**CPU Performance Prediction**

horizontal line

# Introduction

Machine learning is a subset study area of Artificial Intelligence which helps system to automatically learn and predict the data for future. We can solve problems like classification and regression using Machine learning techniques. The two types of problems we are interested in applying the machine learning algorithms are as follows :

* Classification Problem
* Regression Problem

Here in this project, we considered two data sets and classified them into these problems.

**CPU Performance Prediction:** This dataset is considered to be having regression problem.

**Aim:** Given a data point we should be able to predict the performance of CPU based on the features of the data point.

Note: We realized that CPU Performance Prediction data set does not come under classification problem. With our intense interest on solving a classification problem, we considered another data set that comes under classification problem and solved it using some known machine learning algorithms. Below are the details regarding the second data set.

**Liver Disease affected Patient Classification :** This dataset is considered to be having classification problem.

**Aim:** Given a data point of patient we should be able to classify the patient into affected or not affected by liver disease based on the features of the data point.

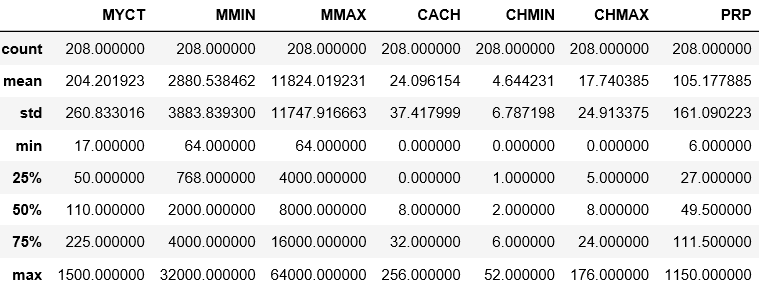
**Regression Problem**

**Data Set:** CPU Performance Prediction

**Features in Dataset:**

* Machine\_cycle\_time(MYCT)
* Minimum main memory (MMIN)
* Maximum main memory (MMAX)
* Cache memory (CACH)
* Minimum Channels (CHMIN)
* Maximum Channels of CPU(CHMAX)
* Published Rate Of Performance (PRP)
* Estimated Rate of Performance (ERP)

**Note:** We have dropped the ERP feature because it is the estimated performance rate that is given by the dataset providers, Also dropped the features like Vendor name and Model name from the dataset before running algorithms. Below is the data set description after dropping columns.



**Algorithms used for Computer Architecture Dataset:**

**Input:** Given Machine\_cycle\_time, minimum main memory, maximum main memory, cache memory, minimum channels and maximum channels of CPU.

**Expected Output:** Predicting the cpu performance.

**Metrics Used**: In regression problem we use the R2-score or coefficient of determination for Regression.

**Formula for R2-Score:** 1- (residual sum of squares/total sum of squares)

Residual sum of squares= sum of squares of error where error is equal to the difference of y-actual and y-predicted

Total sum of squares= difference of (y-mean(y))^2

If r2 -score is one then our model is really good.

If r2-score is less than or equal to zero our model is not performing well.

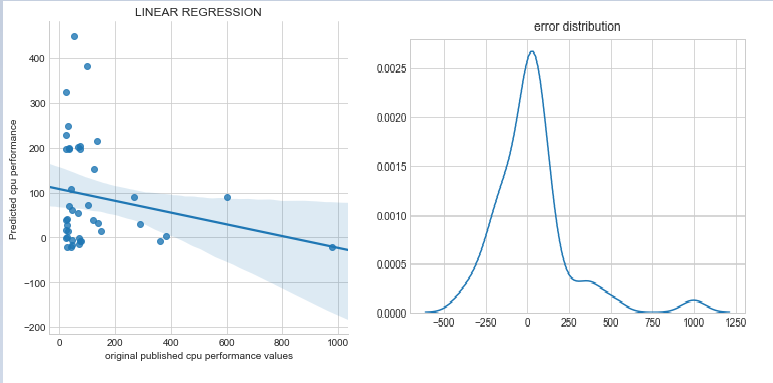
**Algorithm 1: Linear Regression**

Linear-regression is the classic algorithm designed specifically for regression we tried to use sklearn to perform Linear Regression and we got the below ou

