BOSTON DATASET ANALYSIS WITH MACHINE LEARNING

A Project Report for Winter Industrial Training

Submitted by

MayurakshaSikdar

Subha Prasad Dey

Debajyoti Ghosh

in partial fulfillment for the award of the degree of

B. Tech

in

Computer Science & Engineering

Narula Institute of Technology

At

Ogma TechLab Pvt. Ltd.

empowering technologies . enrich ideas

Ogma TechLab Pvt. Ltd.

BONAFIDE CERTIFICATE

Certified that this project work was carried out under my supervision

"Machine Learning with Python" is the bonafide work of

Name of the student: Mayuraksha Sikdar ***Signature:***

Name of the student: Subha Prasad Dey ***Signature:***

Name of the student: Debajyoti Ghosh ***Signature:***

SIGNATURE

Name : Bed Prakash Das
PROJECT MENTOR

OgmaTechLab Original Seal

Acknowledgement

I take this opportunity to express my deep gratitude and sincerest thank to my project mentor, Mr. **BED PRAKASH DAS** for giving most valuable suggestion, helpful guidance and encouragement in the execution of this project work.

I will like to give a special mention to my colleagues. Last but not the least I am grateful to all the faculty members of OgmaTechLab Pvt. Ltd. or their support.

Content

ABSTRACT:

In this project, we will evaluate the performance and predictive power of a model that has been trained and tested on data collected from homes in suburbs of Boston, Massachusetts. A model trained on this data that is seen as a *good fit* could then be used to make certain predictions about a home — in particular, its monetary value. This model would prove to be invaluable for someone like a real estate agent who could make use of such information on a daily basis.

INTRODUCTION:

The Boston housing data was collected in 1978 and each of the 506 entries represent aggregated data about 14 features for homes from various suburbs in Boston, Massachusetts.

PROJECT CATEGORY:

Machine Learning

TOOLS/PLATFORM, HARDWARE AND SOFRWARE REQUIREMENT SPECIFICATIONS:

1) **TOOLS :**
   - Anaconda Navigator
   - Anaconda Prompt
   - Jupyter Notebook
   - MS Office

2) **PLATFORM :**

Windows 7/8/10

3) **HARDWARE REQUIREMENT SPECIFICATION :**

Client Machine	
HDD	200MB

Processor	Pentium 4 or newer processor that supports SSE2
Memory	2GB

4) **SOFTWARE REQUIREMENT SPECIFICATION :**

Client Machine	
Browser	Any standard browser
Python	Version 3
Anaconda	Version 6.7

GOALS OF IMPLEMENTATION:

The implementation aims for gaining expertise in Data Analysis with Machine Learning approach.

PROBLEM DEFINATION:

Implementation of Machine Learning Algorithm for Prediction of residential house price in the Boston housing market dataset.

DATA DESCRIPTION:

This dataset contains information collected by the U.S Census Service concerning housing in the area of Boston Mass.

The Boston data frame has 506 rows and 14 columns.

This data frame contains the following columns:

crim

per capita crime rate by town.

zn

proportion of residential land zoned for lots over 25,000 sq.

indus

proportion of non-retail business acres per town.

chas

Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).

nox

nitrogen oxides concentration (parts per 10 million).

rm

average number of rooms per dwelling.

age

proportion of owner-occupied units built prior to 1940.

dis

weighted mean of distances to five Boston employment centers.

rad

index of accessibility to radial highways.

tax

full-value property-tax rate per $10,000.

ptratio

pupil-teacher ratio by town.

lstat

lower status of the population (percent).

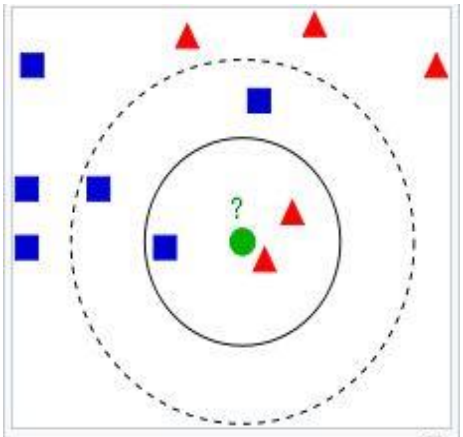
medv

median value of owner-occupied homes in $1000s.

MODELS:

1) **KNN (k-nearest neighbor) :**

The ***k*-nearest neighbors algorithm** (***k*-NN**) is a non-parametric method used
for classification and regression In both cases, the input consists of the *k* closest training examples in the feature space. The output depends on whether *k*-NN is used for classification or regression: In *k-NN classification*, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k* is a positive integer, typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.



Example of *k*-NN classification. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If *k* = 3 (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If *k* = 5 (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

- In *k-NN regression*, the output is the property value for the object. This value is the average of the values of its *k* nearest neighbors

KNN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The *k*-NN algorithm is among the simplest of all machine learning algorithms.

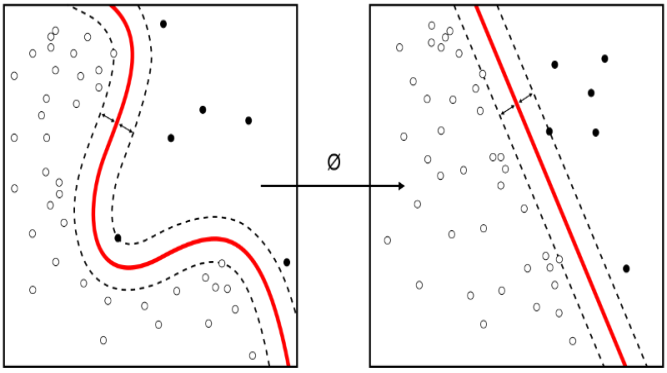
Both for classification and regression, a useful technique can be used to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/*d*, where *d* is the distance to the neighbor.

The neighbors are taken from a set of objects for which the class (for *k*-NN classification) or the object property value (for *k*-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

A peculiarity of the *k*-NN algorithm is that it is sensitive to the local structure of the data.

