RAJIV GANDHI INSTITUTE OF PETROLEUM TECHNOLOGY

विधारत्नम् महद्धनम्

APPLICATION OF MACHINE LEARNING MODELS TO STUDY ON EQUIVALENT CIRCULATING DENSITY IN WELLBORE

B.TECH

PROJECT REPORT

MD ALTAMASH MAZHARI (EPE18023)

PRATHAM CHOUDHARY (EPE18030)

MENTOR: DR. AMIT SAXENA

CONTENTS

ACKNOWLEDGEMENT

It is with immense pleasure that we express our sincere sense of gratitude and humble appreciation to Dr. Amit Saxena Sir for his invaluable guidance, whole-hearted cooperation, constructive criticism and continuous encouragement in the preparation of this project. Without their support and guidance, the present work would have remained a dream.

We would also like to thanks Mr. Romy Agarwal, PhD. Candidate at IIT(ISM) Dhanbad for providing us data and sharing his valuable insights which proved to be very valuable while preparing this project.

We also take this opportunity to thank all our batchmates and family for their valuable support and encouragement throughout the preparation of this work. We also thank all those who have directly or indirectly helped in completion of this work.

- **Md Altamash Mazhari**

Pratham Choudhary

INTRODUCTION

ECD (equivalent circulating density) is the dynamic density exerted by a circulating mud at any point in the borehole. The equivalent circulating density or dynamic density at any time during fluid flow is always greater than the static mud density when flow is stopped. It is a very important parameter in drilling high-temperature and deep-water wells. It is considered as a key parameter while drilling through formations where there is a very thin line between pore pressure and the fracture pressure. In these conditions, it is used to control the formation pressure and hence prevent the occurrence of blowout.

In the oil and gas industry, downhole tools are used to measure and monitor changes of ECD to avoid well control issues such as gas kicks, blowout, and formation fracturing. The main tools used now are measurement while drilling (MWD) and pressure while drilling (PWD). These tools contain pressure sensors that can independently measure the bottomhole pressure of the well during drilling, regardless of the factors controlling the ECD. In addition to the expensive daily rates of such tools, there are some operating limitations for its application such as pressure, temperature, and tool failures.

The objective of our project is to use different machine learning techniques to predict the ECD values and then finding out the most suitable machine learning algorithm for ECD value prediction. The metrics used for comparing different algorithms is the Coefficient of Determination (R^2).

Intelligent systems and machine learning have proven their effectiveness in solving complicated problems that cannot be solved analytically. With the large historical drilling data available in the oil and gas industry, machine learning and intelligent systems can be used to make better future decisions that will help to optimize the drilling operations.

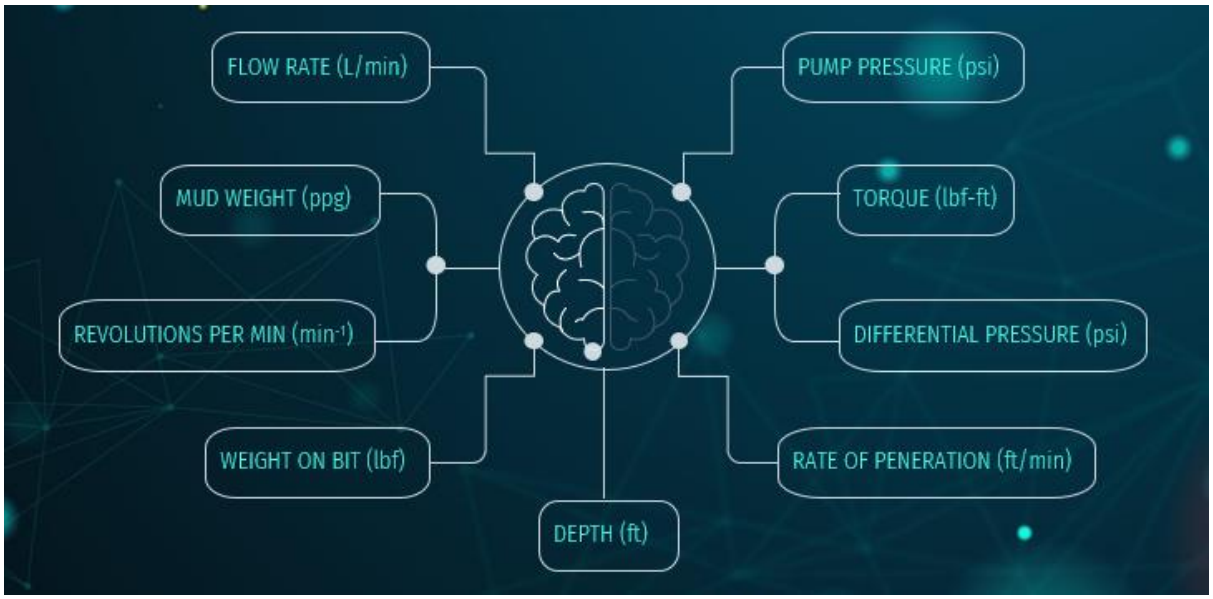
ROADMAP TO PREDICTING ECD

INPUT PARAMETERS

The data from MARATHON OIL EF LLC operating at GUAJILLO UNIT 8 SOUTH 5H situated at Atascosa County Texas, USA was taken. The total data points obtained in the field dataset were 30,195. Out of these 16,298 points were finally left after cleaning the data. The operations performed on data were:

1.Deleting Null data rows
2. Grouping data within 1 ft depth interval by averaging data

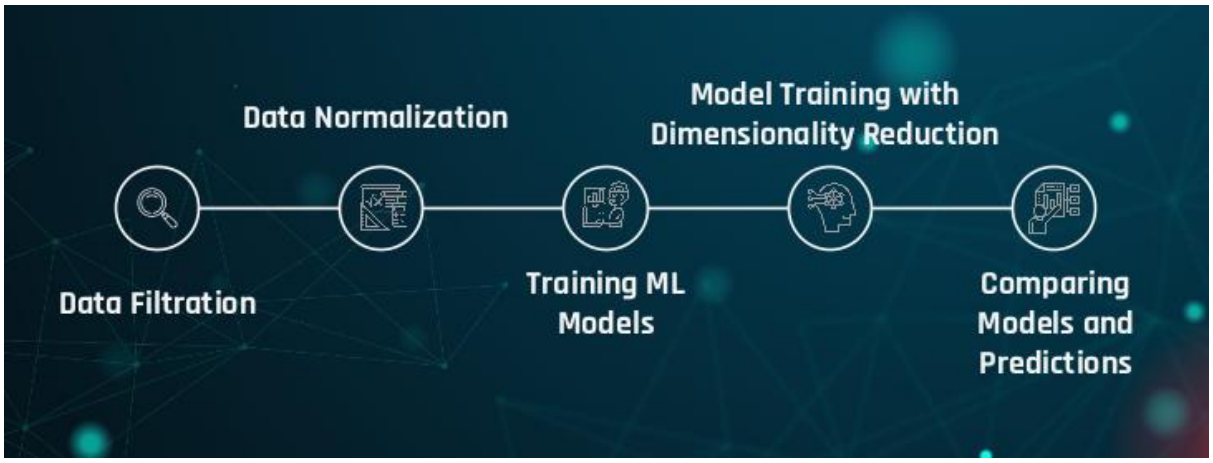
A total of 9 surface parameters were chosen as input parameters for training the regression models. Surface parameters were chosen as besides their data being easily available, they are least dependent on high pressure and high temperature conditions.