**R2 training score:** 0.8794999309611982  
**R2 testing score:** 0.7137043555033928  
**Coefficients for polynomial regression:** [13.88326149 62.7116482 70.58810316 19.92925294 -3.6970514 39.47064592]  
**Intercept for linear regression:** 103.18908234444068

We used Error distribution of testing points and we plotted predicted vs actual plots to see where our regression model is not performing well and how well separated are our error distributions.

From the below figure we understand that, more error distribution lies from -100 to +100 and more values are centered towards zero according to the distribution. From our understanding, linear regression is performing well on the training\_score and not as expected in Testing.

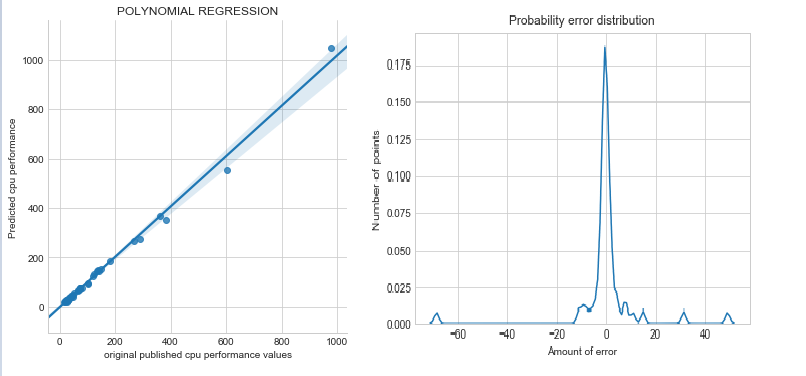


**Algorithm 2: Polynomial Linear Regression**

One of the crucial aspects in machine learning is feature engineering taking the feature from the space of Rd to Rd+k where Rd+k has more features than Rd sometimes yields good results. So we applied polynomial featurization on training and testing data and performed polynomial regression of degree 2. By doing this we acquired the results better than **linear regression**

**R2 training score:** 0.9721102991831799  
**R2 testing score:** 0.8164187879302189

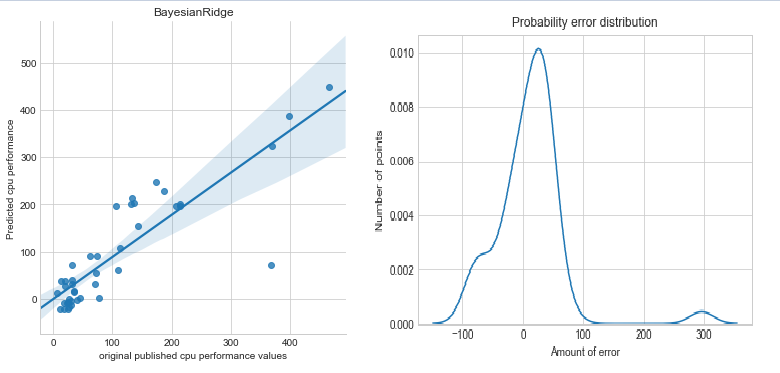
**Coefficients for polynomial regression:** [-1.03121925e-08 8.50905026e-02 2.50126240e-03 1.08956167e-03  
 4.58771441e-01 5.87940196e+00 9.83977615e-01 -5.99274254e-05  
 -6.27206714e-07 8.49829976e-07 -1.86270049e-03 -4.30749876e-03  
 -5.90581814e-03 9.77733941e-07 2.32020858e-07 -1.15495766e-04  
 -1.11707112e-03 2.36100466e-04 -9.01965795e-09 5.15167329e-05  
 -3.22801676e-05 1.15060284e-05 -2.58841442e-03 6.46199655e-02  
 3.94492103e-03 1.73203049e-01 -2.18503969e-01 7.83535384e-03]  
**Intercept for polynomial-regression:** -0.7372044676884855

Testing score using polynomial regression has improved when compared with linear regression. The observation of the error distribution shows that the in linear regression was skewed towards left whereas here in polynomial regression the errors tend more towards zero and variance of the bell curve increased and we see very less values at the points towards 100 and 200.  
  