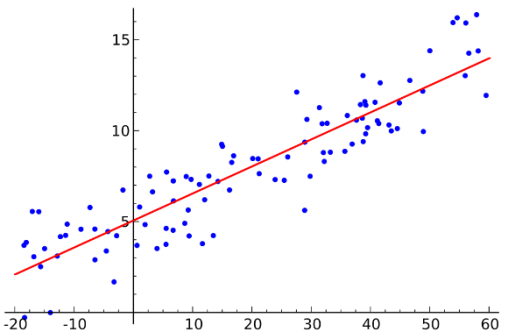
2) <u>SVM (Support Vector Machine) :</u>

In machine learning, **support-vector machines** (**SVMs**, also **support-vector networks**) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. When data is unlabelled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups. The **support-vector clustering** algorithm, created by Hava Siegel Mann and Vladimir Vapnik, applies the statistics of support vectors, developed in the support vector machines algorithm, to categorize unlabeled data, and is one of the most widely used clustering algorithms in industrial applications.



3) Linear Regression :

In statistics, **simple linear regression** is a linear regression model with a single explanatory variable That is, it concerns two-dimensional sample points with one independent variable and one dependent variable (conventionally, the *x* and *y* coordinates in a Cartesian coordinate system) and finds a linear function (a non-vertical straight line) that, as accurately as possible, predicts the dependent variable values as a function of the independent variables. The



adjective *simple* refers to the fact that the outcome variable is related to a single predictor.

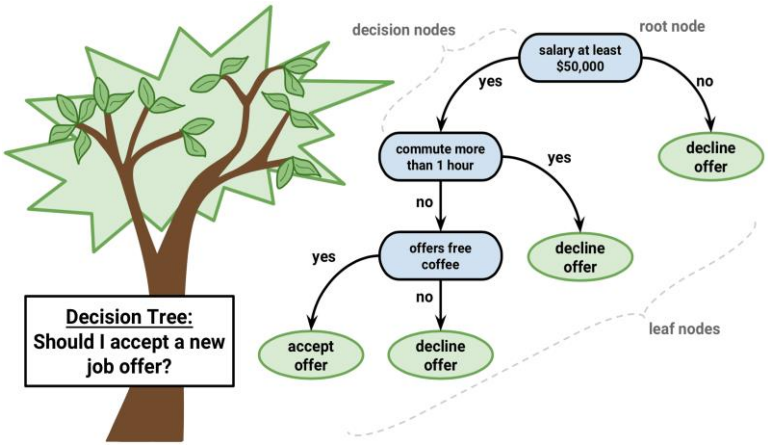
It is common to make the additional stipulation that the ordinary least squares method should be used: the accuracy of each predicted value is measured by its squared *residual* (vertical distance between the point of the data set and the fitted line), and the goal is to make the sum of these squared deviations as small as possible. Other regression methods that can be used in place of ordinary least squares include least absolute deviations (minimizing the sum of absolute values of residuals) and the Theil–Sen estimator (which chooses a line whose slope is the median of the slopes determined by pairs of sample points). Deming regression (total least squares) also finds a line that fits a set of two-dimensional sample points, but (unlike ordinary least squares, least absolute deviations, and median slope regression) it is not really an instance of simple linear regression, because it does not separate the coordinates into one dependent and one independent variable and could potentially return a vertical line as its fit.

The remainder of the article assumes an ordinary least squares regression. In this case, the slope of the fitted line is equal to the correlation between *y* and *x* corrected by the ratio of standard deviations of these variables. The intercept of the fitted line is such that the line passes through the center of mass (*x*, *y*) of the data points.

4) Decision Tree :

Decision Tree Analysis is a general, predictive modelling tool that has applications spanning a number of different areas. In general, decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning. Decision trees are a non-parametric supervised learning method used for both classification and regression tasks. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

In computer science, **Decision tree learning** uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It is one of the predictive modeling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a discrete set of values are called **classification trees**; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called **regression trees**.



In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision-making. In data mining, a decision tree describes data (but the resulting classification tree can be an input for decision-making). This page deals with decision trees in data mining.

5) Naïve Bayes :

In machine learning, **naive Bayes classifiers** are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

Naive Bayes has been studied extensively since the 1950s. It was introduced under a different name into the text retrieval community in the early 1960s,[1]:488 and remains a popular (baseline) method for text categorization, the problem of judging documents as belonging to one category or the other (such as spam or legitimate, sports or politics, etc.) with word frequencies as the features. With appropriate pre-processing, it is competitive in this domain with more advanced methods including support vector machines It also finds application in automatic medical diagnosis.

Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which

takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

In the statistics and computer science literature, naive Bayes models are known under a variety of names, including **simple Bayes** and **independence Bayes**. All these names reference the use of Bayes' theorem in the classifier's decision rule, but naive Bayes is not (necessarily) a Bayesian method

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.

For some types of probability models, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood; in other words, one can work with the naive Bayes model without accepting Bayesian probability or using any Bayesian methods.

Despite their naive design and apparently oversimplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations. In 2004, an analysis of the Bayesian classification problem showed that there are sound theoretical reasons for the apparently implausible efficacy of naive Bayes classifiers. Still, a comprehensive comparison with other classification algorithms in 2006 showed that Bayes classification is outperformed by other approaches, such as boosted trees or random forests.

An advantage of naive Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification.

