PREDICTION OF BLOG FEEDBACK

A PROJECT REPORT (Project Phase - II)

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE ACADEMIC PROJECT

IN

COMPUTER SCIENCE AND ENGINEERING

BY

KONERU SRICHARAN	**2210314931**
GADDAM MANIKANTH REDDY	**2210314916**
BASETTY SAI VENKAT	**2210314902**
MANNE MANI SRAVALYA	**2210314937**

UNDER THE GUIDANCE OF

Mrs. DASARI VIJYALAKSHMI

ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING.

SCHOOL OF TECHNOLOGY
GITAM (Deemed to be University)
HYDERABAD CAMPUS

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MARCH-2018

SCHOOL OF TECHNOLOGY
GITAM (Deemed to be University)
HYDERABAD CAMPUS
(Estd. u/s 3 of UGC Act 1956)

Rudraram Village, Patancheru Mandal, Sangareddy Dist – 502329, TS, INDIA.
Ph: 08455-220556/57; Fax : 08455-20046
Website : www.gitam.edu

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitled ***PREDICTION OF BLOG FEEDBACK*** was presented satisfactorily to Department of Computer Science and Engineering, SCHOOL OF TECHNOLOGY, GITAM UNIVERSITY, HYDERABAD CAMPUS by ***KONERU SRICHARAN, GADDAM MANIKANTH REDDY, BASETTY SAI VENKAT, MANNE MANI SRAVALYA*** bearing H.T-NO's ***2210314931, 2210314916, 2210314902, 2210314937*** in partial fulfillment of requirement for their project work (Project Phase-II) carried out under my guidance and supervision.

Mrs. DASARI VIJYA LAKSHMI
ASSISTANT PROFESSOR
Project guide

B.Rajendra Prasad Babu
Project Coordinator

Dr.S.Phani kumar
Head of the Department
Dept of CSE

Name and Signature of the project panel members.

1.

2.

3.

ACKNOWLEDGEMENT

The success and outcome of this project required a lot of guidance and assistance from many people and we are extremely privileged to have got this all along the completion of our project. All that we have done is only due to such supervision and assistance and we would not forget to thank them.

*We owe our deep gratitude to our project internal guide **DASARI VIJYA LAKSHMI**, ASSISTANT PROFESSOR, who took keen interest on our project work and guided us all along, till the completion of our project phase-I work by providing all the necessary information for developing a good system.*

*We heartily thank our project coordinator **B. RAJENDRA PRASAD BABU**, ASSISTANT PROFESSOR, for his guidance and suggestions during this project work.*

*At the outset we thank our Head of the Department **DR. S. PHANI KUMAR** and our Honourable Pro-Vice Chancellor **PROF. N. SIVA PRASAD** for the moral support and the excellent facilities provided. I would also like to thank all the teaching and non-teaching staff members of Computer Science department who have extended their full cooperation during our course.*

KONERU SRICHARAN GADDAM MANIKANTH REDDY

BASETTY SAI VENKAT MANNE MANI SRAVALYA

DECLARATION

We ***KONERU SRICHARAN, GADDAM MANIKANTH REDDY, BASETTY SAI VENKAT, MANNE MANI SRAVALYA,*** bearing roll numbers***, 2210314931, 2210314916, 2210314902, 2210314937*** hereby declare that the project report entitled ***"PREDICTION OF BLOG FEEDBACK"*,** under the guidance of **Mrs.** ***DASARI VIJYA LAKSHMI, ASSISTANT PROFESSOR,*** Department of Computer Science and Engineering, School of Technology, GITAM University , Hyderabad, has been submitted for Project Phase – II evaluation .

This is a record of bona fide work project phase- II as a part of major project carried out by us and the content embodied in this project have not been reproduced /copied from any source. The content embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

S.NO	**NAME**	**PIN NUMBER**	**SIGNATURE**
1	KONERU SRICHARAN	2210314931	
2	GADDAM MANIKANTH REDDY	2210314916	
3	BASETTY SAI VENKAT	2210314902	
4	MANNE MANI SRAVALYA	2210314937	

Table of Contents

ABSTRACT

The last decade lead to an unbelievable growth of the importance of social media. Due to the huge amounts of documents/data appearing in social media, there is an enormous need for the automatic analysis of such documents/data.

Our data originates from blog posts. The raw HTML-documents of the blog posts were crawled and processed. The prediction task associated with the data is the prediction of the number of comments in the upcoming 24 hours. To simulate this situation, we choose a base time (in the past) and select the blog posts that were published at most 72 hours before the selected base date/time. Then, we calculate all the features of the selected blog posts from the information that was available at the base time, therefore each instance corresponds to a blog post. The target is the number of comments that the blog post received in the next 24 hours relative to the base time.

In the train data, the base times were in the years 2010 and 2011. In the test data the base times were in February and March 2012. This simulates the real-world situation in which training data from the past is available to predict events in the future.

The train data was generated from different base times that may temporally overlap. Therefore, if you simply split the train into disjoint partitions, the underlying time intervals may overlap. Therefore, you should use the provided, temporally disjoint train and test splits to ensure that the evaluation is fair.

LIST OF FIGURES

LIST OF SCREENS

1.INTRODUCTION

1.1 MOTIVATION

The massive data generated by the social media is considered of high business value, and data mining algorithms can be applied to such data to extract hidden information from data. Large volume of data should be analyzed, the latest algorithms should be modified to apply to big data. Data mining involves discovering novel, interesting, and potentially useful patterns from large data sets and applying algorithms to the extraction of hidden information. The objective of any data mining process is to build an efficient predictive or descriptive model of a large amount of data that not only best fits or explains it, but is also able to generalize to new data. Based on a broad view of data mining functionality, data mining is the process of discovering interesting knowledge from large amounts of data stored in either databases, data warehouses, or other information repositories. This interesting knowledge or the knowledge obtained can be used for various purposes based on the user who uses it.

1.2 PROBLEM DEFINITION

1.2.1 What is a blog?

A **blog** (a truncation of the expression "**weblog**") is a discussion or informational website published on the World Wide Web consisting of discrete, often informal diary-style text entries ("posts"). Posts are typically displayed in reverse chronological order, so that the most recent post appears first, at the top of the web page. Until 2009, blogs were usually the work of a single individual, occasionally of a small group, and often covered a single subject or topic. In the 2010s, "multi-author blogs" (MABs) have developed, with posts written by large numbers of authors and sometimes professionally edited. MABs from newspapers, other media outlets, universities, think tanks, advocacy groups, and similar institutions account for an increasing quantity of blog traffic.

1.2.2 What is feedback?

Feedback is information about reactions to a product, a person's performance of a task, etc. which is used as a basis for improvement.

A review or comment on a blog post is referred to as blog feedback.

1.2.3 The real problem!

There are about 2.73 million posts written each day. There are about 152 million blogs in the year 2013. These whooping figures shows the blog traffic on internet and their importance.

Each blog post receives feedback separately by various users who come across the blog post. Due to the huge amounts of such documents appearing in social media, analysis of all these documents by human experts is hopeless, and therefore there is an enormous need for the automatic analysis of such documents.

So our project aims at predicting number of comments a blog post would get in upcoming 24 hours after it is published. It helps the blog administrator to control and curate the contents of his blog.

1.3 OBJECTIVE OF PROJECT

The current analysis shows that there are numerous blog posts generated in a day. Typically, a blog may contain numerous posts written by various authors. The blog administrator cannot check the feedback given to each post. So, there is a need of automatic analysis of such content.

The aim of our project is to predict the number of comments a blog post would get in the upcoming 24 hours after it is published.

1.4 LIMITATIONS OF PROJECT

The project scope is strictly limited to prediction of number of comments a blog post would get in a given time frame. Prediction of content and nature of comments a blog post would get is out of the scope of this project.

2. LITERATURE SURVEY

2.1 INTRODUCTION

The **literature** survey is a text of a scholarly paper, which includes the current knowledge including substantive findings, as well as theoretical and methodological contributions to a topic.

In this segment of document, we list out the existing system, its disadvantages and proposed system.

2.2 EXISTING SYSTEM

Every day millions of people, some of whom have no technical ability whatsoever, write on their blogs. To meet this demand some amazing tools have been created that will allow anyone, even people with very little knowledge of computers, to have their own blog.

WordPress, Tumblr, Ghost, Blogger, Squarespace, Wix etc. are all famous blogging sites which provide a place for such content writers, to post things. There are still many people who like to share the details of their days. They may post twenty or thirty times a day, detailing when they ate lunch and when they headed home from work.

74,652,825 sites out there are depending on good ol' WordPress. That's one site per person in Turkey. Around 50% of this figure (close to 37 million) is hosted on the free WordPress.com. Due to the huge amounts of such documents appearing in social media, analysis of all these documents by human experts is hopeless, and therefore there is an enormous need for the automatic analysis of such documents.

2.3 DISADVANTAGE OF EXISTING SYSTEM

As said above blog administrator cannot check and analyse feedback given to each blog. So automatic analysis is required, as the first step blog data is taken, and number of comments is predicted using data mining and machine learning techniques.