**Algorithm 3: Bayesian Ridge Regression**

As a variation technique of linear regression we used bayesian ridge regression by taking advantage of Bayes Theorem and conditional independence. Below are the output from bayesian ridge regression.

**R2 training score:** 0.8794522753486081  
**R2 testing score:** 0.7183582433537425  
**Coefficients for Bayesian Ridge Regression:** [13.42055879 61.98804101 69.88955647 20.04503542 -2.70568795 39.01936928]  
**Intercept for Bayesian Ridge Regression:** 103.2391305971257



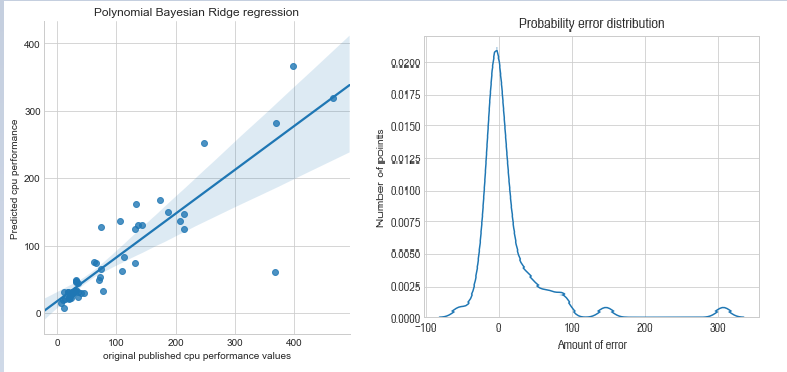
**Algorithm 4: Polynomial bayesian ridge regression**

As a slight modification we applied polynomial bayesian ridge regression of degree 2 we got the results as below:

**R2 training score:** 0.9578532690300267  
**R2 testing score:** 0.7362331759069163

**The coefficients are:** [ 1.46695271e-14 -1.58777591e-05 8.92609550e-04 1.54187002e-03  
 1.10040060e-05 1.74848439e-06 9.32422412e-07 -9.94960894e-06  
 2.84337601e-06 -1.68217988e-06 1.37046898e-04 5.24901359e-06  
 -4.78744313e-04 8.71271796e-08 2.03734629e-07 -1.11393905e-04  
 -2.07255518e-04 3.29866990e-04 -7.56623318e-10 7.77610587e-05  
 -2.50129475e-05 -8.81411823e-06 7.92000039e-04 1.30488996e-04  
 3.64107755e-04 -2.32260242e-06 -2.16147057e-05 -2.10172637e-04]

**The intercept is :** 26.093608609155936



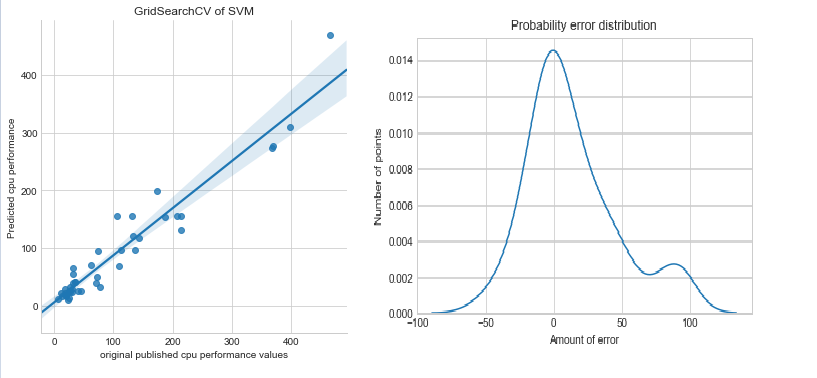
**Algorithm 5: Support Vector Machines Regressor.**