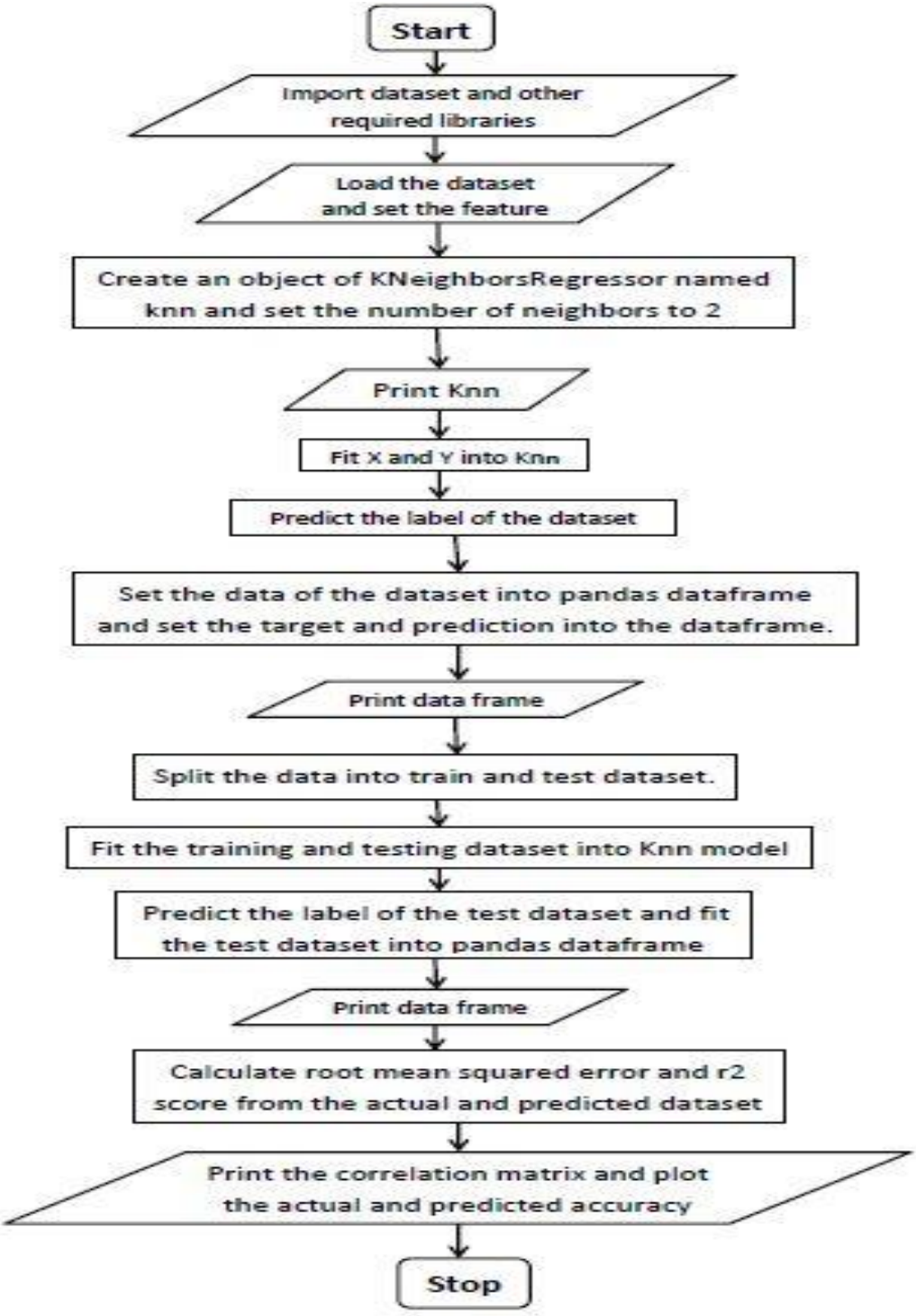
ALGORITHM & FLOWCHART:

KNN ALGORITHM:

- Import sklearn.datasets, matplotlib.pyplot as plt, numpy as np.

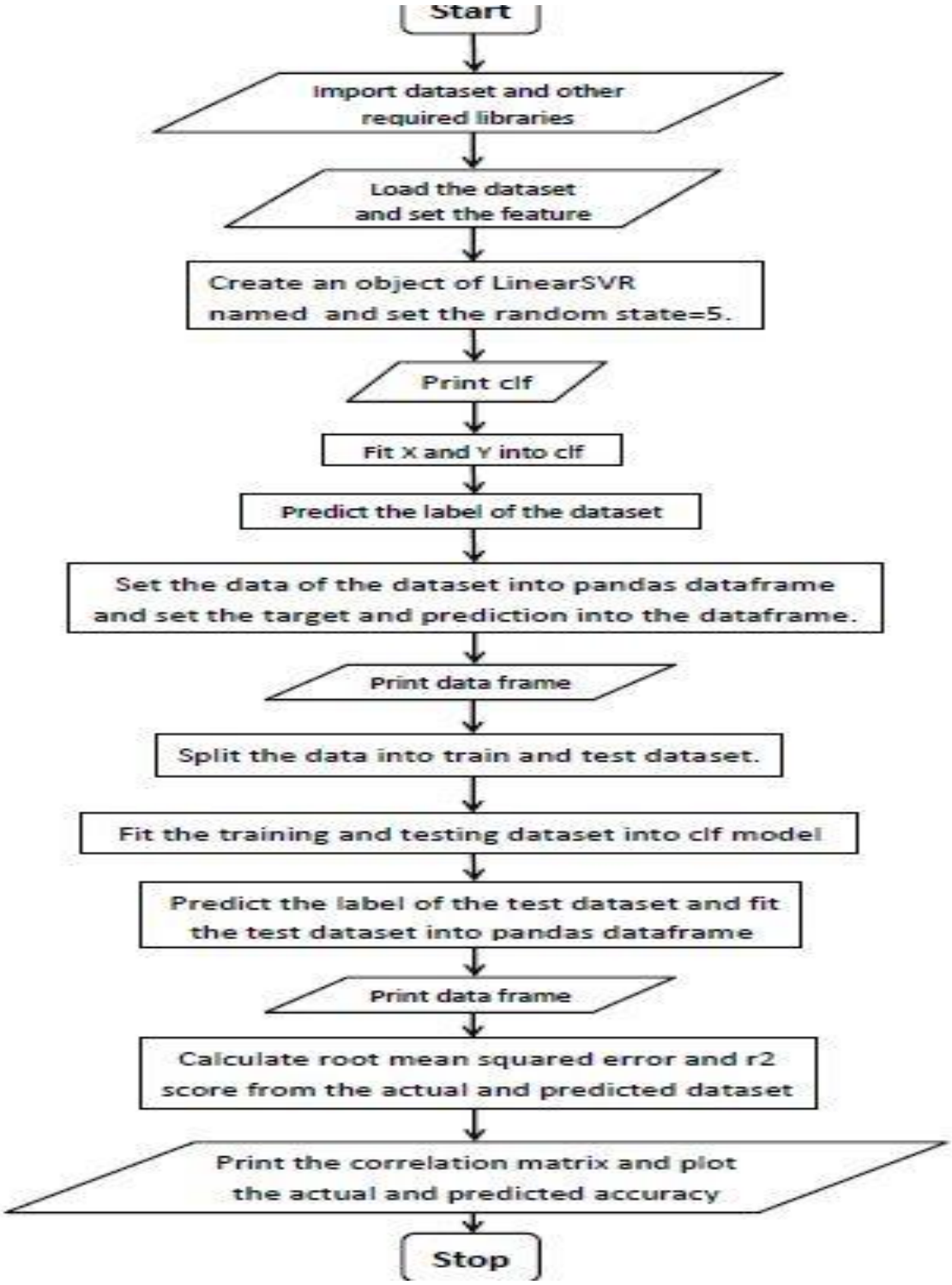
- Load the boston dataset into a variable named boston.
- Set the features and label of the boston dataset into x and y respectively.
- Import KNeighborsRegressor from sklearn.datasets
- Create an object of KNeighborsRegressor named knn and set the number of neighbors to 2.
- Print knn.
- Fit x and y into knn.
- Predict the label of the dataset from knn model from its features.
- Import pandas as pd.
- Set the data of the dataset into pandasdataframe and set the target and prediction into the dataframe.
- Print df.
- From sklearn import model_selection.
- Split the data in 70:30 ratio into train and test dataset.
- Fit the training dataset into knn model.
- Fit the test dataset into knn model.
- Now predict the label from test dataset.
- Fit the test dataset into pandasdataframe.
- Print df.
- Import seaborn as sns.
- From sklearn.metrics import mean_squared_error, r2_score/
- Calculate root mean squared error from actual dataset and predicted dataset.
- Calculate r2 score from the same.
- Now print correlation matrix using seaborn library.
- Import matplotlib.pyplot