2.4 PROPOSED SYSTEM

The proposed system aims at predicting the number of comments a blog would get in the upcoming 24 hours' time frame. This forms the first step in prediction of feedback which can be further extended to predict content and type of comments. This is done by using Python built in libraries.

3. ANALYSIS

3.1 INTRODUCTION

This segment of the document consists of software requirement specifications, content flow diagram and flowchart describing the project.

3.2 SOFTWARE REQUIREMENT SPECIFICATION (SRS)

A **software requirements specification** (**SRS**) is a description of a software system to be developed. It lays out functional and non-functional requirements and may include a set of use cases that describe user interactions that the software must provide.

3.2.1 User Specification

The user requirements specify what the user requires or expects from the system. Here the main requirement is to analyse or predict the number of comments a blog post would get.

3.2.2 Software Requirements

- Operating System
  - Operating system: Windows 7/8/8.1/10.
- Python Integrated Development Environment:
  - python 2.7.x and above
- Setup tools and pip to be installed for 3.6 and above
- Language: Python Scripting
- Jupyter Notebook.

3.2.3 Hardware Requirements

- Random Access Memory (RAM): 4 Giga Bytes (GB) or more.
- Processor: 64-bit architecture (preferably Intel i3 or i5 or i7 processor 6^{th} generation).
- Video memory: 8 Mega Bytes (MB).
- Hard Disk: 100 Giga Bytes (GB).
- Monitor: 15 inches.

3.3 CONTENT DIAGRAM OF PROJECT

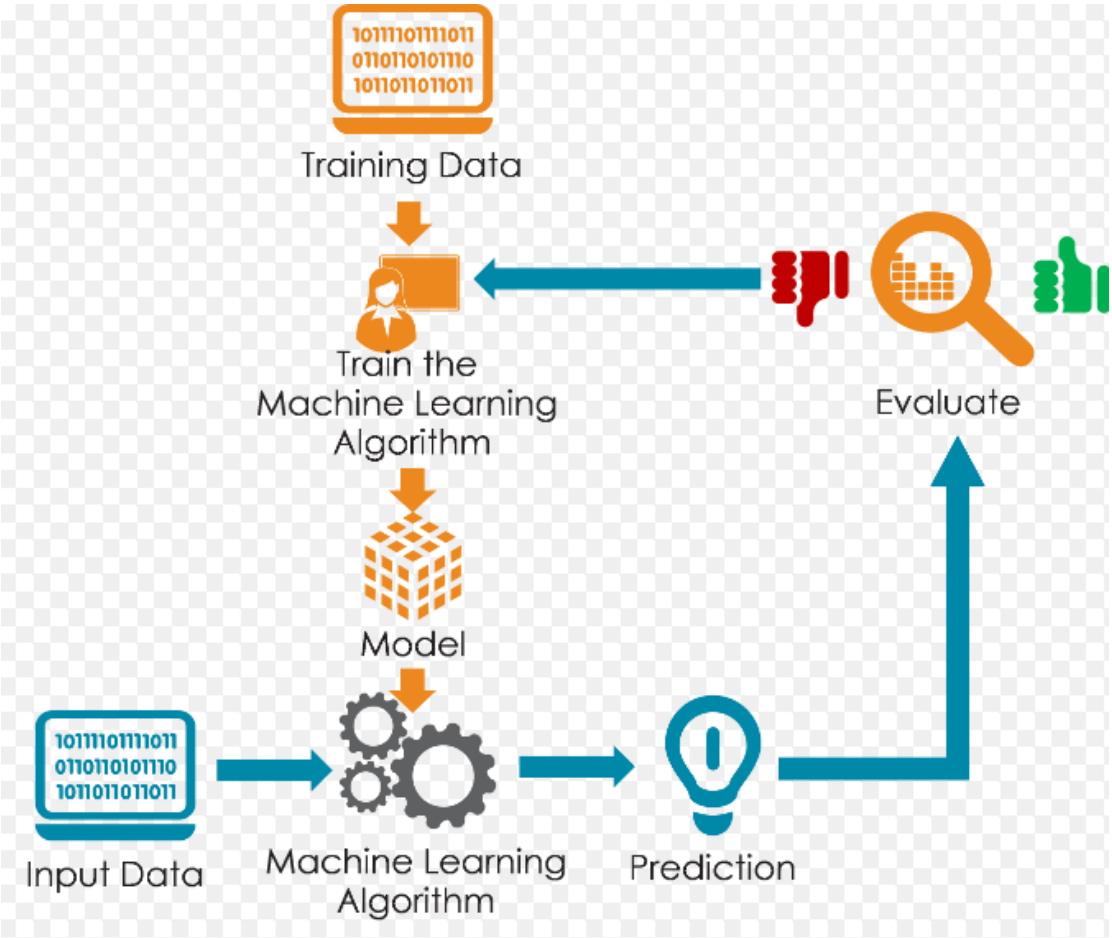


Figure.1: Content Diagram

3.4 ALGORITHMS AND FLOWCHARTS

3.4.1 Random Forest Regressor

3.4.1.1 Introduction

- Random forest (or random forests) is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees.

- The term came from random decision forests that was first proposed by Tin Kam Ho of Bell Labs in 1995.

- The method combines Breiman's "bagging" idea and the random selection of features.

- Each tree is constructed using the following algorithm:

    1. Let the number of training cases be *N*, and the number of variables in the classifier be *M*.

    2. We are told the number *m* of input variables to be used to determine the decision at a node of the tree; *m* should be much less than *M*.

    3. Choose a training set for this tree by choosing *n* times with replacement from all *N* available training cases (i.e. take a bootstrap sample). Use the rest of the cases to estimate the error of the tree, by predicting their classes.

    4. For each node of the tree, randomly choose *m* variables on which to base the decision at that node. Calculate the best split based on these *m* variables in the training set.

    5. Each tree is fully grown and not pruned (as may be done in constructing a normal tree classifier).

- For prediction a new sample is pushed down the tree. It is assigned the label of the training sample in the terminal node it ends up in. This procedure is iterated

over all trees in the ensemble, and the average vote of all trees is reported as random forest prediction.

3.4.1.2 Flowchart

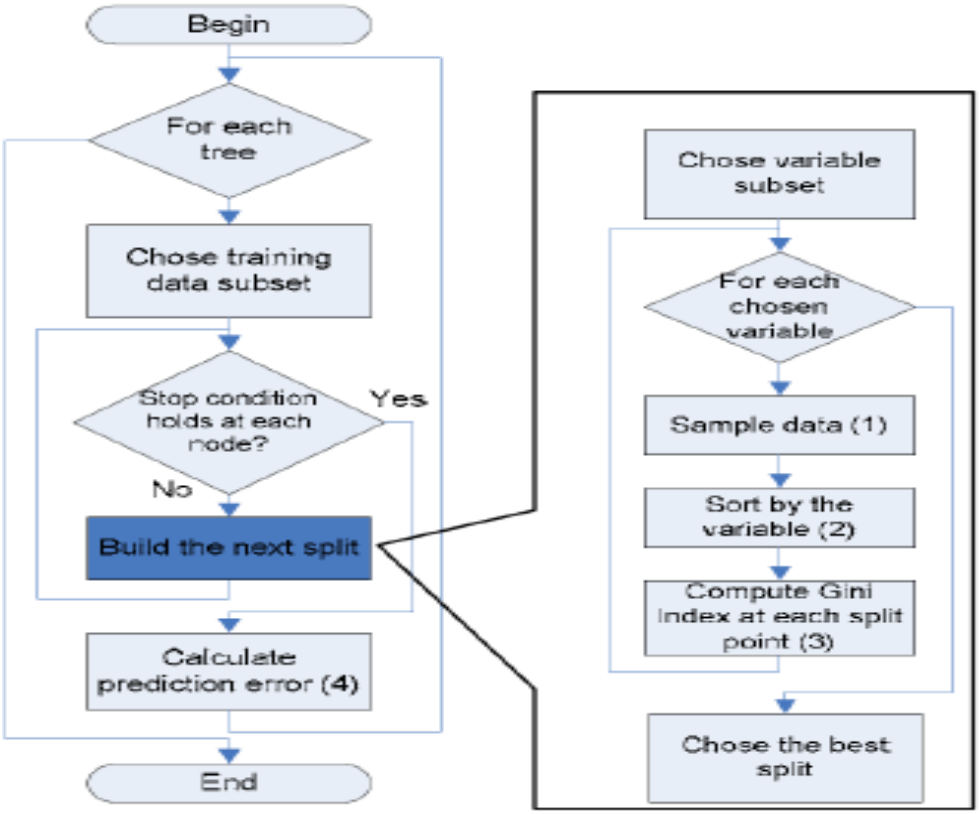


Figure.2: Flow Chart for Random forest Regressor

3.4.1.3 Practical Considerations

- Splits are chosen according to a purity measure:
  - E.g. squared error (regression), Gini index or deviance (classification)
- How to select N?
  - Build trees until the error no longer decreases
- How to select M?
  - Try to recommend defaults, half of them and twice of them and pick the best.