By applying SVM Regression without controlling the hyperparameters we got very bad results.  
**R2 training score:** -0.02588841943933673  
**R2 testing score:** -0.026993756794967627  
**Support for Svm\_regression:**  166

So we studied support Vector Machines and applied GridSearchCV to control the hyperparameters like c and gamma where c is the hinge loss and gamma is the parameter in RBF kernel

SVR(C=1000, cache\_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='auto',  
 kernel='rbf', max\_iter=-1, shrinking=True, tol=0.001, verbose=False)  
**R2\_SCORE for Traing\_score:** 0.9844600554200172  
**R2\_SCORE for Testing\_score:** 0.8999970707138836  
**Total number of Support\_vectors for Svm\_regression:** 167.

The error distribution here is very less it is varied from -100 to +100.which is significantly less.and SVM has the great potential to give best results but finding the right hinge loss gives the best results.

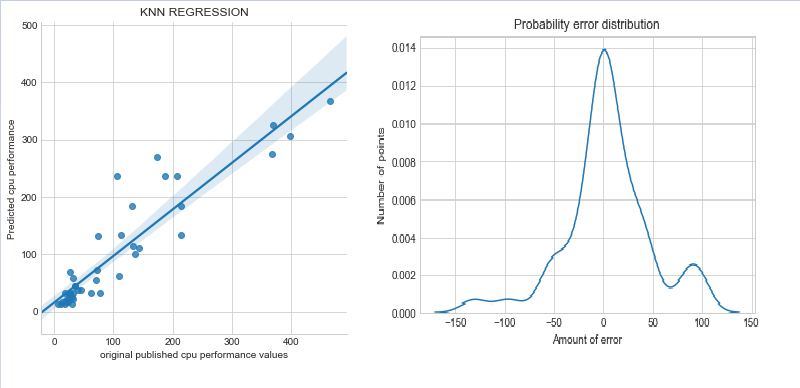


**Algorithm 6: KNN-Regressor**

Applying KNN regressor gives good results here the hyperparameter is finding the right K-Value is 1. By applying 10 fold cross validation.

KNeighborsRegressor(algorithm='auto', leaf\_size=30, metric='minkowski',  
 metric\_params=None, n\_jobs=1, n\_neighbors=1, p=2,  
 weights='uniform')  
**R2\_score for Training knn\_reg:** 0.9930106547971913  
**R2 score for Testing knn\_reg:** 0.8461244414384788

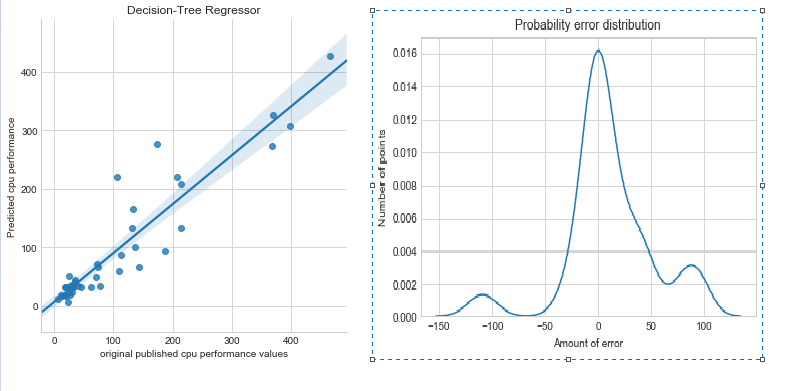
Cross-validation error for my k=1 is 9072.531914893616  
Cross-validation error for my k=3 is 33832.36170212766  
Cross-validation error for my k=5 is 51712.36170212766  
Cross-validation error for my k=7 is 56045.29787234042  
Cross-validation error for my k=9 is 47617.36170212766  
Cross-validation error for my k=11 is 47046.14893617021  
Cross-validation error for my k=13 is 47490.127659574464  
Cross-validation error for my k=15 is 65085.127659574464  
Cross-validation error for my k=17 is 67034.5744680851  
Cross-validation error for my k=19 is 63017.89361702128  
Cross-validation error for my k=21 is 58559.17021276596  
Cross-validation error for my k=23 is 56045.29787234042  
Cross-validation error for my k=25 is 47426.574468085106



**Algorithm 7: decision-Tree**

Decision Tree has the behaviour to overfit easily .choosing the right depth gives good results .finding max\_depth:9 gives good results. We applied grid search to find the right height of decision Tree.