- Print a plot of actual and predicted accuracy using matplotlib.pyplot library.



SVM Algorithm:

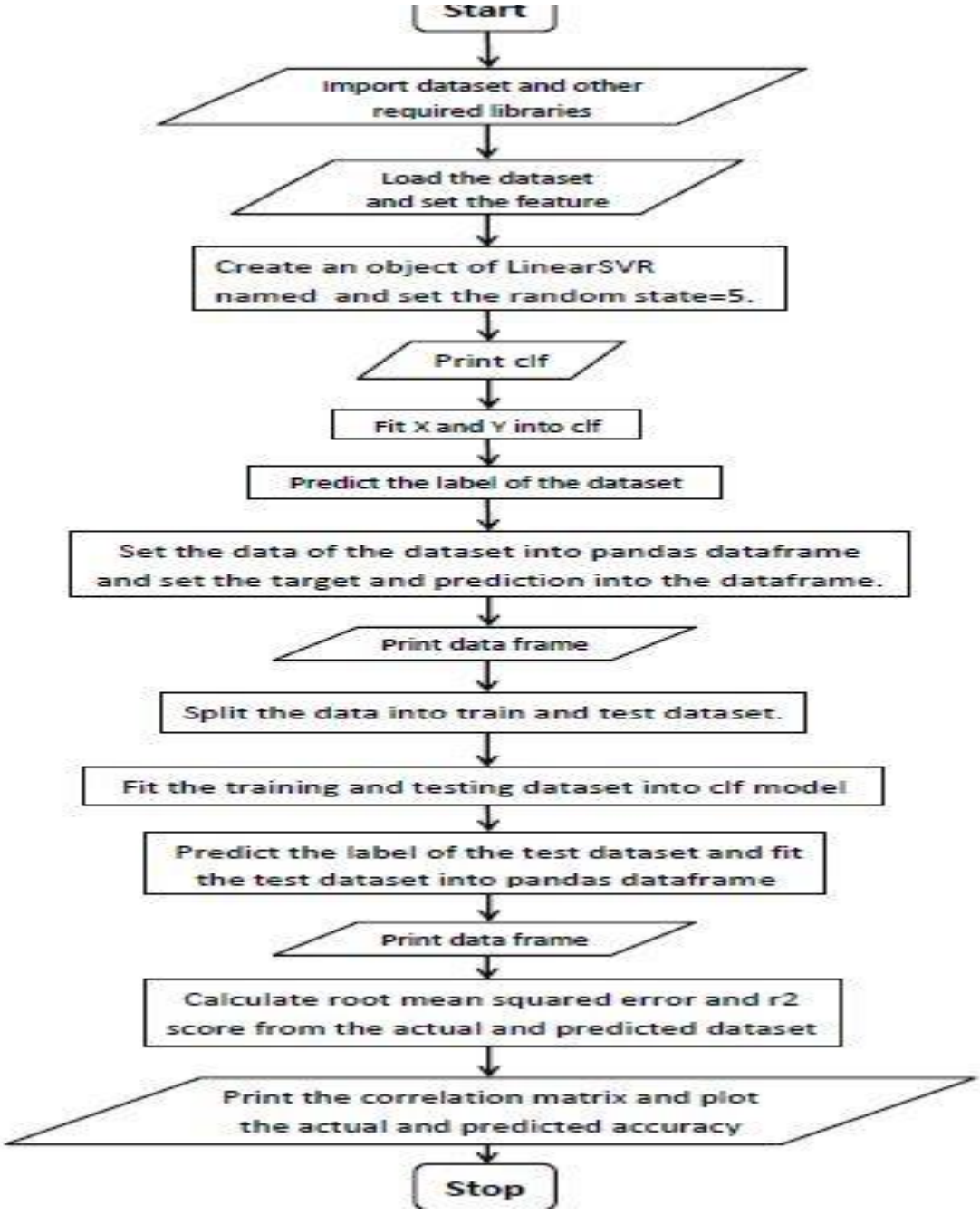
- Import sklearn.datasets, matplotlib.pyplot as plt, numpy as np.
- Load the boston dataset into a variable named boston.
- Set the features and label of the boston dataset into x and y respectively.
- Import SupportVectorRegressor(SVR) from sklearn.datasets
- Create an object of LinearSVRnamed and set the random state=5.
- Print clf.
- Fit x and y into clf.
- Predict the label of the dataset from clf model from its features.
- Import pandas as pd.
- Set the data of the dataset into pandasdataframe and set the target and prediction into the dataframe.
- Print df.
- From sklearn import model_selection.
- Split the data in 70:30 ratio into train and test dataset.
- Fit the training dataset into clf model.
- Fit the test dataset into clf model.
- Now predict the label from test dataset.
- Fit the test dataset into pandasdataframe.
- Print df.
- Import seaborn as sns.
- From sklearn.metrics import mean_squared_error, r2_score.
- Calculate root mean squared error from actual dataset and predicted dataset.
- Calculate r2 score from the same.
- Now print correlation matrix using seaborn library.
- Import matplotlib.pyplot
- Print a plot of actual and predicted accuracy using matplotlib.pyplot library.