3.4.1.4 Feature and Advantages

The advantages of random forest are:

- It is one of the most accurate learning algorithms available. For many data sets, it produces a highly accurate classifier.
- It runs efficiently on large databases.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- It has methods for balancing error in class population unbalanced data sets.
- Generated forests can be saved for future use on other data.
- Prototypes are computed that give information about the relation between the variables and the classification.
- It computes proximities between pairs of cases that can be used in clustering, locating outliers, or (by scaling) give interesting views of the data.
- The capabilities of the above can be extended to unlabelled data, leading to unsupervised clustering, data views and outlier detection.
- It offers an experimental method for detecting variable interactions.

3.4.1.5 Disadvantages

- Random forests have been observed to overfit for some datasets with noisy classification/regression tasks.
- For data including categorical variables with different number of levels, random forests are biased in favour of those attributes with more levels. Therefore, the

variable importance scores from random forest are not reliable for this type of data.

3.4.1.6 Conclusion

- Fast fast fast!
  - RF is fast to build. Even faster to predict!
  - Practically speaking, not requiring cross-validation alone for model selection significantly speeds training by 10x-100x or more.
  - Fully parallelizable … to go even faster!
- Automatic predictor selection from large number of candidates
- Resistance to over training
- Ability to handle data without pre-processing
  - data does not need to be rescaled, transformed, or modified
  - resistant to outliers
  - automatic handling of missing values
- Cluster identification can be used to generate tree-based clusters through sample proximity.

3.4.2 Adaboost Regression

3.4.2.1 Introduction

- **AdaBoost**, short for Adaptive Boosting, is a machine learning meta-algorithm formulated by Yoav Freund and Robert Schapire, who won the 2003 Gödel Prize for their work.
- It can be used in conjunction with many other types of learning algorithms to improve performance.
- The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier.
- AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favour of those instances misclassified by previous classifiers.
- AdaBoost is sensitive to noisy data and outliers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms.
- The individual learners can be weak, but if the performance of each one is slightly better than random guessing, the final model can be proven to converge to a strong learner.
- Every learning algorithm tends to suit some problem types better than others, and typically has many different parameters and configurations to adjust before it achieves optimal performance on a dataset, AdaBoost (with decision trees as the weak learners) is often referred to as the best out-of-the-box classifier.
- When used with decision tree learning, information gathered at each stage of the AdaBoost algorithm about the relative 'hardness' of each training sample is fed into the tree growing algorithm such that later trees tend to focus on harder-to-classify examples.

3.4.2.2 Training

- A weak classifier (decision stump) is prepared on the training data using the weighted samples. Only binary (two-class) classification problems are supported, so each decision stump makes one decision on one input variable and outputs a +1.0 or -1.0 value for the first or second class value.

- The misclassification rate is calculated for the trained model. Traditionally, this is calculated as:

$$error = (correct – N) / N$$

- Where error is the misclassification rate, correct are the number of training instance predicted correctly by the model and N is the total number of training instances. For example, if the model predicted 78 of 100 training instances correctly the error or misclassification rate would be (78-100)/100 or 0.22.

- This is modified to use the weighting of the training instances:

$$error = sum(w(i) * terror(i)) / sum(w)$$

- Which is the weighted sum of the misclassification rate, where w is the weight for training instance i and terror is the prediction error for training instance i which is 1 if misclassified and 0 if correctly classified.

- For example, if we had 3 training instances with the weights 0.01, 0.5 and 0.2. The predicted values were -1, -1 and -1, and the actual output variables in the instances were -1, 1 and -1, then the terrors would be 0, 1, and 0. The misclassification rate would be calculated as:

$$error = (0.01*0 + 0.5*1 + 0.2*0) / (0.01 + 0.5 + 0.2)$$

or

$$error = 0.704$$

- A stage value is calculated for the trained model which provides a weighting for any predictions that the model makes. The stage value for a trained model is calculated as follows:

$$stage = ln((1-error) / error)$$

- Where stage is the stage value used to weight predictions from the model, ln() is the natural logarithm and error is the misclassification error for the model. The effect of the stage weight is that more accurate models have more weight or contribution to the final prediction.

- The training weights are updated giving more weight to incorrectly predicted instances, and less weight to correctly predicted instances.

- For example, the weight of one training instance (w) is updated using:

$$w = w * \exp(stage * terror)$$

- rate for the weak classifier and terror is the error the weak classifier made predicting the output variable for the training instance, evaluated as:

$$terror = 0 \text{ if}(y == p), \text{ otherwise } 1$$

- Where y is the output variable for the training instance and p is the prediction from the weak learner.

- This has the effect of not changing the weight if the training instance was classified correctly and making the weight slightly larger if the weak learner misclassified the instance.

3.4.2.3 AdaBoost Ensemble

- Weak models are added sequentially, trained using the weighted training data.

- The process continues until a pre-set number of weak learners have been created (a user parameter) or no further improvement can be made on the training dataset.

- Once completed, you are left with a pool of weak learners each with a stage value.

3.4.2.4 Making Predictions with AdaBoost

- Predictions are made by calculating the weighted average of the weak classifiers.

- For a new input instance, each weak learner calculates a predicted value as either +1.0 or -1.0. The predicted values are weighted by each weak learners stage value. The prediction for the ensemble model is taken as a the sum of the

weighted predictions. If the sum is positive, then the first class is predicted, if negative the second class is predicted.

- For example, 5 weak classifiers may predict the values 1.0, 1.0, -1.0, 1.0, -1.0. From a majority vote, it looks like the model will predict a value of 1.0 or the first class. These same 5 weak classifiers may have the stage values 0.2, 0.5, 0.8, 0.2 and 0.9 respectively. Calculating the weighted sum of these predictions results in an output of -0.8, which would be an ensemble prediction of -1.0 or the second class.

3.4.2.5 Data Preparation for AdaBoost

This section lists some heuristics for best preparing your data for AdaBoost.

- **Quality Data**: Because the ensemble method continues to attempt to correct misclassifications in the training data, you need to be careful that the training data is of a high-quality.
- **Outliers**: Outliers will force the ensemble down the rabbit hole of working hard to correct for cases that are unrealistic. These could be removed from the training dataset.
- **Noisy Data**: Noisy data, specifically noise in the output variable can be problematic. If possible, attempt to isolate and clean these from your training dataset.

3.4.2.6 Parameters

- **base_estimator** : object, optional (default=DecisionTreeRegressor)
  - The base estimator from which the boosted ensemble is built. Support for sample weighting is required.
- **n_estimators** : integer, optional (default=50)
  - The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early.

- **learning_rate** : float, optional (default=1.)
    - Learning rate shrinks the contribution of each regressor by learning_rate. There is a trade-off between learning_rate and n_estimators.
- **loss** : {'linear', 'square', 'exponential'}, optional (default='linear')
    - The loss function to use when updating the weights after each boosting iteration.
- **random_state** : int, RandomState instance or None, optional (default=None)
    - If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random.

3.4.2.7 Attributes

- **estimators_** : list of classifiers
    - The collection of fitted sub-estimators.
- **estimator_weights_** : array of floats
    - Weights for each estimator in the boosted ensemble.
- **estimator_errors_** : array of floats
    - Regression error for each estimator in the boosted ensemble.
- **feature_importances_** : array of shape = [n_features]
    - The feature importances if supported by the base_estimator.

3.4.2.8 Methods

- fit (X,y[sample_weight]) : Build a boosted regressor from the training set (X, y).
    - get_params([deep]) : Get parameters for this estimator.
- Predict(X) : Predict regression value for X.

- Score (X,y[Sample_weight]) : Returns the coefficient of determination R^2 of the prediction.

- set_params(**params) : Set the parameters of this estimator.
  - Staged_Predict(X) : Return staged predictions for X.
- Staged_score(X,y[sample_weight]) : Return staged scores for X, y.

3.4.3 Bagging Regressor

3.4.3.1 Introduction

- Bootstrap Aggregation (or Bagging for short), is a simple and very powerful ensemble method.
- An ensemble method is a technique that combines the predictions from multiple machine learning algorithms together to make more accurate predictions than any individual model.
- Bootstrap Aggregation is a general procedure that can be used to reduce the variance for those algorithm that have high variance. An algorithm that has high variance are decision trees, like classification and regression trees (CART).
- Decision trees are sensitive to the specific data on which they are trained. If the training data is changed (e.g. a tree is trained on a subset of the training data) the resulting decision tree can be quite different and in turn the predictions can be quite different.
- Bagging is the application of the Bootstrap procedure to a high-variance machine learning algorithm, typically decision trees.
- Let's assume we have a sample dataset of 1000 instances (x) and we are using the CART algorithm. Bagging of the CART algorithm would work as follows.
    - Create many (e.g. 100) random sub-samples of our dataset with replacement.
    - Train a CART model on each sample.
    - Given a new dataset, calculate the average prediction from each model.
- For example, if we had 5 bagged decision trees that made the following class predictions for a in input sample: blue, blue, red, blue and red, we would take the most frequent class and predict blue.
- When bagging with decision trees, we are less concerned about individual trees overfitting the training data. For this reason and for efficiency, the individual decision trees are grown deep (e.g. few training samples at each leaf-node of the tree) and the trees are not pruned. These trees will have both high variance and low bias. These are important characterize of sub-models when combining predictions using bagging.

