

**Is there a better way to tune parameters of algorithm automatically?**

There is. It is called Grid Search Cv which automatically tune parameters in K-Fold cross validation for better accurate results.

**Regression:**

Regression is method used for predicting continuous value output.

**Linear Regression** is one of the best and oldest method for solving regression problems

**Before performing linear regression**, we have some function to visualise the performance of each features to that of output. It is by using a package called **seaborn.** This can be used to find the importance of particular feature contributing to the output and can draw conclusions on feature selections also.

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**sns.pairplot(data, X\_vars=[‘Features’], y\_vars = [‘Target’] , size=7, aspect=0.7, kind = ‘reg’)**

Here size and aspect parameters are related to the graph plot and kind =’reg’ will draw the hypothesis line that fits the given data.

Linear Regression can be implemented for datasets in python using sklearn library called “**linear\_model**”

**from sklearn.cross\_validation import train\_test\_split**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=4)**

**From sklearn.linear\_model import LinearRegression**

**linreg = LinearRegression()**

**linreg.fit(X\_train, y\_train)**

Normally fit() function will learn the training data or memorises the data. But in the case of linear regression, it calculates the intercept and parameters for each features in the dataset.

**linreg.intercept\_**

**linreg.coef\_**

Wherelinreg.intercept\_ prints the intercept and linreg.coef\_ prints the parameters associated with hypothesis of linear regression.

**Evaluation methods for Linear Regression**

Like classification, there are also evaluation methods to determine the efficiency of linear regression algorithm. They are

* Mean Absolute error
* Root mean square error

**Mean squared error**

Mean squared error is defined as the mean value of square of difference between true value and the predicted value.

It can be found by using metrics package

**metrics.mean\_squared\_error(ytest, ypred)**

**Root Mean square error**

Root mean squared error as the name suggests, is the value of square root of mean squared error.

**np.sqrt(metrics.mean\_squared\_error(ytest,ypred))**

On comparing, RMSE is a better evaluation method for regression problems than MSE as it is the standard error correction mechanism used for all kind of problems.

**Feature Selection in Classification and Regression**

Thankfully for both classification and regression, we have a technique to find the features which really contribute to the label or target. There are numerous methods to do this but **SelectKBest** library in sklearn offers you the best feature selection.

**from sklearn.feature\_selection import SelectKBest**

This library is common for both classification and regression. But to find best features in classification, use f\_classify and for regression, use f\_regression.

**from sklearn.feature\_selection import f\_classify**

**from sklearn.feature\_selection import f\_regression**

Like all algorithm implementation, instantiate the function “SelectKbest” with parameter “f\_classify” for classification or “f\_regression” for regression to a variable and fit that to data and target.

**selector = SelectKBest(f\_regression, k=3)**

**selector.fit(train[feature\_names], target)**

Here parameter **k** defines number of best features to be selected out of all features of the dataset.

Now **selector.pvalues\_** will display the score of the features on the basis of its contribution to the output. The values will be very low in order of negative exponential.

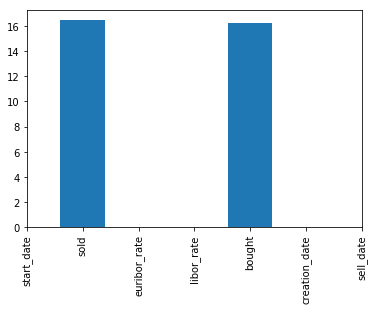
Plotting the features with their scores will have a better angle on selection than displaying the scores alone for which **matplotlib** offers lot more than any other.

**plt.bar(range(len(feature\_names)), scores)**

**plt.xticks(range(len(feature\_names)), feature\_names, rotation='vertical')**

**plt.show()**

This will display graph like this

****

It applies the same for regression too.