Linear Regression Algorithm:

- Import libraries – pandas , matplotlib , numpy , seaborn , sklearn .
- Load Boston dataset from sklearn .
- Create dataframe ‘Boston’ using pandas library function .
- Insert target variable ‘MEDV’ in Boston dataframe .
- Plot graph showing relation between ‘MEDV’ and other features .
- Create correlation matrix from pandas dataframe library (round to 2 decimal places) .

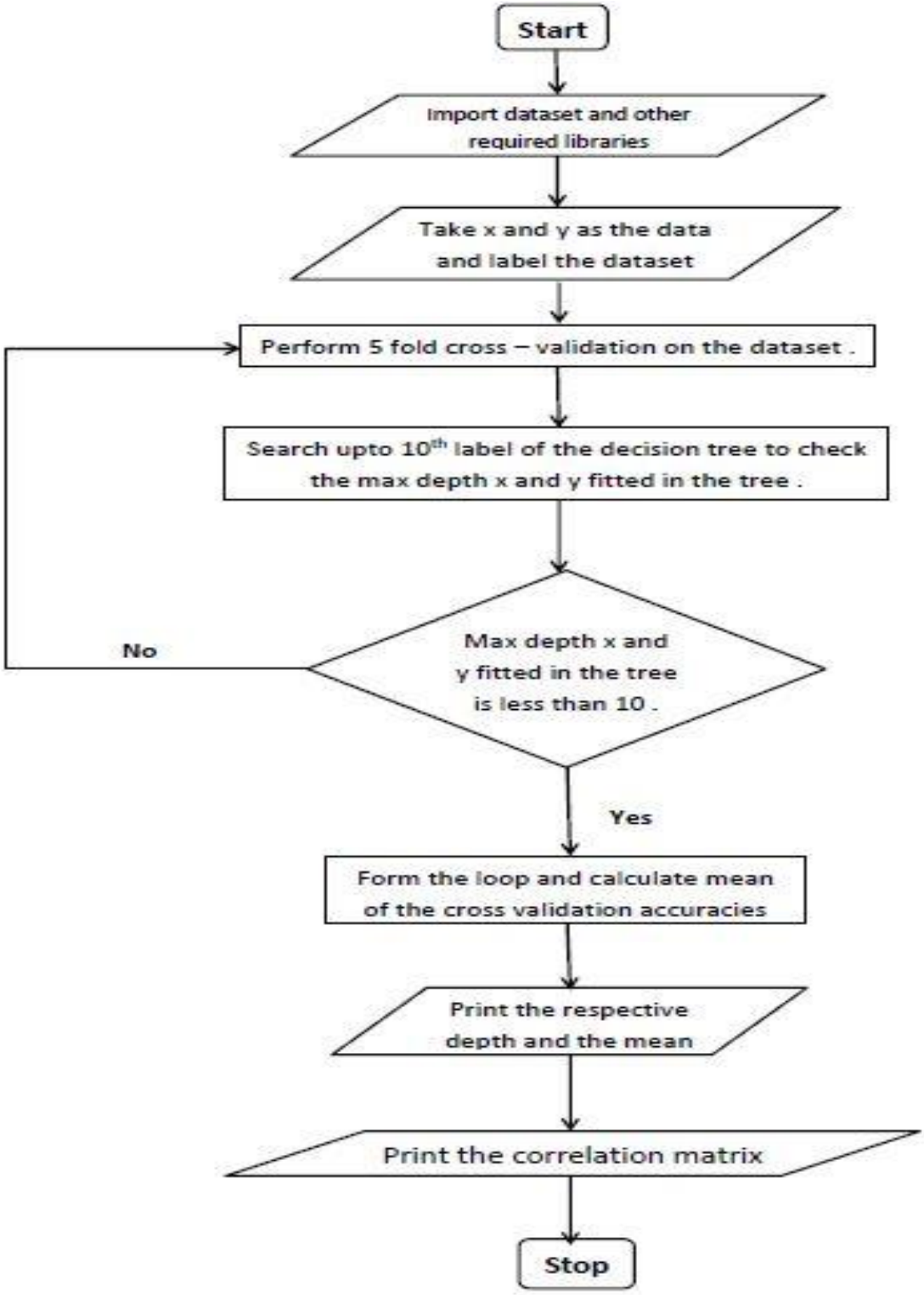
- Plot correlation matrix using seaborn heatmap function .
- Analysis of correlation matrix reveals ,
- RM have strong positive correlation with MEDV .
- LSTAT have strong negative correlation with MEDV .
- Define list features containing ‘LSTAT’ and ‘RM’ features .
- Define target containing ‘MEDV’ from dataframe Boston .
- For i , col in enumerate (features) , repeat steps 12 to 18 .
- Subplot (i , len(features) ,(i+1)) .
- Define x = Boston[col] .
- Define y = target .
- Use scatter plot function of matplotlib for values of x and y with marker = 0 .
- Label title (col)
- Define label for x – x label (col) .
- Define label for y – y label (MEDV) .
- Define new dataframe x as
    - X = pd.Dataframe(np.c_[Boston[‘LSTAT’] , Boston[‘RM’]],columns = [‘LSTAT’,’RM’]) .
- Y = Boston[‘MEDV’]
- Split the dataset using model_selection.test_train_split function of sklearn .
- Define :- x_train , x_test , y_train , y_test = train_test_split (x , y , test_size = 0.2 , random_state = 5)
- Import LinearRegression and mean_squared_error from sklearn .
- Define lin_model as
    - lin_model = LinearRegression()
    - lin_model.fit (x_train , y_train)
- Define y_train_predict = lin_model.predict (x_train) .
- Calculate root mean squared deviation as ,
    - rmse = (np.sqrt(mean_squared_error (y_train , y_train_predict)))
- Calculate r2 score as ,
    - r2 = r2_score(y_train , y_train_predict)
- Define y_test_predict = lin_model.predict(x_test)
- Calculate root mean squared deviation as ,
    - rmse = (np.sqrt(mean_squared_error(y_test , y_test_predict)))
- Calculate r2 score as ,
    - r2 = r2_score(y_test , y_test_predict)
- Print the value of RMSE and R2 Score for test and train set . Now the module is ready .