These parameters are called surface parameters as they can be obtained on the surface and not through the downhole sensors. Therefore, they are independent of high-pressure and high-temperature conditions.

METHODOLOGY

The following attached image shows how have we proceeded with our methodology starting with the filtration of the provided data, scaling the data with normalisation and then applying the Machine Learning Models to obtain our results.



1. Data Filtration

To have an accurate prediction using machine learning techniques, data must be filtered. Filtration process starts with removing all random values that cannot represent the measurements such as negative values, 999 values and null ones.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
surface = pd.read_csv('Guajillo_SurfaceData.csv')
pump = pd.read_csv('Guajillo_PumpData.csv')
d_pump = pump['Depth Index( ft )'].apply(lambda x: np.ceil(x))
d_surface = surface['Depth (ft)'].apply(lambda x: np.ceil(x))
surface['Depth (ft)'] = d_surface
pump['Depth Index( ft )'] = d_pump
surface.drop(['Well Name', 'Rig State'], axis=1, inplace=True)
pump.drop(['MudFlowInAvg (gpm )', 'ROPAvg (ft/hr )', 'RPMA (rpm )', 'WeightonBitMax (klb
)',
      'SurfaceTorqueMax (ft-lb )'], axis=1, inplace=True)
surface = surface.rename({'Depth (ft)': 'Depth', 'ROP (ft/hr)': 'ROP', 'WOB (klb)': 'WOB', 'Diff
Press (psi)': 'DelP',
             'Torque (ft-lb)': 'Torque', 'RPM (rpm)': 'RPM', 'Flow In (gpm)': 'Flow Rate'}, axis=1)
pump = pump.rename({'Depth Index( ft )': 'Depth', 'PumpPressure (psi )': 'Pump Pressure',
                'MudDensityInAvg (ppg )': 'Mud Weight'}, axis=1)
data = pd.merge(surface, pump, how = 'outer')
data = data.dropna()
data = data.groupby('Depth').mean()
data = data.reset_index()
```

```
data.head()
```
The final dataframe obtained is as follows:

	Depth	ROP	WOB	DelP	Torque	RPM	Flow Rate	ECD	Mud Weight	Pump Pressure
0	36.0	918.4	0.2	27.6	454.9	19.8	522.9	14.5	10.0	663.4
1	38.0	1585.2	0.1	28.5	362.7	19.8	522.9	15.6	10.0	648.6
2	40.0	1558.7	0.2	7.1	173.1	19.8	522.9	15.6	10.0	644.8
3	42.0	1307.0	0.3	17.4	310.3	19.8	522.9	15.2	10.0	620.6
4	44.0	1567.5	0.2	20.8	400.3	19.8	521.7	15.6	10.0	643.0

2. Normalisation of Data

The Data available to us is of wide range ranging from tens in density to thousands in depth. Therefore, it is very important to normalise the data between 0 to 1 before applying ML models. A properly designed and well-functioning database should undergo data normalization to be used successfully. Data normalization gets rid of several anomalies that can make analysis of the data more complicated. Some of those anomalies can crop up from deleting data, inserting more information, or updating existing information. Once those errors are worked out and removed from the system, further benefits can be gained through other uses of the data and data analytics. In Python we use MinMaxScaler function to normalise data and a sample of the code is attached below:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
Normalized_data = scaler.fit_transform(data)
Normdata = pd.DataFrame(Normalized_data, columns = data.columns)
```

3. Principal Component Analysis

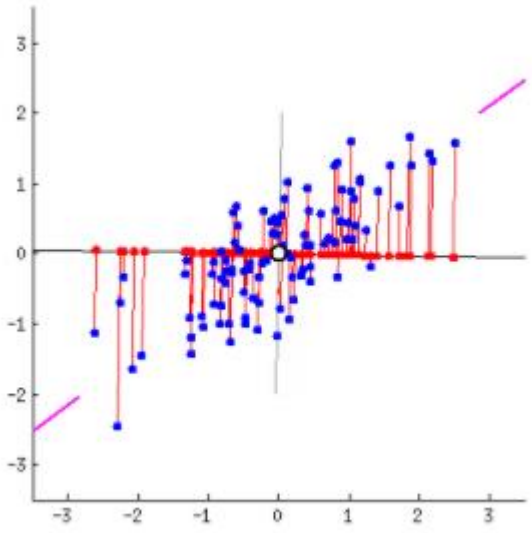
Principal Component Analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analysing data much easier and faster for machine learning algorithms without extraneous variables to process.

In short, the idea of PCA is simple — reduce the number of variables of a data set, while preserving as much information as possible.

HOW DOES PCA CONSTRUCT THE PRINCIPAL COMPONENTS?

As there are as many principal components as there are variables in the data, principal components are constructed in such a manner that the first principal component accounts for the largest possible variance in the data set.



For example, let's assume that the scatter plot of our data set is as shown below, can we guess the first principal component? Yes, it's approximately the line that matches the purple marks because it goes through the origin and it's the line in which the projection of the points (red dots) is the most spread out. Or mathematically speaking, it's the line that maximizes the variance (the average of the squared distances from the projected points (red dots) to the origin).

The second principal component is calculated in the same way, with the condition that it is uncorrelated with (i.e., perpendicular to) the first principal component and that it accounts for the next highest variance.

This continues until a total of p principal components have been calculated, equal to the original number of variables.

Principal components are new variables that are constructed as linear combinations or mixtures of the initial variables. These combinations are done in such a way that the new variables (i.e., principal components) are uncorrelated and most of the information within the initial variables is squeezed or compressed into the first components. So, the idea is 10-dimensional data gives you 10 principal components, but PCA tries to put maximum possible information in the first component, then maximum remaining information in the second and so on.

```
from sklearn.decomposition import PCA
pca5 = PCA(n_components=5)
pca5 = pca5.fit_transform(X)
X5 = pd.DataFrame(pca5, columns=['1', '2','3','4','5']
```

4. Applying Machine Learning Models

In our project after normalising the data we have applied the Machine Learning Regression Models while simultaneously applying dimensionality reduction using PCA to compare the accuracy scores.