- The only parameters when bagging decision trees is the number of samples and hence the number of trees to include. This can be chosen by increasing the number of trees on run after run until the accuracy begins to stop showing improvement (e.g. on a cross validation test harness). Very large numbers of models may take a long time to prepare, but will not overfit the training data.
- Just like the decision trees themselves, Bagging can be used for classification and regression problems.

3.4.3.2 Variable Importance

- As the Bagged decision trees are constructed, we can calculate how much the error function drops for a variable at each split point.

- In regression problems this may be the drop in sum squared error and in classification this might be the Gini score.

- These drops in error can be averaged across all decision trees and output to provide an estimate of the importance of each input variable. The greater the drop when the variable was chosen, the greater the importance.

- These outputs can help identify subsets of input variables that may be most or least relevant to the problem and suggest at possible feature selection experiments you could perform where some features are removed from the dataset.

3.4.3.3 Parameters

- base_estimator : object or None, optional (default=None)
    - The base estimator to fit on random subsets of the dataset. If None, then the base estimator is a decision tree.
- n_estimators : int, optional (default=10)
    - The number of base estimators in the ensemble.
- max_samples : int or float, optional (default=1.0)
    - The number of samples to draw from X to train each base estimator.
        - If int, then draw max_samples samples.
        - If float, then draw max_samples * X.shape[0] samples.

- max_features : int or float, optional (default=1.0)
    - The number of features to draw from X to train each base estimator.
        - If int, then draw max_features features.
        - If float, then draw max_features * X.shape[1] features.
- bootstrap : boolean, optional (default=True)
    - Whether samples are drawn with replacement.
- bootstrap_features : boolean, optional (default=False)
    - Whether features are drawn with replacement.
- oob_score : bool
    - Whether to use out-of-bag samples to estimate the generalization error.
- warm_start : bool, optional (default=False)
    - When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new ensemble.
- n_jobs : int, optional (default=1)
    - The number of jobs to run in parallel for both fit and predict. If -1, then the number of jobs is set to the number of cores.
- random_state : int, RandomState instance or None, optional (default=None)
    - If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random.
- verbose : int, optional (default=0)
    - Controls the verbosity of the building process.

3.4.3.4 Attributes

- estimators_ : list of estimators
    - The collection of fitted sub-estimators.
- estimators_samples_ : list of arrays
    - The subset of drawn samples (i.e., the in-bag samples) for each base estimator. Each subset is defined by a boolean mask.
- estimators_features_ : list of arrays
    - The subset of drawn features for each base estimator.
- oob_score_ : float
    - Score of the training dataset obtained using an out-of-bag estimate.
- oob_prediction_ : array of shape = [n_samples]
    - Prediction computed with out-of-bag estimate on the training set. If n_estimators is small it might be possible that a data point was never left out during the bootstrap. In this case, oob_prediction_ might contain NaN.

3.4.3.5 Methods

- fit (X,y[sample_weight]) : Build a boosted regressor from the training set (X, y).
- get_params([deep]) : Get parameters for this estimator.
- Predict(X) : Predict regression value for X.
- Score (X,y[Sample_weight]) : Returns the coefficient of determination R^2 of the prediction.
- set_params(**params) : Set the parameters of this estimator.

3.4.4 Gradient Boost Regressor

3.4.4.1 Introduction

- **Gradient boosting** is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

- It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

- The idea of gradient boosting originated in the observation by Leo Breiman that boosting can be interpreted as an optimization algorithm on a suitable cost function. Explicit regression gradient boosting algorithms were subsequently developed by Jerome H. Friedman simultaneously with the more general functional gradient boosting perspective of Llew Mason, Jonathan Baxter, Peter Bartlett and Marcus Frean.

- The latter two papers introduced the view of boosting algorithms as iterative functional gradient descent algorithms.

- That is, algorithms that optimize a cost function over function space by iteratively choosing a function (weak hypothesis) that points in the negative gradient direction.

- This functional gradient view of boosting has led to the development of boosting algorithms in many areas of machine learning and statistics beyond regression and classification.

3.4.4.2 Working of Gradient Boosting

- Gradient boosting involves three elements:

    - A loss function to be optimized.
    - A weak learner to make predictions.
    - An additive model to add weak learners to minimize the loss function.

- **Loss Function**
  - The loss function used depends on the type of problem being solved.
  - It must be differentiable, but many standard loss functions are supported and you can define your own.
  - For example, regression may use a squared error and classification may use logarithmic loss.
  - A benefit of the gradient boosting framework is that a new boosting algorithm does not have to be derived for each loss function that may want to be used, instead, it is a generic enough framework that any differentiable loss function can be used.
- **Weak Learner**
  - Decision trees are used as the weak learner in gradient boosting.
  - Specifically regression trees are used that output real values for splits and whose output can be added together, allowing subsequent models outputs to be added and "correct" the residuals in the predictions.
  - Trees are constructed in a greedy manner, choosing the best split points based on purity scores like Gini or to minimize the loss.
  - Initially, such as in the case of AdaBoost, very short decision trees were used that only had a single split, called a decision stump. Larger trees can be used generally with 4-to-8 levels.
  - It is common to constrain the weak learners in specific ways, such as a maximum number of layers, nodes, splits or leaf nodes.
  - This is to ensure that the learners remain weak but can still be constructed in a greedy manner.

- **Additive Model**
    - Trees are added one at a time, and existing trees in the model are not changed.
    - A gradient descent procedure is used to minimize the loss when adding trees.
    - Traditionally, gradient descent is used to minimize a set of parameters, such as the coefficients in a regression equation or weights in a neural network. After calculating error or loss, the weights are updated to minimize that error.
    - Instead of parameters, we have weak learner sub-models or more specifically decision trees. After calculating the loss, to perform the gradient descent procedure, we must add a tree to the model that reduces the loss (i.e. follow the gradient). We do this by parameterizing the tree, then modify the parameters of the tree and move in the right direction by (reducing the residual loss.

3.4.4.3 Parameters

- **loss** : {'ls', 'lad', 'huber', 'quantile'}, optional (default='ls')
    - loss function to be optimized. 'ls' refers to least squares regression. 'lad' (least absolute deviation) is a highly robust loss function solely based on order information of the input variables. 'huber' is a combination of the two. 'quantile' allows quantile regression (use alpha to specify the quantile).
- **learning_rate** : float, optional (default=0.1)
    - learning rate shrinks the contribution of each tree by learning_rate. There is a trade-off between learning_rate and n_estimators.
- **n_estimators** : int (default=100)
    - The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.
- **max_depth** : integer, optional (default=3)

- maximum depth of the individual regression estimators. The maximum depth limits the number of nodes in the tree. Tune this parameter for best performance; the best value depends on the interaction of the input variables.

3.4.4.4 Attributes

- **feature_importances_** : array, shape = [n_features]
  - The feature importances (the higher, the more important the feature).
- **oob_improvement_** : array, shape = [n_estimators]
  - The improvement in loss (= deviance) on the out-of-bag samples relative to the previous iteration. oob_improvement_[0] is the improvement in loss of the first stage over the init estimator.
- **train_score_** : array, shape = [n_estimators]
  - The i-th score train_score_[i] is the deviance (= loss) of the model at iteration i on the in-bag sample. If subsample == 1 this is the deviance on the training data.
- **loss_** : LossFunction
  - The concrete LossFunction object.
- **init** : BaseEstimator
  - The estimator that provides the initial predictions. Set via the init argument or loss.init_estimator.
- **estimators_** : ndarray of DecisionTreeRegressor, shape = [n_estimators, 1]
  - The collection of fitted sub-estimators.

3.4.5 Bayesian Ridge

3.4.5.1 Introduction

- In statistics, Bayesian linear regression is an approach to linear regression in which the statistical analysis is undertaken within the context of Bayesian inference.
- When the regression model has errors that have a normal distribution, and if a particular form of prior distribution is assumed, explicit results are available for the posterior probability distributions of the model's parameters.

3.4.5.2 Parameters

- **n_iter** : int, optional
  - Maximum number of iterations. Default is 300.
- **tol** : float, optional
  - Stop the algorithm if w has converged. Default is 1.e-3.
- **alpha_1** : float, optional
  - Hyper-parameter : shape parameter for the Gamma distribution prior over the alpha parameter. Default is 1.e-6
- **alpha_2** : float, optional
  - Hyper-parameter : inverse scale parameter (rate parameter) for the Gamma distribution prior over the alpha parameter. Default is 1.e-6.

3.4.5.3 Attributes

- **coef_** : array, shape = (n_features)
  - Coefficients of the regression model (mean of distribution)
- **alpha_** : float
  - estimated precision of the noise.
- **lambda_** : float
  - estimated precision of the weights.
- **sigma_** : array, shape = (n_features, n_features)
  - estimated variance-covariance matrix of the weights
- **scores_** : float
  - if computed, value of the objective function (to be maximized).

4. DESIGN

4.1 INTRODUCTION

Software design is the process of implementing **software** solutions to one or more sets of problems. This section consists data flow diagrams, module design and organization of the project.