DECISION TREE:

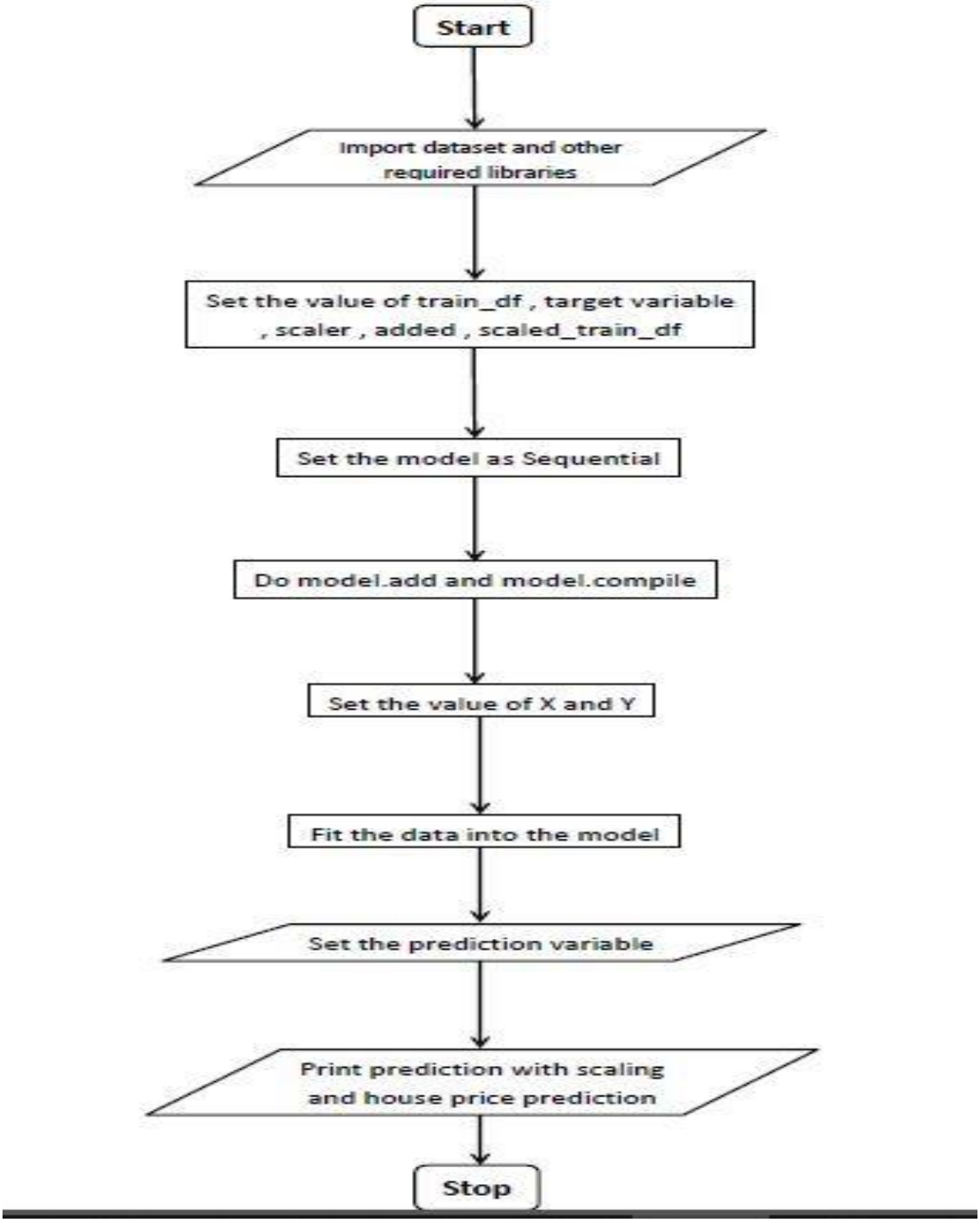
- Import boston dataset from sklearn.datasets, import numpy.
- Take x and y as the data and label of the dataset.
- Import cross_val_score from sklearn.model_selection.
- Perform 5 fold cross-validation on the dataset.
- Import tree from sklearn.
- Search upto 10^{th} label of the decision tree to check if the max depth x and y fitted in the tree is less than 10.

- If the above condition is true exit from the loop.
- Now calculate mean of the cross validation accuracies and print the respective depth and the mean.
- To get better accuracy assign the value of min_sample_split 30
- Now print the correlation matrix using seaborn library's heatmap function.



- **Artificial Neural Network:**
- Import pandas as pd , MinMaxScaler from sklearn.preprocessing , tensorflow as tf .
- Set train_df = pd.read_csv('train.csv',index_col ='ID')
- Set the target as medv .
- Set scaler = MinMaxScaler(feature_range=(0,1))
- Set scaler.fit_transform(train_df)
- Set multiplied_by as scaler.scale_[13]
- Set added = scaler.min_[13]
- Set scaled_train_df = pd.DataFrame(scaled_train , columns = train_df.columns.values)
- Import keras from tensorflow , Sequential from keras , Dense from keral.layers .
- Set the model = Sequential()
- Do model.add(Dense(50, activation = 'relu'))
  Do model.add(Dense(100, activation = 'relu'))
  Do model.add(Dense(50, activation = 'relu'))
  Do model.add(Dense(1))
- Do model.compile (loss = 'mean_squared_error' , optimization = 'adam')
- Set the value of X and Y as ,
  X = scaled_train_df.drop(target , axis = 1).values
  Y = scaled_train_df[[target]].values
- Fit the data into the model .
- Set prediction as model.predict(X[:1])
- Y_0 = prediction[0][0]
- Print the prediction with scaling .
- Y_0 -= added
- Y_0 /= multiplied_by

Print the result as house price prediction .



- **Why Naïve Bayes is not applicable:**

Naïve-Bayes assigns a probability in every possible value in the target range. The resulting distribution is then conducted into a single prediction. In categorical problems, the optimal prediction under zero-one loss is the most likely value -- the mode of underlying distribution.

However in numeric problems the mode of underlying predictions is either the mean or the median, depending on the los function. These two statistics are far more sensitive to the underlying distribution than the most likely value: they almost always change when the underlying distribution changes, even by a small amount. Therefore, when used for numeric prediction, Naïve- Bayes is more sensitive to inaccurate probability estimates than when it is used for classification.

We address the problem of predicting a numeric target value *Y* , given an example *E*. *E* consists of *m* attributes *X*1*; X*2*; ::: ; Xm*. Each attribute is either numeric, in which case it is treated as a real number, or nominal, in which case it is a set of unordered values.