Regression Models

Regression analysis is a form of predictive modelling technique which investigates the relationship between a dependent (target) and independent variable (s) (predictor). This technique is used for forecasting, time series modelling and finding the causal effect relationship between the variables.

Regression analysis is an important tool for modelling and analyzing data. Here, we fit a curve / line to the data points, in such a manner that the differences between the distances of data points from the curve or line is minimized.

The regression models used in our projects are:

1. Decision Tree
2. Random Forest Regressor
3. Elastic Net Regressor
4. XGBoost Regressor
5. Support Vector Machine
6. K-Nearest Neighbour

For all the models we have used python jupyter notebook and used scikitlearn library in which all the regression models were in build.

1. Decision Tree Regressor

Decision Trees can be summarized with the below bullet points:

- Decision trees are predictive models that use a set of binary rules to calculate a target value.
- Each individual tree is a simple model that has branches, nodes and leaves.

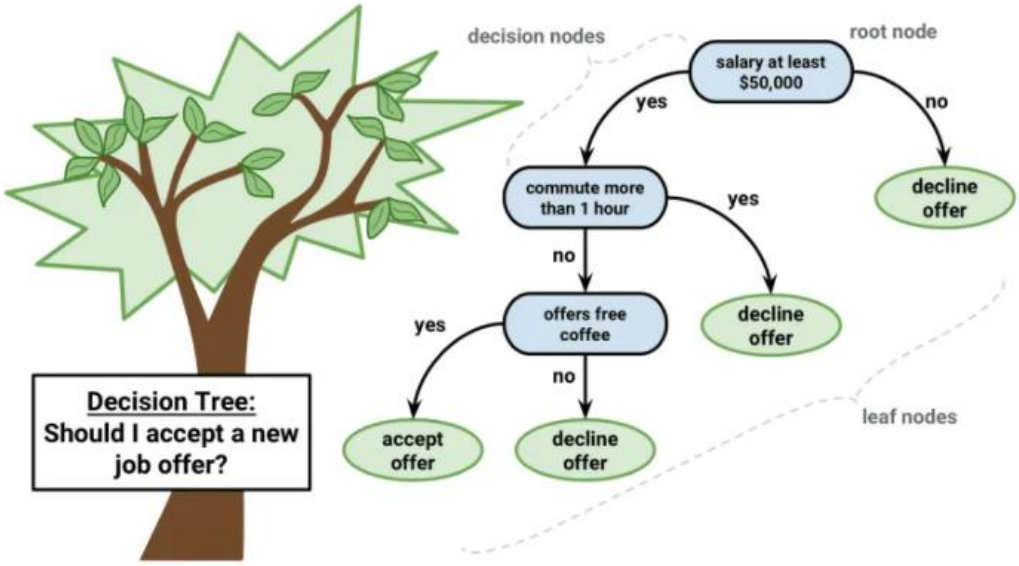
Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, decision tree algorithm can be used for solving regression and classification problems too.

The general motive of using Decision Tree is to create a training model which can use to predict class or value of target variables by learning decision rules inferred from prior training data.

The understanding level of Decision Trees algorithm is so easy compared with other classification algorithms. The decision tree algorithm tries to solve the problem, by using tree representation. Each internal node of the tree corresponds to an attribute, and each leaf node corresponds to a class label.

Decision Tree Algorithm Pseudocode

1. Place the best attribute of the dataset at the root of the tree.
2. Split the training set into subsets. Subsets should be made in such a way that each subset contains data with the same value for an attribute.
3. Repeat step 1 and step 2 on each subset until you find leaf nodes in all the branches of the tree.



Decision Tree classifier, **Image credit:** www.packtpub.com

In decision trees, for predicting a class label for a record we start from the root of the tree. We compare the values of the root attribute with record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

We continue comparing our record's attribute values with other internal nodes of the tree until we reach a leaf node with predicted class value. As we know how the modelled decision tree can be used to predict the target class or the value. Now let's understanding how we can create the decision tree model.

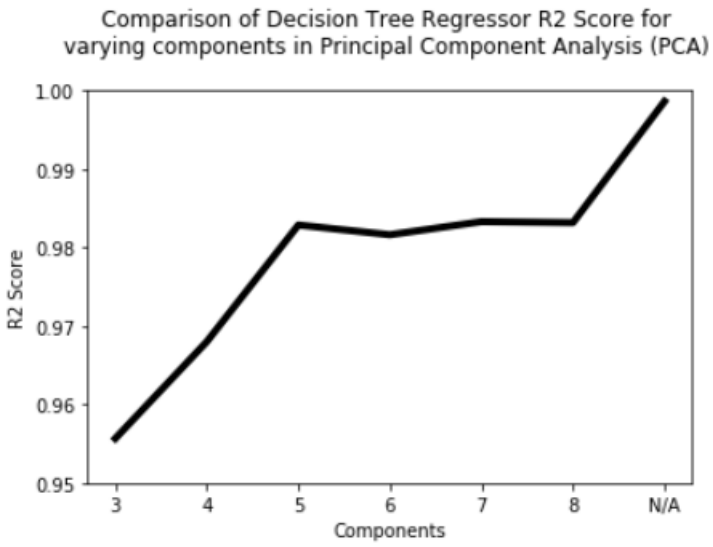
For our Model:

We imported the file by sklearn library in python then again trained on 70% data and tested on 30% data and got the results of following accuracy scores after applying PCA:

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score
DTreg = DecisionTreeRegressor(random_state=101)rfr.fit(X_train, y_train)
R21 = DTreg.score(X_test,y_test)
comp = [3, 4, 5, 6, 7, 8]
R2_Scores = []
for c in comp:
    X = Normdata[['Depth', 'ROP', 'WOB', 'DelP', 'Torque', 'RPM', 'Flow Rate',
      'Mud Weight', 'Pump Pressure']]
    y = Normdata['ECD']
    pca = PCA(n_components = c)
    pca = pca.fit_transform(X)
    X = pd.DataFrame(pca)
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75, random_state=101)
    rfr.fit(X_train,y_train)
    R2 = DTreg.score(X_test,y_test)
    R2_Scores.append(R2)
comp.append('N/A')
R2_Scores.append(R21)
R2_Scores
```

The accuracy scores we got were:

R-squared (Dim=3): 0.9557
R-squared (Dim=4): 0.9680
R-squared (Dim=5): 0.9829
R-squared (Dim=6): 0.9816
R-squared (Dim=7): 0.9833
R-squared (Dim=8): 0.9832
R-squared (None): 0.9832



2. Random Forest Regressor:

Random Forest is an ensemble machine learning technique capable of performing both regression and classification tasks using multiple decision trees and a statistical technique called bagging. Bagging along with boosting are two of the most popular ensemble techniques which aim to tackle high variance and high bias. A RF instead of just averaging the prediction of trees it uses two key concepts that give it the name *random*:

3. Random sampling of training observations when building trees
4. Random subsets of features for splitting nodes

In other words, Random forest builds multiple decision trees and merge their predictions together to get a more accurate and stable prediction

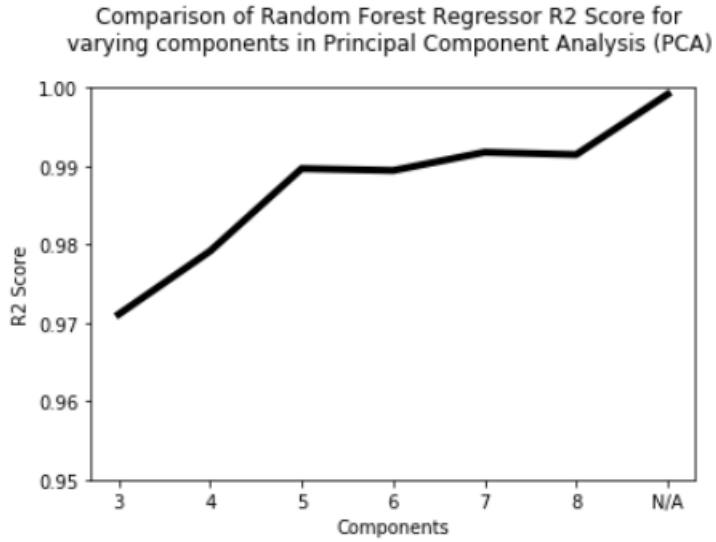
For our model:

```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor()
rfr.fit(X_train, y_train)
R21 = rfr.score(X_test,y_test)
comp = [3, 4, 5, 6, 7, 8]
R2_Scores = []
for c in comp:
    X = Normdata[['Depth', 'ROP', 'WOB', 'DelP', 'Torque', 'RPM', 'Flow Rate',
      'Mud Weight', 'Pump Pressure']]
    y = Normdata['ECD']
    pca = PCA(n_components = c)
    pca = pca.fit_transform(X)
```

```
    X = pd.DataFrame(pca)
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75, random_state=101)
    rfr.fit(X_train,y_train)
    R2 = rfr.score(X_test,y_test)
    R2_Scores.append(R2)
comp.append('N/A')
R2_Scores.append(R21)
R2_Scores
```

The accuracy scores we got were:

R-squared (Dim=3): 0.9711
R-squared (Dim=4): 0.9792
R-squared (Dim=5): 0.9897
R-squared (Dim=6): 0.9894
R-squared (Dim=7): 0.9918
R-squared (Dim=8): 0.9915
R-squared (None): 0.9992



3. Elastic Net regression model:

A problem with linear regression is that estimated coefficients of the model can become large, making the model sensitive to inputs and possibly unstable. This is particularly true for problems with few observations (*samples*) or more samples (*n*) than input predictors (*p*) or variables (so-called *p >> n problems*). One approach to addressing the stability of regression models is to change the loss function to include additional costs for a model that has large coefficients. Linear regression models that use these modified loss functions during training are referred to collectively as penalized linear regression.

One popular penalty is to penalize a model based on the sum of the squared coefficient values. This is called an L2 penalty. An L2 penalty minimizes the

size of all coefficients, although it prevents any coefficients from being removed from the model.

- l2_penalty = sum j=0 to p beta_j^2
  Another popular penalty is to penalize a model based on the sum of the absolute coefficient values. This is called the L1 penalty. An L1 penalty minimizes the size of all coefficients and allows some coefficients to be minimized to the value zero, which removes the predictor from the model.

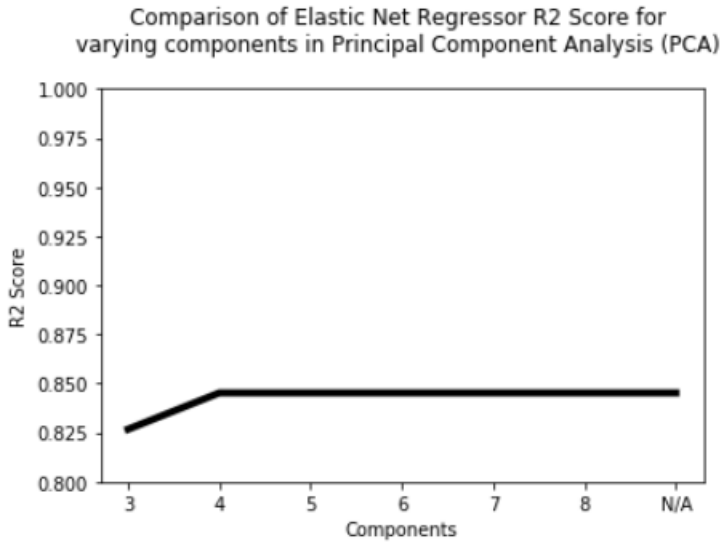
- l1_penalty = sum j=0 to p abs(beta_j)
  Elastic net is a penalized linear regression model that includes both the L1 and L2 penalties during training.

  Using the terminology from "The Elements of Statistical Learning," a hyperparameter "*alpha*" is provided to assign how much weight is given to each of the L1 and L2 penalties. Alpha is a value between 0 and 1 and is used to weight the contribution of the L1 penalty and one minus the alpha value is used to weight the L2 penalty.
- elastic_net_penalty = (alpha * l1_penalty) + ((1 – alpha) * l2_penalty)
  For example, an alpha of 0.5 would provide a 50 percent contribution of each penalty to the loss function. An alpha value of 0 gives all weight to the L2 penalty and a value of 1 gives all weight to the L1 penalty.

  For our model:

```
from sklearn.linear_model import ElasticNet
ereg = ElasticNet(alpha=0.01)rfr.fit(X_train, y_train)
ereg.fit(X_train,y_train)
R21 = ereg.score(X_test,y_test)
comp = [3, 4, 5, 6, 7, 8]
R2_Scores = []
for c in comp:
    X = Normdata[['Depth', 'ROP', 'WOB', 'DelP', 'Torque', 'RPM', 'Flow Rate',
      'Mud Weight', 'Pump Pressure']]
    y = Normdata['ECD']
    pca = PCA(n_components = c)
    pca = pca.fit_transform(X)
    X = pd.DataFrame(pca)
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75, random_state=101)
    ereg.fit(X_train,y_train)
    R2 = ereg.score(X_test,y_test)
    R2_Scores.append(R2)
comp.append('N/A')
R2_Scores.append(R21)
```

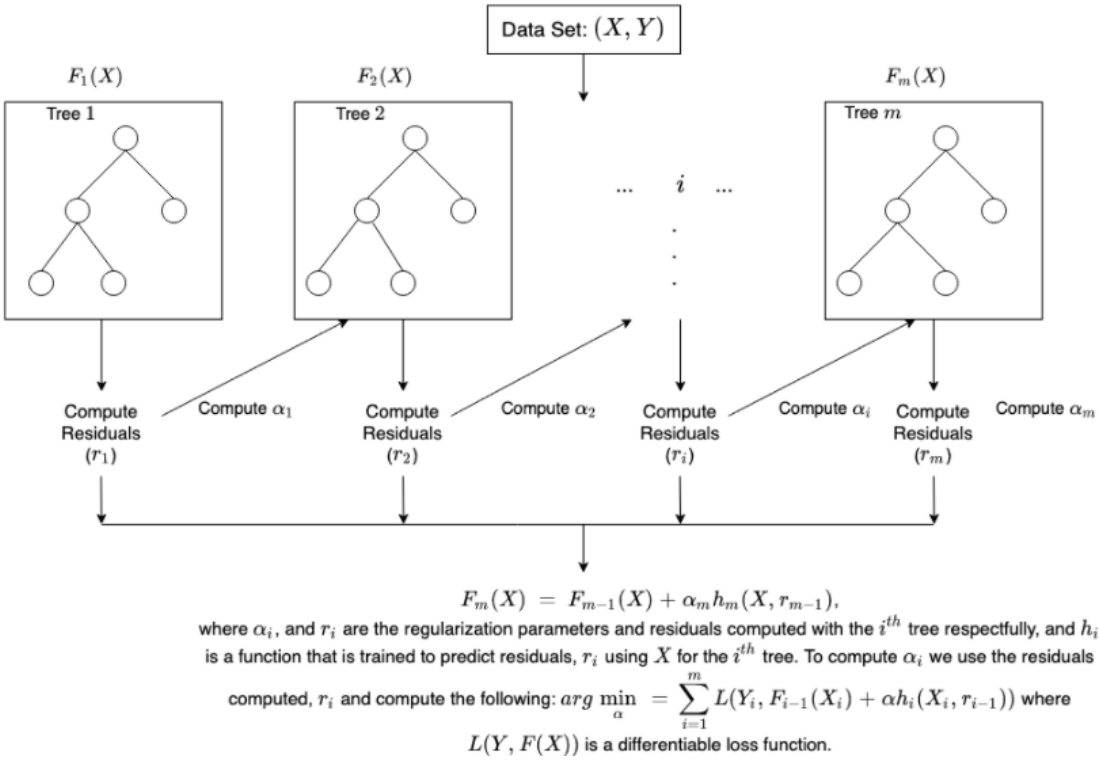
The accuracy scores we got were:
R-squared (Dim=3): 0.8267
R-squared (Dim=4): 0.8452
R-squared (Dim=5): 0.8452
R-squared (Dim=6): 0.8452
R-squared (Dim=7): 0.8452
R-squared (Dim=8): 0.8452
R-squared (None): 0.8452



4. XGBoost regressor:

XGBoost is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm, which attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models.
When using gradient boosting for regression, the weak learners are regression trees, and each regression tree maps an input data point to one of its leafs that contains a continuous score. XGBoost minimizes a regularized (L1 and L2) objective function that combines a convex loss function (based on the difference between the predicted and target outputs) and a penalty term for model complexity (in other words, the regression tree functions). The training proceeds iteratively, adding new trees that predict the residuals or errors of prior trees that are then combined with previous trees to make the final prediction. It's called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.

Below is a brief illustration on how gradient tree bosting works-

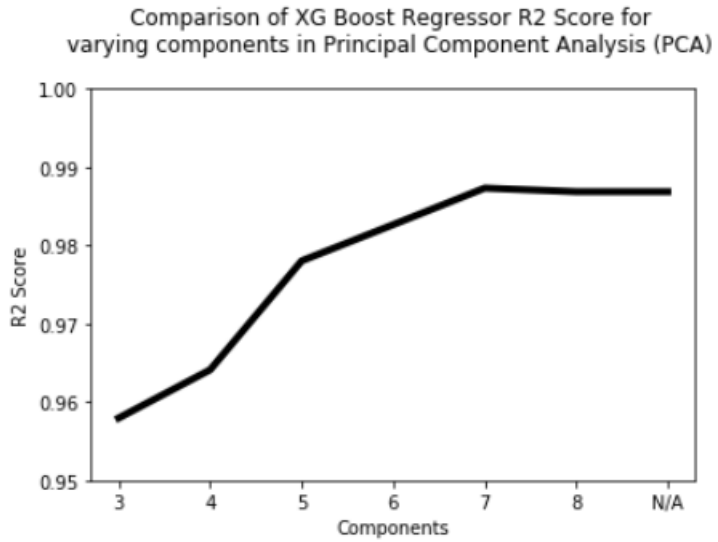


For our model:

Our test size was 30% of all the data points in all the models, the code sample is attached below :