DecisionTreeRegressor(criterion='mse', max\_depth=9, max\_features=None,  
 max\_leaf\_nodes=None, min\_impurity\_decrease=0.0,  
 min\_impurity\_split=None, min\_samples\_leaf=1,  
 min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,  
 presort=False, random\_state=0, splitter='best')  
**Training\_r2\_score:** 0.9941417278543623  
**Testing\_r2\_score:** 0.8750089481049828

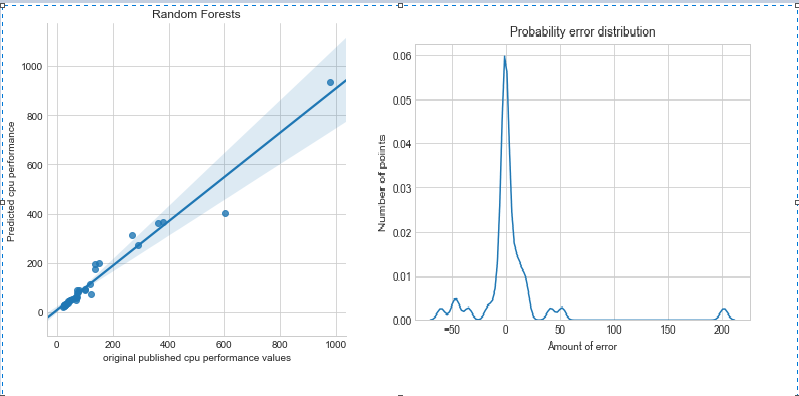


**Algorithm 8: Bagging with Random Forests**

Bagging is the Technique which is specifically used for the models which has the high variance and low bias .This bagging has no of estimators to control Finding the right no of decision Trees is crucial we applied grid search with 10 fold cross validation with 21 decision Trees with we got an output below.

RandomForestRegressor(bootstrap=True, criterion='mse', max\_depth=None,  
 max\_features='auto', max\_leaf\_nodes=None,  
 min\_impurity\_decrease=0.0, min\_impurity\_split=None,  
 min\_samples\_leaf=1, min\_samples\_split=2,  
 min\_weight\_fraction\_leaf=0.0, n\_estimators=21, n\_jobs=-1,  
 oob\_score=False, random\_state=0, verbose=0, warm\_start=False)

**Training\_r2\_score:** 0.9817751483306185  
**Testing\_r2\_score:** 0.9199904658711162



### Sklearn implementation of Gradient Boosting Trees will take more time to train very small amount of data and gradient boosting trees cannot be trivially parallelized. That is why i see n\_jobs option absent here. Hyperparameters to control here are learning\_rate and no of trees and depth of the tree.

### One key idea in boosting is we need to keep the depth of tree as less as possible because boosting is applied to the models with high bias and low-variance.