If the probability density function *p.Y* j *E/* of the target value were known for all possible examples *E*, we could choose *Y* to minimize the expected prediction error. However, p(Y|E) is usually not known, and has to be estimated from data. Naive Bayes achieves this by applying Bayes' theorem and assuming independence of the attributes *X*1*; X*2*; : : : ; Xm* given the target value *Y* . Bayes' theorem states that

$$p(Y \mid E) = \frac{p(E, Y)}{\int p(E, Y)\, dY} = \frac{p(E \mid Y)p(Y)}{\int p(E \mid Y)p(Y)\, dY},$$

(1)

where the likelihood *p(E |Y)*is the probability density function (pdf) of the example *E* for a given target value *Y* , and the prior *p(Y)* is the pdf of the target value before any examples have been seen. Naive Bayes makes the key assumption that the attributes are independent given the target value, and so Eq. (1) can be written

$$p(Y \mid E) = \frac{p(X_1 \mid Y)p(X_2 \mid Y) \cdots p(X_m \mid Y)p(Y)}{\int p(X_1 \mid Y)p(X_2 \mid Y) \cdots p(X_m \mid Y)p(Y)\, dY}.$$

(2)

Instead of estimating the pdf *p(E |Y)*, the individual pdfs *p(Xi | Y)* can now be estimated separately. This dimensionality reduction makes the learning problem much easier. Because the amount of data needed to obtain an accurate estimate increases with the dimensionality of the problem, *p(Xi |Y)* can be estimated more reliably than *p(E | Y)* .However, a question remains: how harmful is the independence assumption when it is not valid for the learning problem at hand? The empirical results presented in Section 4 shed some light on this issue. The following sections discuss how *p(Xi |Y)* and *p(Y)* can be estimated from a set of training

examples. For *p(Xi | Y)* there are two cases to consider: the case where attribute *Xi* is numeric, and the case where it is nominal.

We first discuss the estimation of *p(X | Y)* for numeric attributes *X*, where we assume that they are normalized by their range, as determined using the training data. In this case, *p(X | Y)* is a pdf involving two numeric variables. Because

$$p(X \mid Y) = \frac{p(X, Y)}{\int p(X, Y)\, dX}, \tag{3}$$

the conditional probability *p(X |Y)* can be estimated by computing an approximation to the joint probability *p(X|Y)*. In principle, any estimator for two-dimensional pdfs can be used to model *p(X| Y)*, for example, mixture models (Ghahramani & Jordan, 1994). We have chosen the kernel density estimator

$$\hat{p}(X = x, Y = y) = \frac{1}{n h_X h_Y} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h_X}\right) K\left(\frac{y - y_i}{h_Y}\right), \tag{4}$$

where *xi* is the attribute value, *yi* the target value of training example *i* , *K.:/* is a given kernel function, and *hX* and *hY* are the kernel widths for *X* and *Y* . If either *xi* or *yi* is missing, the example is not included in the calculation. It can be shown that this estimate converges to the true pdf if the kernel function obeys certain smoothness properties and the kernel widths are chosen appropriately

Acommon choice for *K.:/* is the Gaussian kernel $K(t)=(2\pi)^{-1/2}e^{-t2/2}$, and this is what we use in our experiments. Ideally, the kernel widths *hX* and *hY* should be chosen so that the difference between the estimated pdf *p(X, Y)* and the true pdf *p(X, Y)* is minimized. One way of measuring this difference is the expected cross-entropy between the two pdfs, an unbiased estimate of which can be obtained by leave-one-out cross-validation .

$$CV_{CE} = -\frac{1}{n} \sum_{j=1}^{n} \log\left(\frac{1}{(n-1) h_X h_Y} \sum_{i=1, i \neq j}^{n} K\left(\frac{x_j - x_i}{h_X}\right) K\left(\frac{y_j - y_i}{h_Y}\right)\right) \tag{5}$$

Then, *hX* and *hY* are set to $hX=cX/\sqrt{n}$ *and* $hY=cY//\sqrt{n}$; where *cX* and *cY* are chosen to minimize the estimated cross-entropy. In our experiments, we use a grid search for *(cX ,cY) belongs to* [0*:*4*;* 0*:*8] X [0*:*4*;* 0*:*8] with a grid width of 0.1. We have tried other parameter settings for the search and found little difference in the result. Because

$$\int \hat{p}(X, Y)\, dX = \frac{1}{n h_Y} \sum_{i=1}^{n} K\left(\frac{y - y_i}{h_Y}\right), \tag{6}$$

we have all the terms needed to compute an estimate O *p.X* j *Y* / of *p.X* j *Y* / for a numeric attribute *X*. Nowconsider the case where *p.X* j *Y* / has to be estimated for a nominal attribute *X* with a set of unordered values *v*1*; v*2*; : : : ; v*o. Rather than estimating *p(X*| *Y)* directly, we transform the problem into one of estimating *p.Y* j *X/*, the pdf of a numeric variable given a nominal one, and *p.X/*, the prior probability of a nominal attribute value. This transformation is effected by applying Bayes' theorem:

$$p(X = v_k \mid Y = y) = \frac{p(X = v_k) p(Y = y \mid X = v_k)}{\sum_{k=1}^{o} p(X = v_k) p(Y = y \mid X = v_k)} \tag{7}$$

Both *p.Y* j *X/* and *p.X/* can be estimated easily. For the former,we use the one-dimensional counterpart of the kernel density estimator from above:

$$\hat{p}(Y = y \mid X = v_k) = \frac{1}{n_k h_k} \sum_{i=1}^{n_k} K\left(\frac{y - y_i}{h_k}\right), \tag{8}$$

where the sum is over all *nk* examples with attribute value *X* D *vk* . This type of estimator was also used by John and Langley (1995) to estimate the density of numeric attributes in naive Bayes for categorical prediction. For the latter, an estimate of *p.X/* is obtained simply by computing the proportion of examples with attribute value

$$\hat{p}(X = v_k) = \frac{n_k}{\sum_{l=1}^{o} n_l} \tag{9}$$

Again, the cross-validation procedure can be used to choose *hk* D *ck*=
p
nk for each *vk* so that the estimated cross-entropy is minimized. Here we use a grid search for *ck* 2 [0*:*4*;* 0*:*8] with a grid width of 0.1. This gives all the terms necessary to compute an estimate O *p.X* j *Y* / of *p.X* j *Y* / for a nominal attribute *X*.