4.2 DATA FLOW DIAGRAM AND MODULE DESIGN

A **data flow diagram** (**DFD**) is a graphical representation of the "flow" of data through an information system, modelling its process aspects.

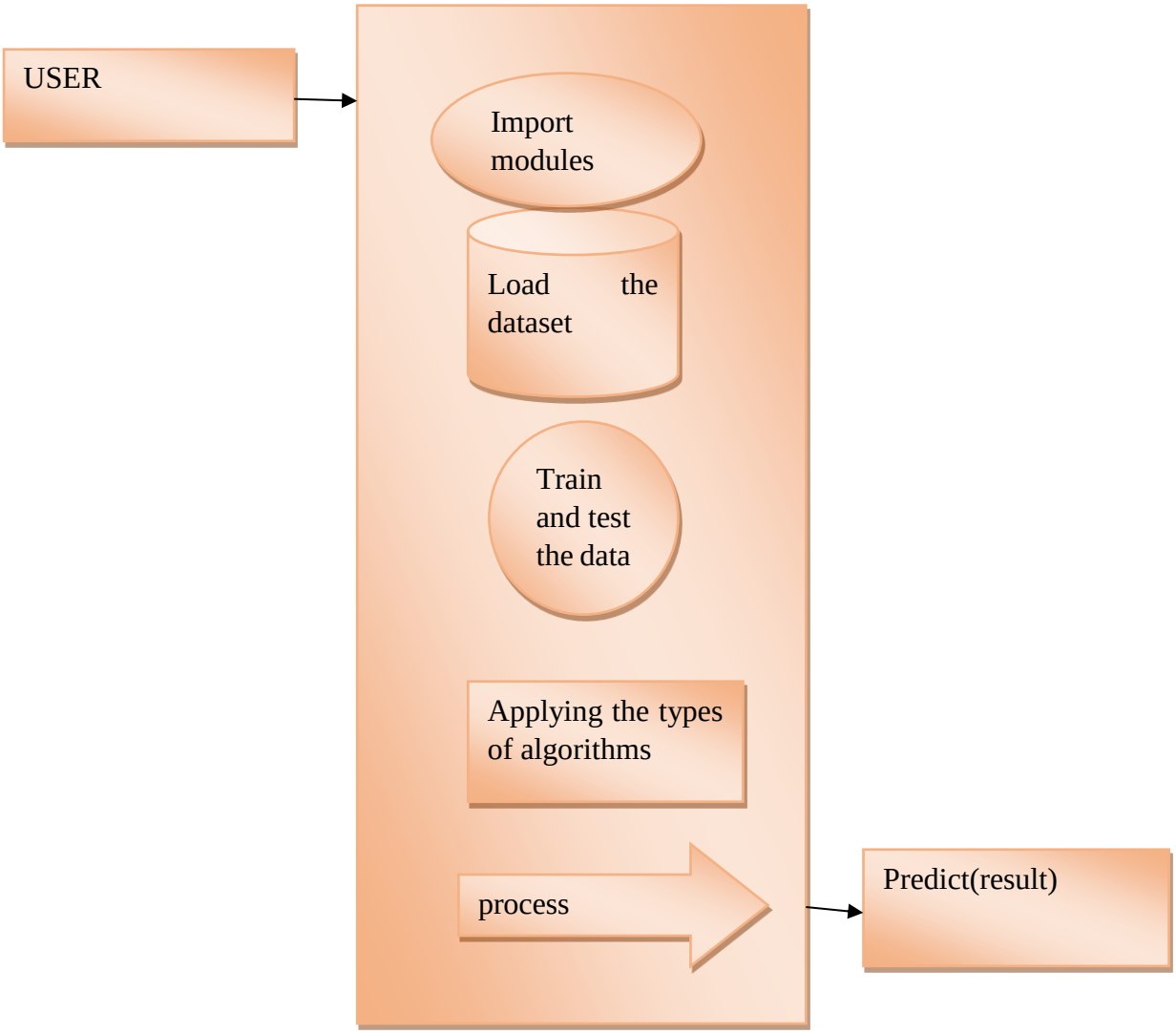


Figure.3: Data Flow Diagram.

The module design steps are as follows

1. Input Data set is loaded.
2. Data preprocessing is performed for feature selection and feature variability i.e. permutation among rows etc. are made for more consistency in data.
3. Then the data is mined using the regression equation formed by relations between features and prediction algorithm is trained.
4. The trained algorithm is tested on test data set.
5. The information obtained is the predicted value.
6. Calculate the error values and accuracy percentage.

5. IMPLEMENTATION AND RESULTS

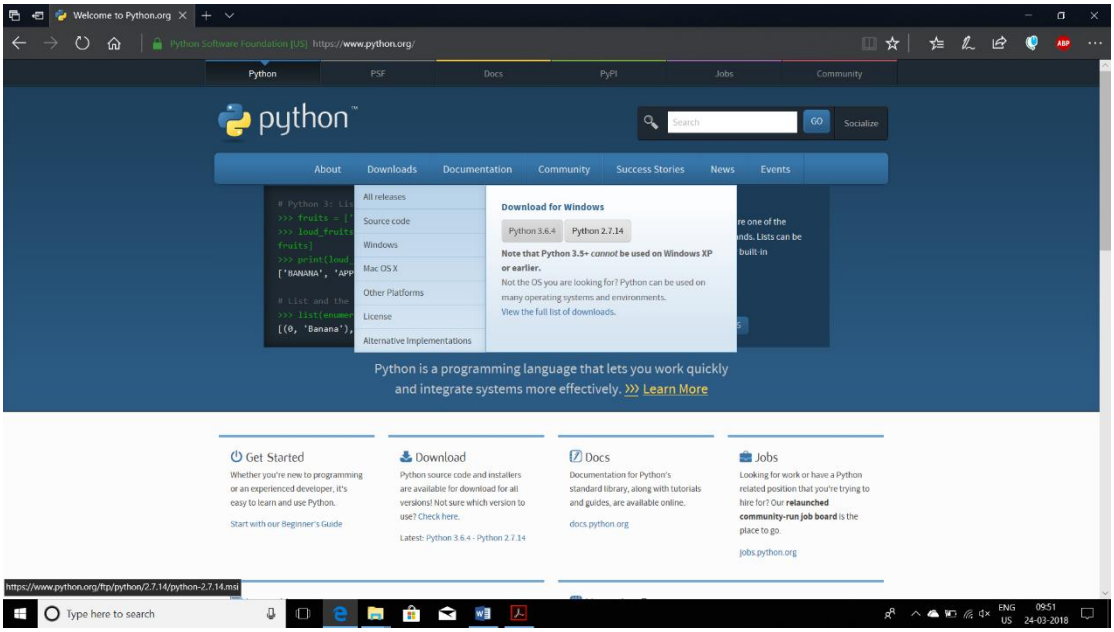
5.1 EXPLANATION OF KEY FUNCTIONS

- pd.read_csv(filename,header = None)
  - This is used to read a comma separated variable(CSV) file using the pandas library with no header.
- numpy.random.permutation
  - Randomly permute a sequence or return a permuted range.
  - If *x* is a multi-dimensional array, it is only shuffled along its first index.
- In zip ()
  - The zip () function take iterables (can be zero or more), makes iterator that aggregates elements based on the iterables passed, and returns an iterator of tuples.
- For
  - The for statement in Python differs a bit from what you may be used to in C or Pascal.
  - Rather than always iterating over an arithmetic progression of numbers (like in Pascal), or giving the user the ability to define both the iteration step and halting condition (as C), Python's for statement iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence.
- model_selection.train_test_split
  - Split arrays or matrices into random train and test subsets
  - Quick utility that wraps input validation and next(ShuffleSplit().split(X, y)) and application to input data into a single call for splitting (and optionally subsampling) data in a oneliner.

5.2 METHOD OF IMPLEMENTATION

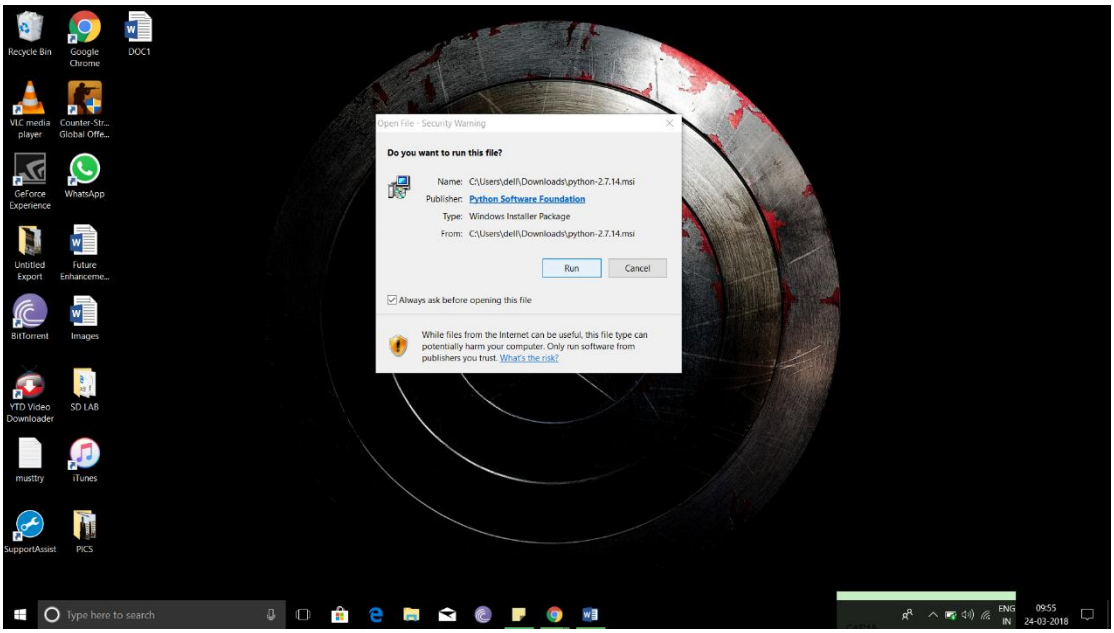
5.2.1 Installation

- Go to the given link **https://www.python.org/**
- Click on downloads and download python 2.7.14