```
import xgboost as xgb
xgreg = xgb.XGBRegressor(objective ='reg:linear', n_estimators = 10, seed = 123)
xgreg.fit(X_train,y_train)
R21 = clf.score(X_test,y_test)
comp = [3, 4, 5, 6, 7, 8]
R2_Scores = []
for c in comp:
    X = Normdata[['Depth', 'ROP', 'WOB', 'DelP', 'Torque', 'RPM', 'Flow Rate',
      'Mud Weight', 'Pump Pressure']]
    y = Normdata['ECD']
    pca = PCA(n_components = c)
    pca = pca.fit_transform(X)
    X = pd.DataFrame(pca)
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75, random_state=101)
    xgreg.fit(X_train,y_train)
    R2 = DTreg.score(X_test,y_test)
    R2_Scores.append(R2)
comp.append('N/A')
R2_Scores.append(R21)
R2_Scores
```

The accuracy scores we got were:
R-squared (Dim=3): 0.9579
R-squared (Dim=4): 0.9641
R-squared (Dim=5): 0.9780
R-squared (Dim=6):0.9827
R-squared (Dim=7): 0.9873
R-squared (Dim=8): 0.9869
R-squared (None): 0.9869

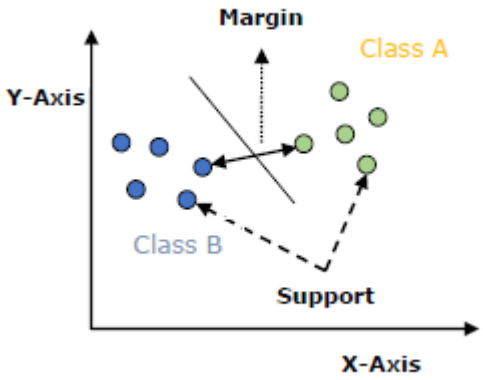


5. Support vector machine regressor:

Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression. But generally, they are used in classification problems. In 1960s, SVMs were first introduced but later they got refined in 1990. SVMs have their unique way of implementation as compared to other machine learning algorithms. Lately, they are extremely popular because of their ability to handle multiple continuous and categorical variables.

Working of SVM

An SVM model is basically a representation of different classes in a hyperplane in multidimensional space. The hyperplane will be generated in an iterative manner by SVM so that the error can be minimized. The goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH).



The followings are important concepts in SVM −

- Support Vectors − Datapoints that are closest to the hyperplane is called support vectors. Separating line will be defined with the help of these data points.
- Hyperplane − As we can see in the above diagram, it is a decision plane or space which is divided between a set of objects having different classes.
- Margin − It may be defined as the gap between two lines on the closet data points of different classes. It can be calculated as the perpendicular distance from the line to the support vectors. Large margin is considered as a good margin and small margin is considered as a bad margin.

The main goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH) and it can be done in the following two steps −

- First, SVM will generate hyperplanes iteratively that segregates the classes in best way.
- Then, it will choose the hyperplane that separates the classes correctly.

For our Model:

Our test size was 30% of all the data points in all the models, the code sample is attached below:

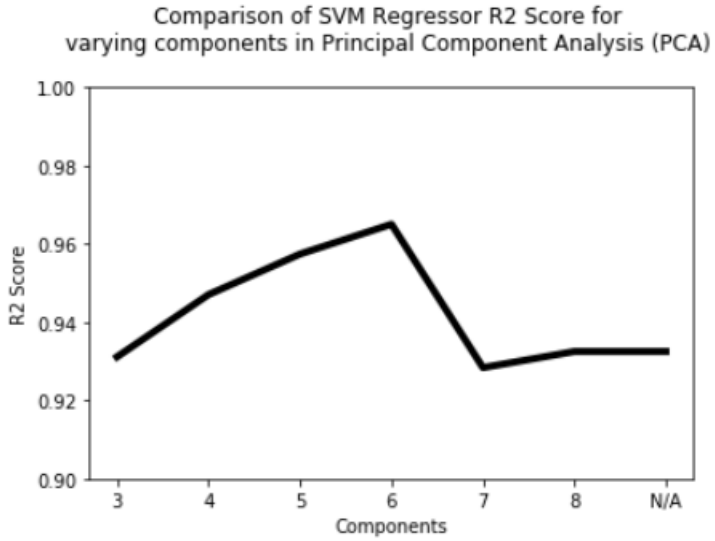
```
from sklearn.svm import SVR
svr = SVR()
svr.fit(X_train,y_train)
R21 = xgreg.score(X_test,y_test)
comp = [3, 4, 5, 6, 7, 8]
R2_Scores = []
for c in comp:
    X = Normdata[['Depth', 'ROP', 'WOB', 'DelP', 'Torque', 'RPM', 'Flow Rate',
      'Mud Weight', 'Pump Pressure']]
    y = Normdata['ECD']
```