The procedure for estimating the prior *p.Y* / is the same as for *p.Y* j *X/* (Eq. (8)). The only difference is that, instead of including only examples with a particular attribute value, all *n* examples are used. The kernel width is chosen using the previously described cross-validation procedure.

The optimal prediction *t .e/* for an example *e* with respect to the posterior probability *p.Y* j *E* D *e/* depends on the loss function. We consider two loss functions: the squared error and the absolute error. In either case, the predicted value should minimize the expected loss. It is easy to show that the expected squared error:

$$t_{SE}(e) = \int p(Y = y \mid E = e) y \, dy. \tag{11}$$

is minimized if the expected value of *y* (that is, its mean) is predicted:
An estimate O*tSE.e/* of this quantity can be obtained from O *p.Y* D *y; E* D *e/*. Let *G* be a set of equally spaced grid points in the domain of *y*. Suppose *y*min is the minimum and *y*max the maximum value for *y* in the training data, and let *h* D *.y*max ¡ *y*min*/=.d* ¡ 1*/* for some number of grid points *d*. Then we use *G* D f*: : : ; y*min ¡ 2*h; y*min ¡ *h; y*min*; y*min C *h; : : : ; y*max ¡ *h; y*max*; y*max C *h; y*max C 2*h; : : :*g. Note that one can stop evaluating points to the left of *y*min and to the right of *y*max as soon as O *p.Y* D *y; E* D *e/* becomes negligible.

In our experiments, *d* is set to 50 and attributes *Xi* for which O *p.Xi* j *Y* D *y/* is negligible for all values in *G* are excluded from the computation of O *p.Y* D *y; E* D *e/*. Then,

$$\hat{t}_{SE}(e) = \frac{\sum_{y \in G} \hat{p}(Y = y, E = e) y}{\sum_{y \in G} \hat{p}(Y = y, E = e)}. \tag{12}$$

The sum in the denominator approximates the integral in the denominator of Eq. (2).
If, on the other hand, the expected absolute error

$$E[|t(e) - y|] = \int p(Y = y \mid E = e) |t(e) - y| \, dy \tag{13}$$

is to be minimized, this is achieved by setting *tAE.e/* so that

$$\int_{-\infty}^{t_{AE}(e)} p(Y = y \mid E = e) \, dy = 0.5. \tag{14}$$

In this case, the optimum prediction *tAE.e/* is the median . Again, an

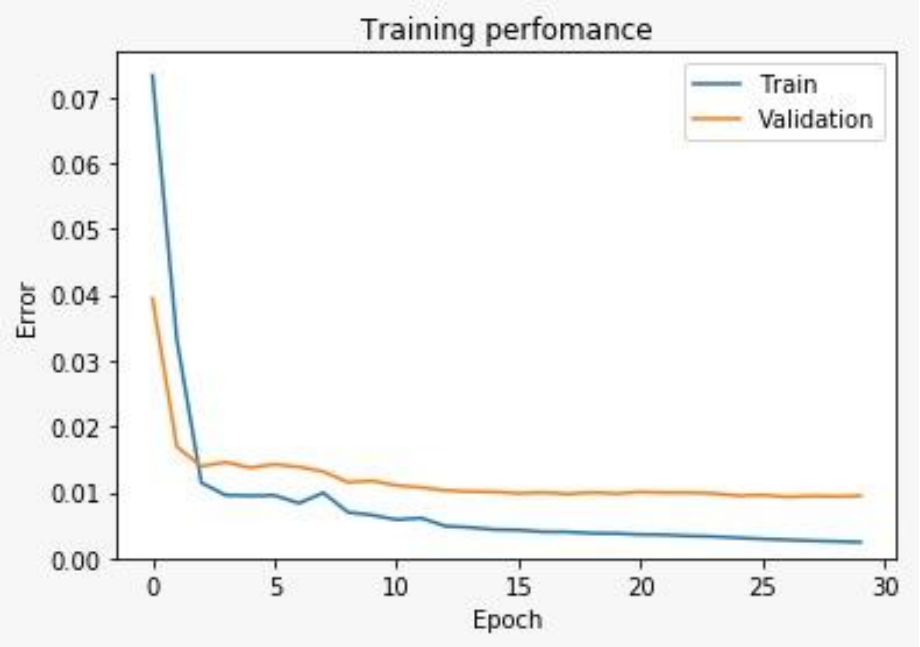
estimate O*tAE.e/* can be obtained using O *p.Y* D *y; E* D*e/* by finding the smallest *y*0 for which

$$\frac{\sum_{y\in G, y<y'} \hat{p}(Y=y, E=e)}{\sum_{y\in G} \hat{p}(Y=y, E=e)} > 0.5, \qquad (15)$$

and setting *tAE.e/* D *y*0. For estimating *tAE* we use *d* D 100 grid points.

Comparative data Analysis:

We have implemented five algorithms on boston data set. They are KNN(k-nearest neighbors), SVM(support vector machine), LR(linear regression), ANN(Artificial Neural Network) and Decision Tree. Their accuracies are 71.75%(for linear regression), 49.28% (for svm), 73.66% (for knn), 83.08%(for ANN) and 79.51% (for decision tree).And error of Artificial Neural Network is 0.18%.So we can conclude that Artificial Neural Network is the best among it's counterparts on boston dataset and svm is the worst.



Training Graph Performance

CONCLUSION:

In this project, we have learned the basic process of analyzing a dataset to implement various types of Machine Learning models for the purpose of data classification and unknown sample prediction. We have understood the outline of basic architecture of Applied Machine Learning Process and applied them on the classic Boston housing dataset. We described how powerful tools like Python and its libraries can help us to get quickly to the results while leaving us the freedom to get more complicated.

FUTURE SCOPE &FURTHER ENHANCEMENT OF THE PROJECT:

Machine Learning can be a competitive advantage to any company be it a top MNC or a startup as things that are currently being done manually will be done tomorrow by machines. Machine Learning revolution will stay with us for long and so will be the future of Machine Learning.

REFERENCES:

- **Websites :**
  - https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/
  - https://www.ritchieng.com/machine-learning-project-boston-home-prices/
  - https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72
  - https://machinelearningmastery.com/linear-regression-for-machine-learning/
  - https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/ml-decision-tree/tutorial/
  - https://www.geeksforgeeks.org/naive-bayes-classifiers/