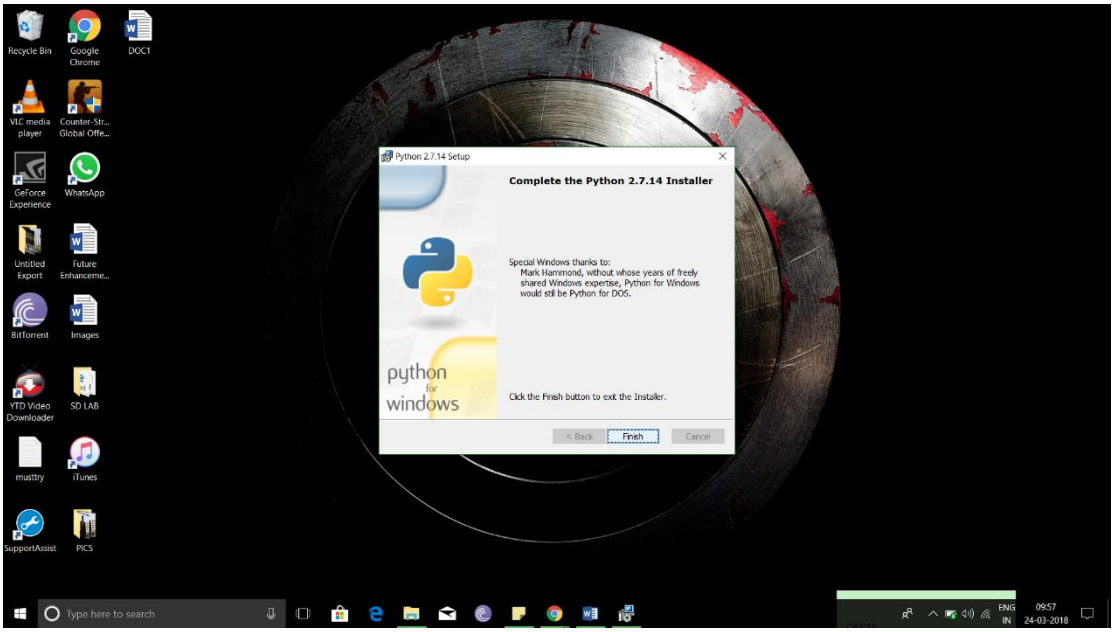


Screen.1: Python download

- After downloading the software run and install it

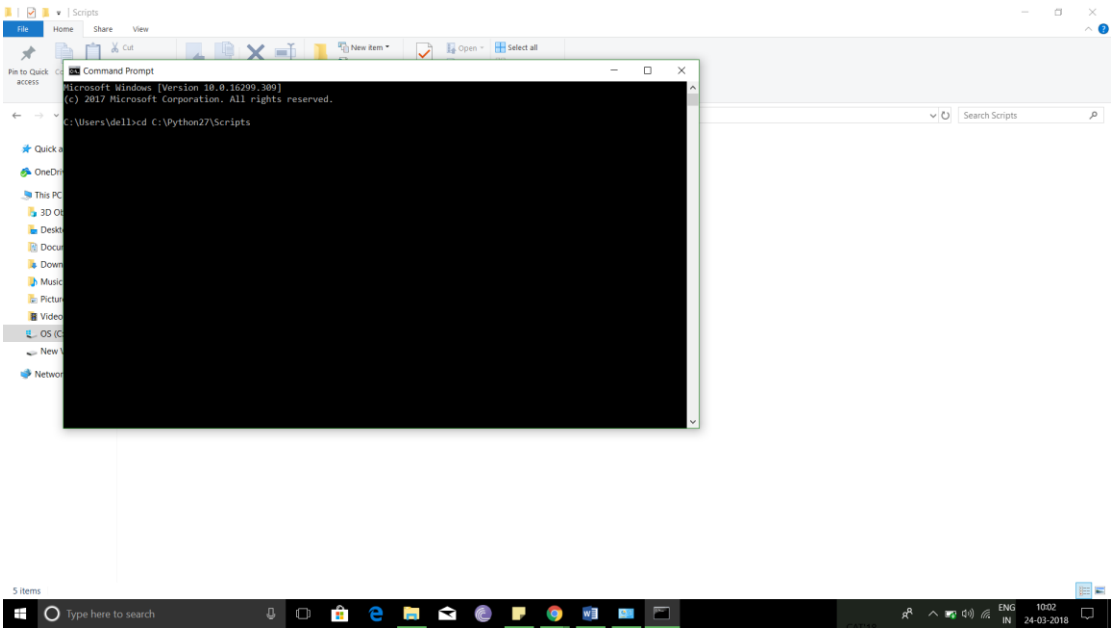


Screen.2: Running Python



Screen.3: Python installation

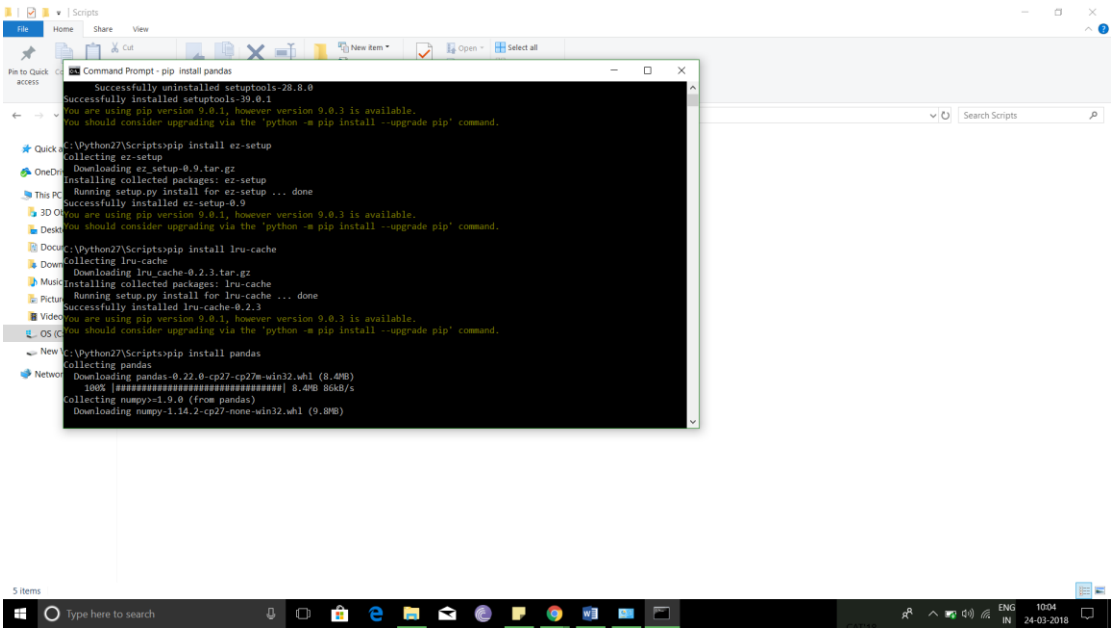
- Set the paths at Windows Environment variables
- Open the command prompt and set the path to C:\Python27 by using cd command



Screen.4: Setting paths in cmd.