### When there is decrease in bias at the same time the variance in the model might increase.

### So controlling learning rate is essential to avoid overfit or underfit

### Applying subsample <1 here is similar to row sampling in the bootstrap technique.

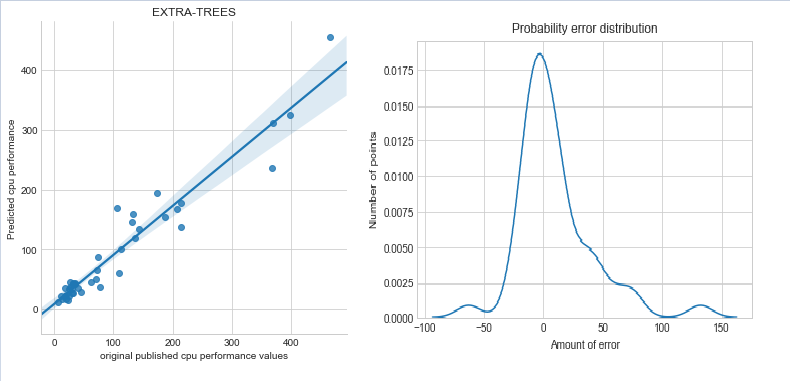
### Column sampling is not implemented here in sklearn gradient boosting trees.if implemented we got one more level of randomness

### Only decision-tree can be applied as base learners or estimators other algorithm like linear regression cannot be applied.

**Algorithm 9: Extremely Randomized Trees(Extra Trees):**

Applying Extra-Trees gives the next level of randomness apart from Bootstrapping Technique we used in Bagging we need to control the no of estimators we applied grid search CV to find the right no of estimators we got no of estimators as 68.

RandomForestRegressor(bootstrap=True, criterion='mse', max\_depth=None,  
 max\_features='auto', max\_leaf\_nodes=None,  
 min\_impurity\_decrease=0.0, min\_impurity\_split=None,  
 min\_samples\_leaf=1, min\_samples\_split=2,  
 min\_weight\_fraction\_leaf=0.0, n\_estimators=68, n\_jobs=-1,  
 oob\_score=False, random\_state=0, verbose=0, warm\_start=False)  
**Training\_r2\_score:** 0.9861289892354653  
**Testing\_r2\_score:** 0.9103812966684149



From the plot we see that error got reduced -50 to 100+ which is significantly lower than every plot we have seen before.

**Algorithm 10: Gradient Boosting Decision Trees(Boosting)**

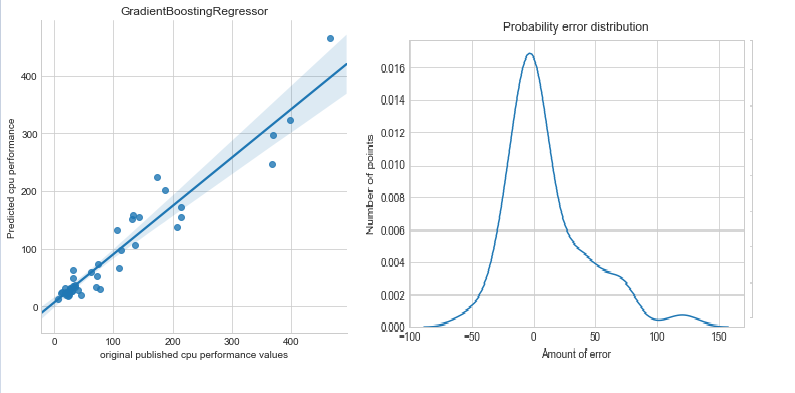
In Gradient Boosting Decision Trees we need to control 4 hyperparameters such as max depth, learning\_rate, n\_estimators,subsample. We apply boosting specifically for the regressors which has high bias and less variance.

Here Training under with 40% subsample with bootstrapping gives really good results despite training on 100% subsample

GradientBoostingRegressor(alpha=0.9, criterion='friedman\_mse', init=None,  
 learning\_rate=0.1, loss='ls', max\_depth=3, max\_features=None,  
 max\_leaf\_nodes=None, min\_impurity\_decrease=0.0,  
 min\_impurity\_split=None, min\_samples\_leaf=1,  
 min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,  
 n\_estimators=100, presort='auto', random\_state=0,  
 subsample=0.4, verbose=0, warm\_start=False)  
Training\_r2\_score:0.9880518621093645  
Testing\_r2\_score:0.9081562594074961

Just applying Gradient boosting by sklearn gives the results below:

GradientBoostingRegressor(alpha=0.9, criterion='friedman\_mse', init=None,  
 learning\_rate=0.1, loss='ls', max\_depth=3, max\_features=None,  
 max\_leaf\_nodes=None, min\_impurity\_decrease=0.0,  
 min\_impurity\_split=None, min\_samples\_leaf=1,  
 min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,  
 n\_estimators=100, presort='auto', random\_state=0,  
 subsample=1.0, verbose=0, warm\_start=False)  
Training\_score:0.9934594539871406  
Testing\_score:0.9169590212195388



**Gradient Descent and Stochastic Gradient Descent:**

A small experiment has been done to check how general gradient descent works when compared to Stochastic gradient descent. Stochastic Gradient converge faster than general gradient descent for getting optimal values. From this experimentation we know choosing the right learning rate and weights are crucial for optimal convergence of weights. General gradient descent requires more no of iterations than stochastic gradient descent. Stopping the algorithm if our weights do not change significantly which we did not handle but there are many techniques which we learnt and will apply in our future projects

**Future Scope:**

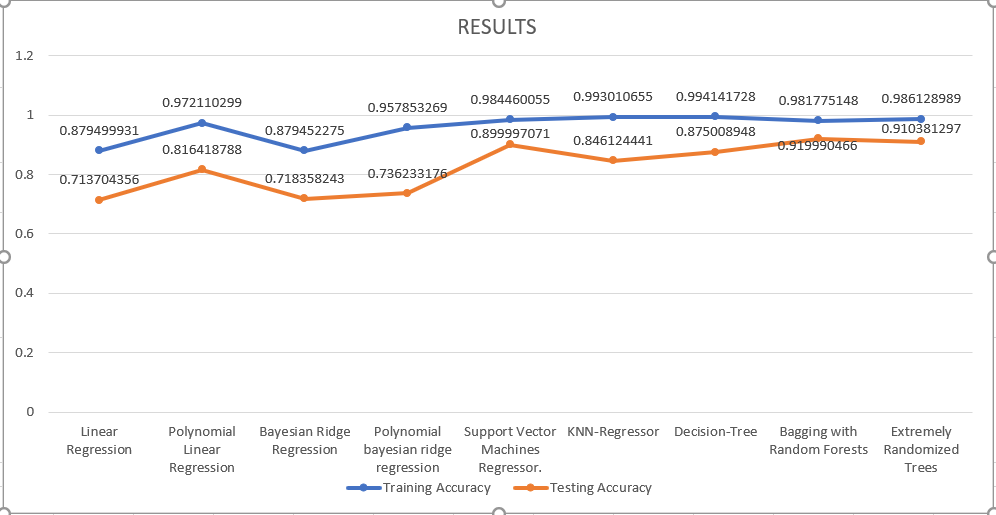
1.We heard XG-BOOST a library from University of washington a phenomenal library we can use this library to get good results because in this we have linear regression and logistic regression and decision trees and l1 and l2 regularization which is really crucial when we deal with large datasets which we can apply boosting on this algorithms for better accuracies.