```
    pca = PCA(n_components = c)
    pca = pca.fit_transform(X)
    X = pd.DataFrame(pca)
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75, random_state=101)
    svr.fit(X_train,y_train)
    R2 = DTreg.score(X_test,y_test)
    R2_Scores.append(R2)
comp.append('N/A')
R2_Scores.append(R21)
R2_Scores
```

The accuracy scores we got were:

We imported from sklearn library from python and applied on 70% train and 30%test data and found the following accuracy scores:

R-squared (Dim=3): 0.9311
R-squared (Dim=4): 0.9470
R-squared (Dim=5): 0.9574
R-squared (Dim=6): 0.9650
R-squared (Dim=7): 0.9284
R-squared (Dim=8): 0.9325
R-squared (None): 0.9325



6. K-Nearest Neighbour Classification

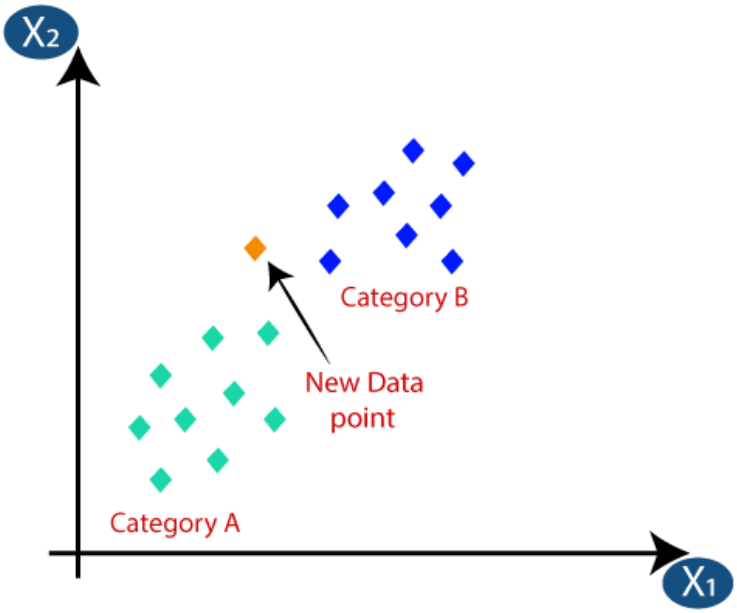
- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most like the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

Working Algorithm:

The K-NN working can be explained on the basis of the below algorithm:

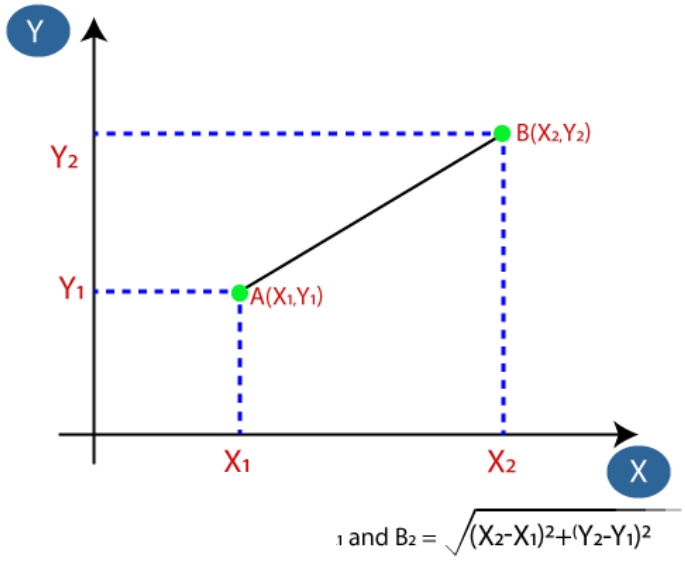
- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbour is maximum.
- **Step-6:** Our model is ready

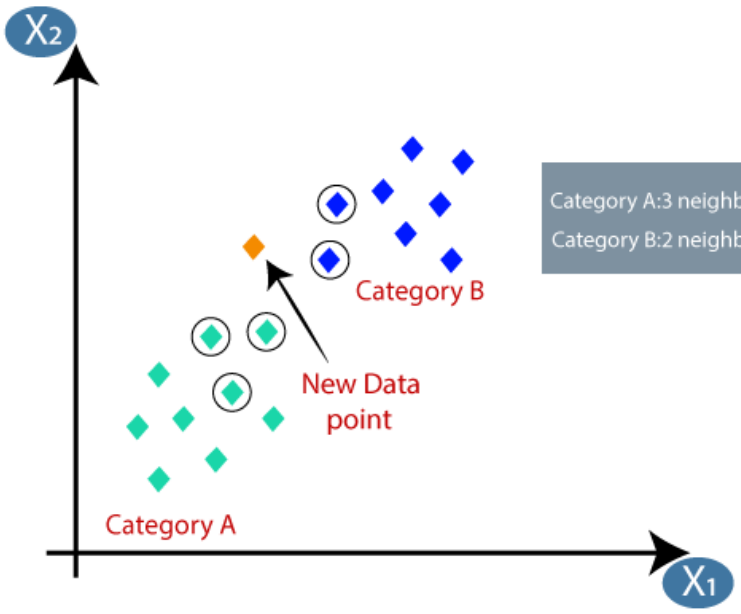
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- Firstly, we will choose the number of neighbours, so we will choose the k=5.
- We will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:

- By calculating the Euclidean distance we got the nearest neighbours, as three nearest neighbours in category A and two nearest neighbors in category B. Consider the below image:





Category A:3 neighbors
Category B:2 neighbors

- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

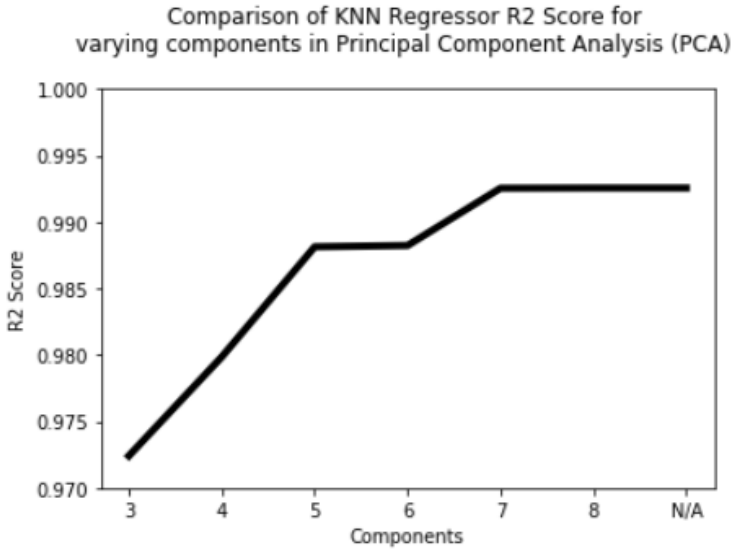
For our Model:

```
from sklearn.neighbors import KNeighborsRegressor
clf = KNeighborsRegressor(5)
clf.fit(X_train,y_train)
R21 = clf.score(X_test,y_test)
comp = [3, 4, 5, 6, 7, 8]
R2_Scores = []
for c in comp:
    X = Normdata[['Depth', 'ROP', 'WOB', 'DelP', 'Torque', 'RPM', 'Flow Rate',
      'Mud Weight', 'Pump Pressure']]
    y = Normdata['ECD']
    pca = PCA(n_components = c)
    pca = pca.fit_transform(X)
    X = pd.DataFrame(pca)
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75, random_state=101)
    clf.fit(X_train,y_train)
    R2 = DTreg.score(X_test,y_test)
    R2_Scores.append(R2)
comp.append('N/A')
R2_Scores.append(R21)
R2_Scores
```

The accuracy scores we got were:

We imported from sklearn library from python and applied on 70% train and 30%test data and found the following accuracy scores:

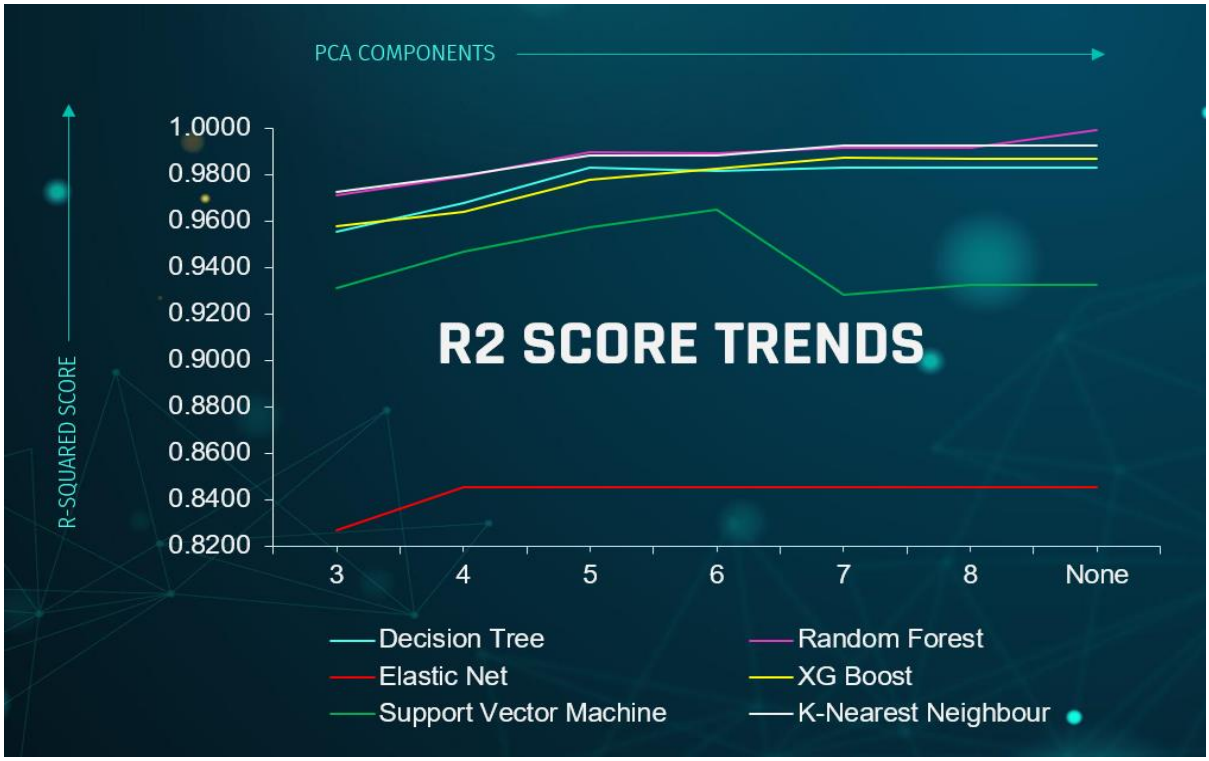
R-squared (Dim=3): 0.9724
R-squared (Dim=4): 0.9798
R-squared (Dim=5): 0.9881
R-squared (Dim=6): 0.9882
R-squared (Dim=7): 0.9925
R-squared (Dim=8): 0.9926
R-squared (None): 0.9926

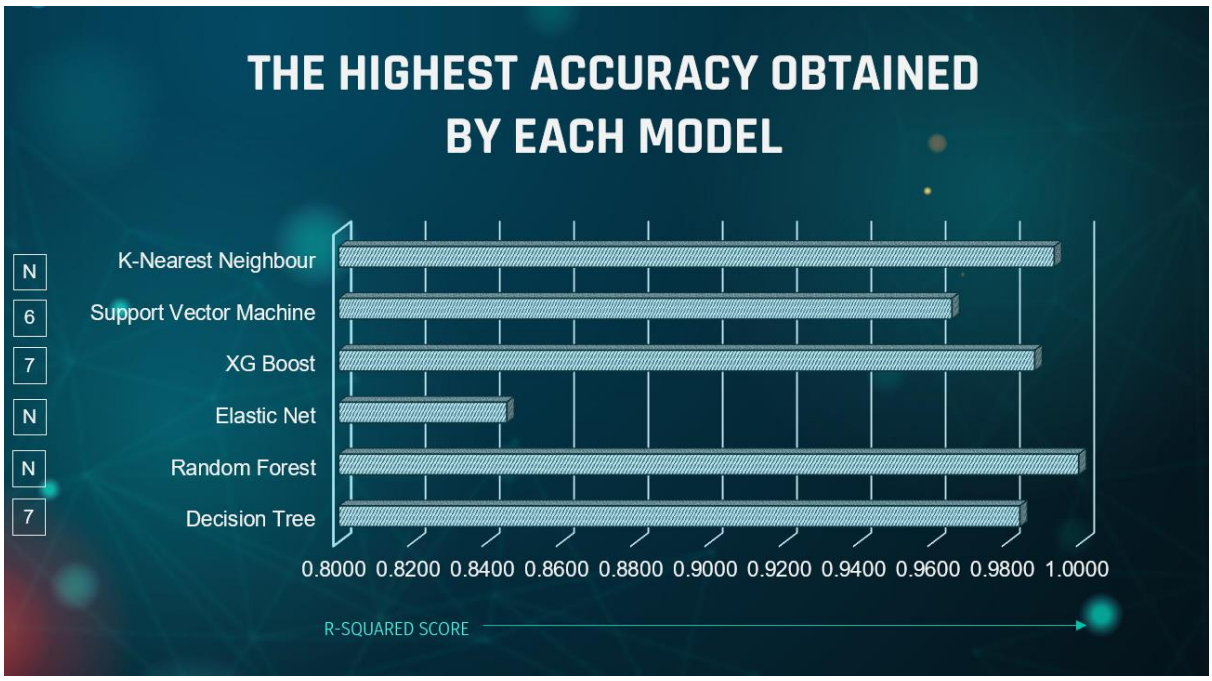


Results and Comparison of Models

Now as we have applied the different regression models for prediction of ECD we can compare all and tell which model is best suited for calculation of ECD for that particular well:

R2 comparison:





Conclusions

Obtaining real-time information about ongoing operations is crucial for safe and efficient drilling operations. The data cleaning and EDA as done on the field data after which Five intelligent machines XGboost, RF, and Decision tree, SVM, Elastic net regressor were developed to predict the ECD while drilling. PCA was applied to reduce the data set dimensions and compare the result of the PCA-based XGboost, RF, and Decision tree, SVM, Elastic net regressor. Based on the results, the following can be concluded:

- The Random Forest achieved the the highest R^2 compared to all the models and predicted the ECD with almost 99.92% accuracy.
- Elastic Net Regressor predicted the values poorly with its highest accuracy only being 84.52%.
- Varying PCA components on the Elastic Net Regressor was almost ineffective as it consistently predicted values with poor accuracy.
- Most of the models displayed highest accuracy without applying PCA except in the case of Support Vector Machine.
- Random Forest, KNN, Decision tree and XG boost were the high accuracy models in descending order respectively.

References:

1. Real-Time Prediction of Equivalent Circulation Density for Horizontal Wells Using Intelligent Machines ,Ahmed Alsaihati, Salaheldin Elkatatny*, and Abdulazeez Abdulraheem, ACS omega, 2021
2. Data-Driven Neural Network Model to Predict Equivalent Circulation Density ECD, Husam H. Alkinani, Abo Taleb T. Al-Hameedi, Shari Dunn-Norman, and Mustafa A. Al-Alwani, Missouri University of Science and Technology; Rusul A. Mutar, Ministry of Communications and Technology, Iraq; Waleed H. AlBazzaz, Kuwait Institute for Scientific Research,2019
3. Prediction of Pore and Fracture Pressures Using Support Vector Machine, Abdulmalek Ahmed S., Ahmed Abdulhamid Mahmoud, Salaheldin Elkatatny, Mohamed Mahmoud, and Abdulazeez Abdulraheem, King Fahd University of Petroleum & Minerals, 2019
4. https://www.datatechnotes.com
5. https://gdcoder.com/
6. https://www.analyticsvidhya.com/
7. https://www.tutorialspoint.com/machine_learning_with_python/