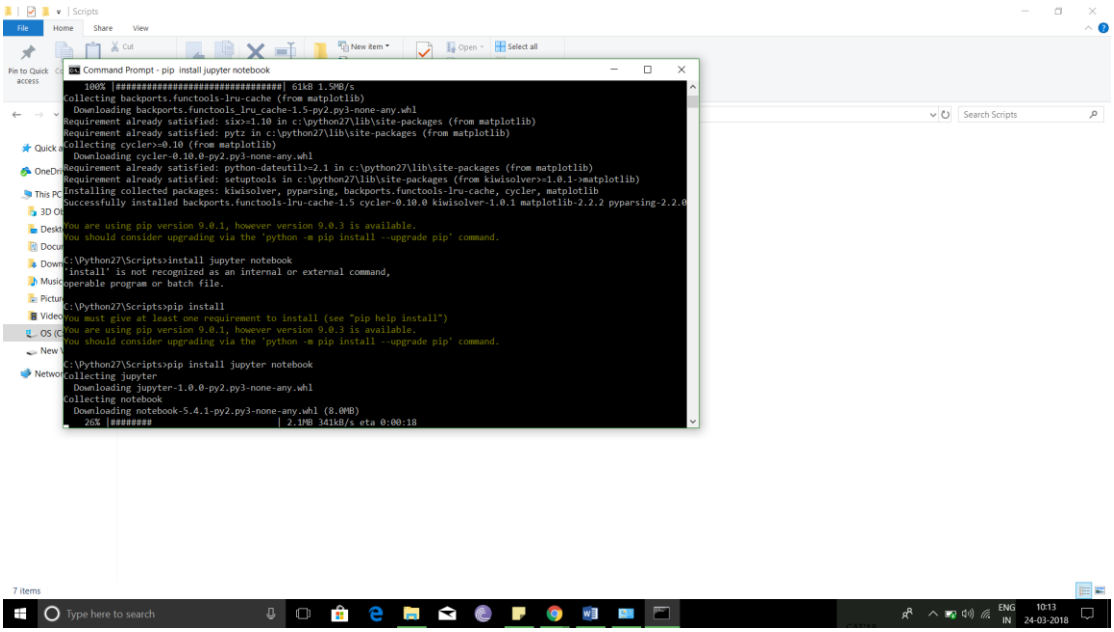
- Enter the following commands to install libraries of python
  - pip install –upgrade setuptools
  - pip install ez-setup
  - pip install lru-cache

- pip install pandas
- pip install sklearn
- pip install scipy



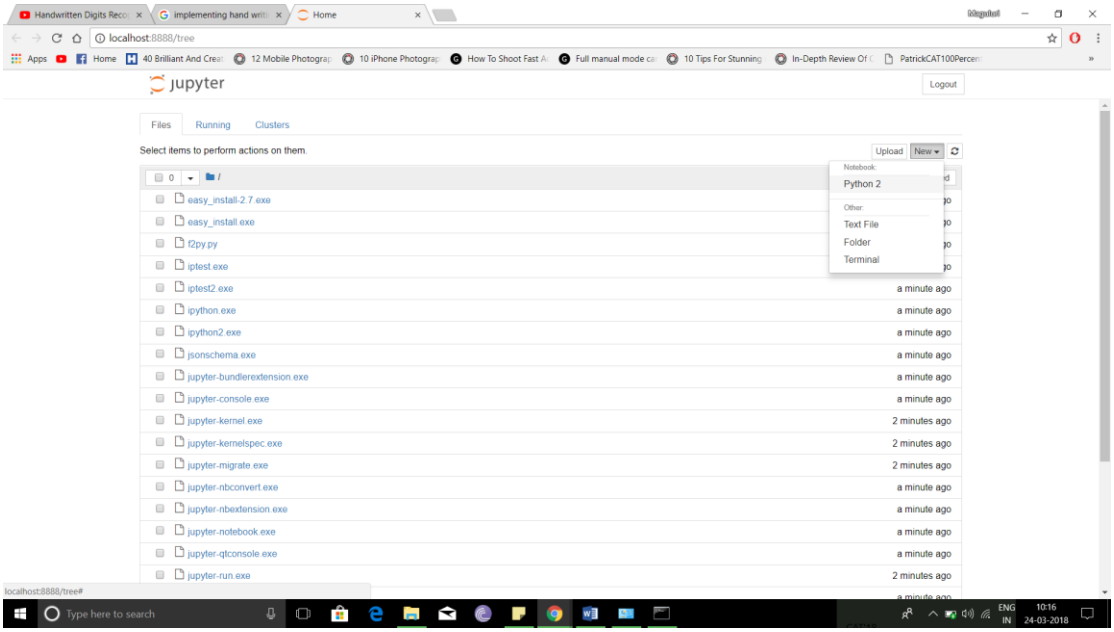
Screen.5: Installing libraries

- ❖ Then install jupyter notebook by entering the following command
  - pip install jupyter notebook



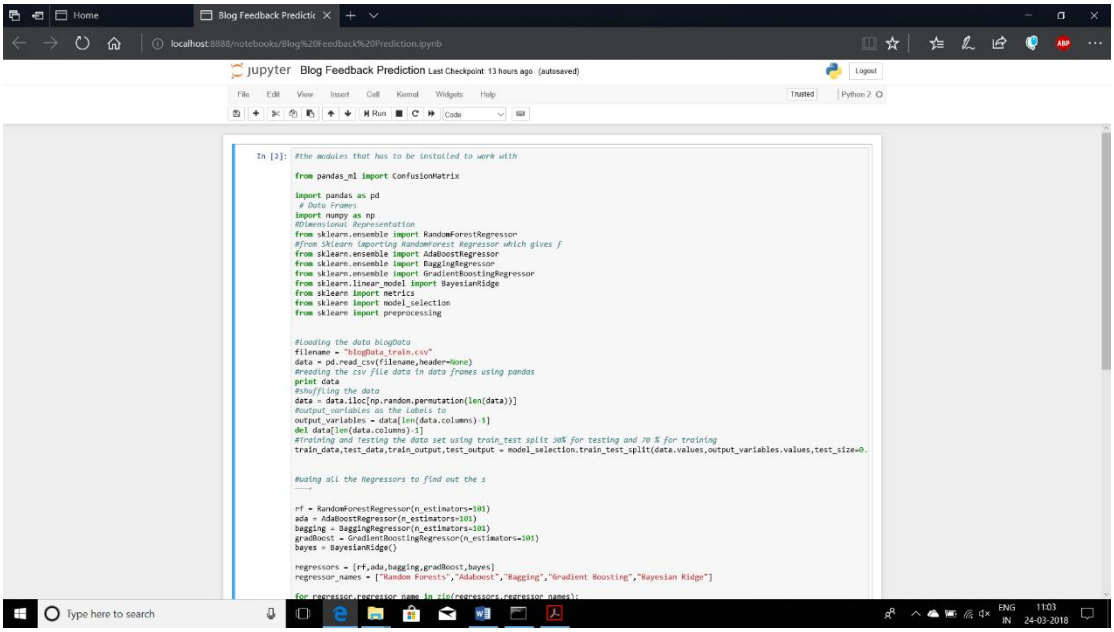
Screen.6: Installing Jupyter notebook.

- After installation to open the jupyter notebook the following command is used
  - Jupyter notebook
- Then click on new->Python 27 in the jupyter notebook



Screen.7: Creating new python27 notebook.

- Type and run the code and we can even save it as notebook.



Screen.8: Working in the notebook.

5.2.2 Code Implementation

- First, the libraries are imported
  - Import numpy as np
    - This command is used to import numerical python library and to rename it as np for our program usage.
  - Import pandas as pd
    - This command is used to import pandas library and to rename it as pd for our program usage.
- Then the data file is loaded, and the header is nullified
- The second major operation is permuting the entire data set randomly to achieve better consistency
- Then the final column is removed and stored separately as target variable.
- The third major step is to break down the data set into train and test data in the ratio of 70:30
- The ensemble and boosting algorithms are then called in and renamed as per our requirement.
- In the fourth step the algorithms are fit over the train data followed by test data
- In the fifth and final step the error calculation or accuracy of each algorithm is carried out by various methods.

5.3 CODE

```
#the modules that has to be installed to work with

from pandas_ml import ConfusionMatrix

import pandas as pd

 # Data Frames

import numpy as np

#Dimensional Representation

from sklearn.ensemble import RandomForestRegressor

#from Sklearn importing RandomForest Regressor which gives f

from sklearn.ensemble import AdaBoostRegressor

from sklearn.ensemble import BaggingRegressor

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.linear_model import BayesianRidge

from sklearn import metrics

from sklearn import model_selection

from sklearn import preprocessing




#loading the data blogData

filename = "blogData_train.csv"

data = pd.read_csv(filename,header=None)

#reading the csv file data in data frames using pandas
```

```
print data

#shuffling the data

data = data.iloc[np.random.permutation(len(data))]

#output_variables as the labels to

output_variables = data[len(data.columns)-1]

del data[len(data.columns)-1]

#Training and Testing the data set using train_test split 30% for testing and 70 % for training

train_data,test_data,train_output,test_output = model_selection.train_test_split(data.values,output_variables.values,test_size=0.3)



#using all the Regressors to find out the s



rf = RandomForestRegressor(n_estimators=101)

ada = AdaBoostRegressor(n_estimators=101)

bagging = BaggingRegressor(n_estimators=101)

gradBoost = GradientBoostingRegressor(n_estimators=101)

bayes = BayesianRidge()


regressors = [rf,ada,bagging,gradBoost,bayes]
```

```
regressor_names = ["Random Forests","Adaboost","Bagging","Gradient Boosting","Bayesian Ridge"]


for regressor,regressor_name in zip(regressors,regressor_names):


    regressor.fit(train_data,train_output)

    predicted_values = regressor.predict(test_data)

    print test_data




    print ("---------------------------------\n")

    print ("Mean Absolute Error for ",regressor_name," : ",metrics.mean_absolute_error(test_output,predicted_values))


    print ("Median Absolute Error for ",regressor_name, " : ",metrics.median_absolute_error(test_output,predicted_values))

    print ("Mean Squared Error for ",regressor_name, " : ",metrics.mean_squared_error(test_output,predicted_values))

    print ("R2 score for ",regressor_name, " : ",metrics.r2_score(test_output,predicted_values))

    print ("---------------------------------\n")
```