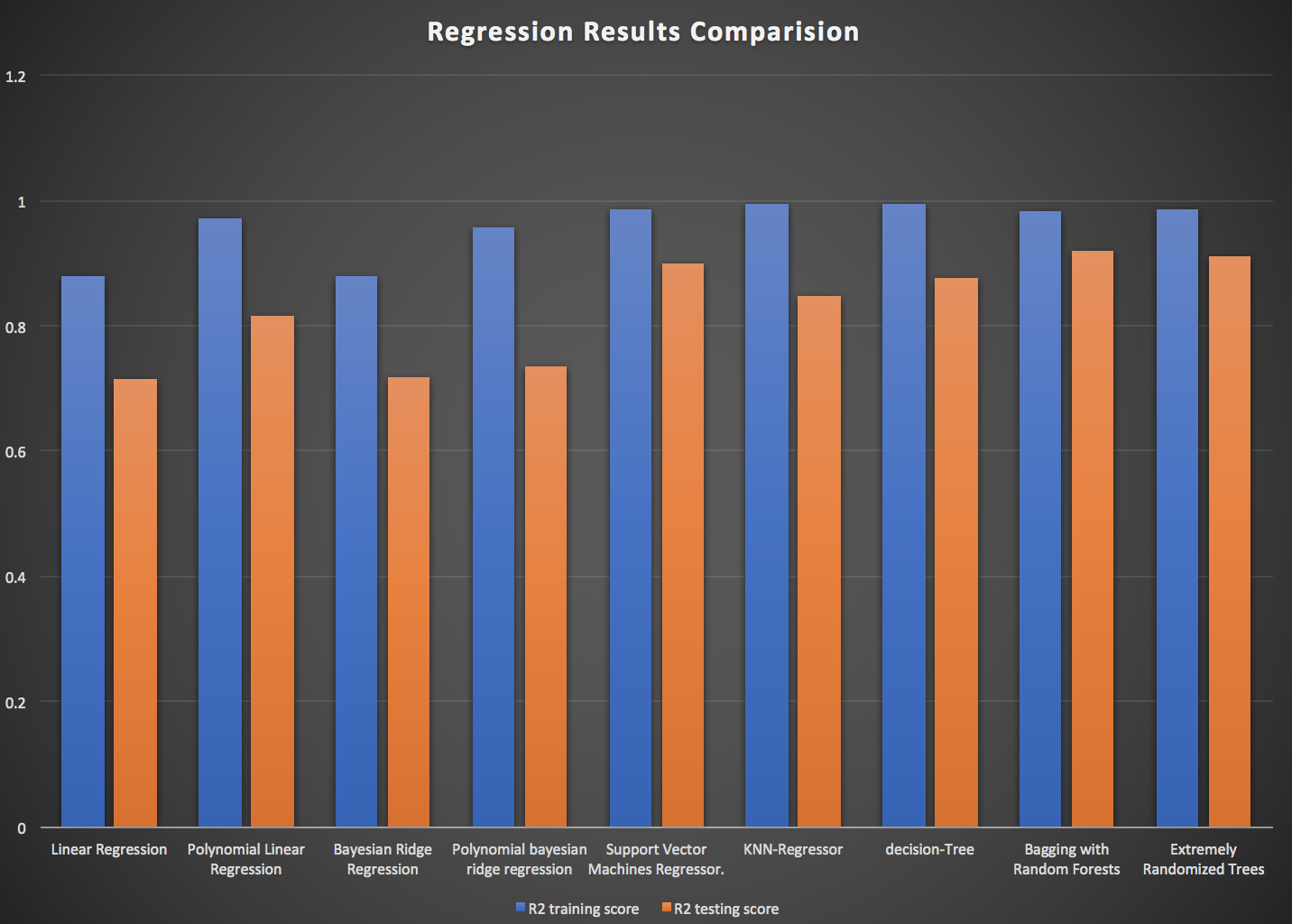
2.Learnt about the concept of stacking where we can combine multiple classifier to design one robust meta classifier to yield good results.

3.Cascading can also be applied to yield better results.

**RESULT COMPARISON:**

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **R2 Training score** | **R2 Testing score** |
| Linear Regression | 0.879499931 | 0.713704356 |
| Polynomial Linear Regression | 0.972110299 | 0.816418788 |
| Bayesian Ridge Regression | 0.879452275 | 0.718358243 |
| Polynomial bayesian ridge regression | 0.957853269 | 0.736233176 |
| Support Vector Machines Regressor. | 0.984460055 | 0.899997071 |
| KNN-Regressor | 0.993010655 | 0.846124441 |
| Decision-Tree | 0.994141728 | 0.875008948 |
| Bagging with Random Forests | 0.981775148 | 0.919990466 |
| Extremely Randomized Trees | 0.986128989 | 0.910381297 |





**CONCLUSION :**

Learned and implemented new machine learning techniques to improve the accuracy and We believe we made a successful attempt in applying the machine learning techniques that we have been taught.