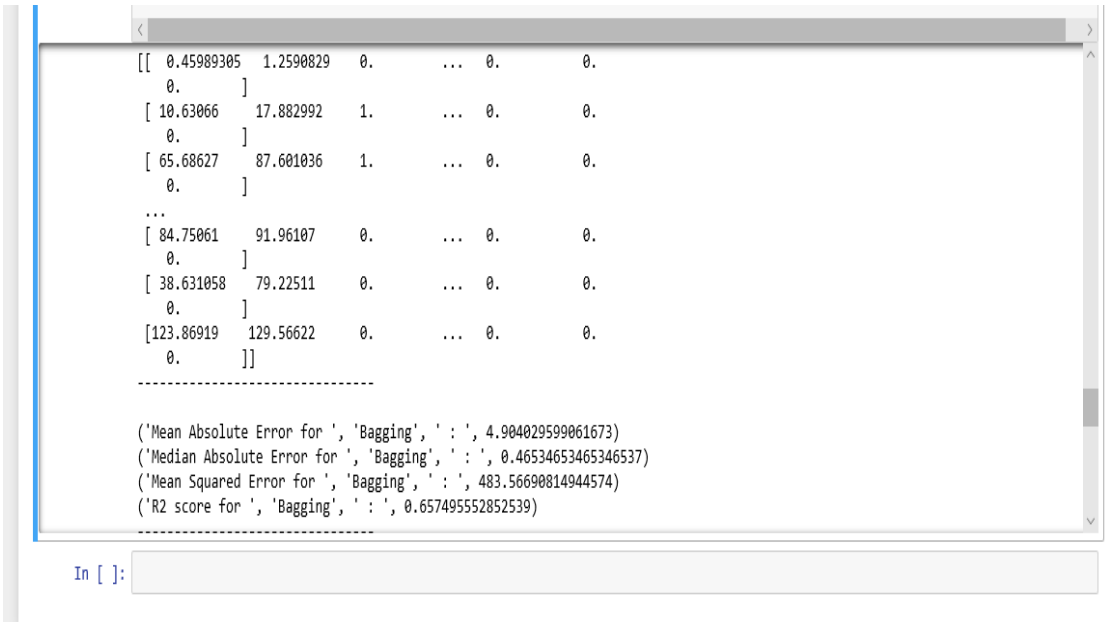
5.4 OUTPUT

```
[[  0.45989305   1.2590829    0.         ...   0.          0.
    0.        ]
 [ 10.63066     17.882992     1.         ...   0.          0.
    0.        ]
 [ 65.68627     87.601036     1.         ...   0.          0.
    0.        ]
 ...
 [ 84.75061     91.96107      0.         ...   0.          0.
    0.        ]
 [ 38.631058    79.22511      0.         ...   0.          0.
    0.        ]
 [123.86919    129.56622      0.         ...   0.          0.
    0.        ]]
--------------------------------

('Mean Absolute Error for ', 'Random Forests', ' : ', 4.898503181365586)
('Median Absolute Error for ', 'Random Forests', ' : ', 0.46534653465346537)
('Mean Squared Error for ', 'Random Forests', ' : ', 495.3228668968211)
('R2 score for ', 'Random Forests', ' : ', 0.6491689529888987)
--------------------------------

In [ ]:
```

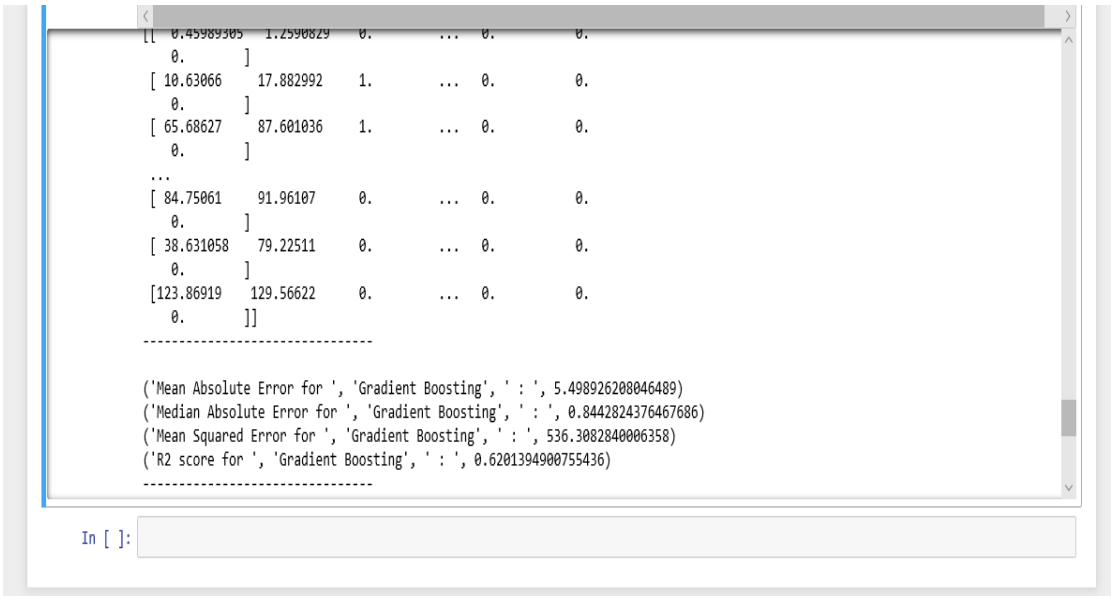
Screen.9: Output for Random Forest Regressor.

```
[[  0.45989305   1.2590829    0.         ...   0.          0.
    0.        ]
 [ 10.63066     17.882992     1.         ...   0.          0.
    0.        ]
 [ 65.68627     87.601036     1.         ...   0.          0.
    0.        ]
 ...
 [ 84.75061     91.96107      0.         ...   0.          0.
    0.        ]
 [ 38.631058    79.22511      0.         ...   0.          0.
    0.        ]
 [123.86919    129.56622      0.         ...   0.          0.
    0.        ]]
--------------------------------

('Mean Absolute Error for ', 'Adaboost', ' : ', 8.421607745652281)
('Median Absolute Error for ', 'Adaboost', ' : ', 4.289189689195856)
('Mean Squared Error for ', 'Adaboost', ' : ', 643.8323801758361)
('R2 score for ', 'Adaboost', ' : ', 0.5439815055342692)

In [ ]:
```

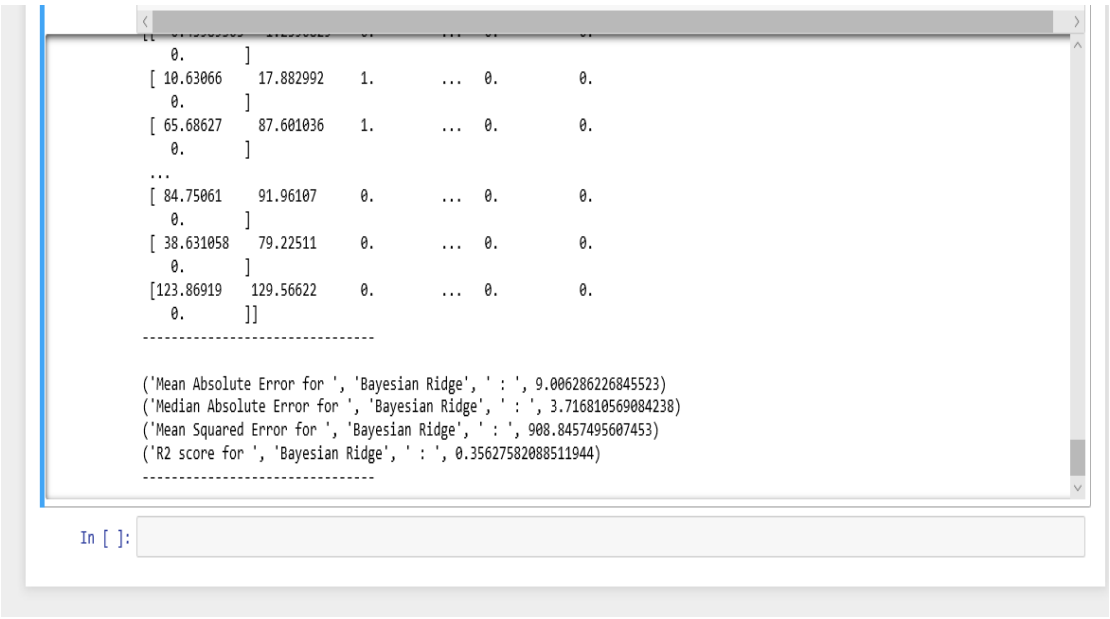
Screen.10: Output for AdaBoost Regressor.



Screen.11: Output for Bagging Regressor.



Screen.12: Output for Gradient Boosting.



Screen.13: Output for Bayesian Ridge.

5.5 RESULT ANALYSIS

5.5.1 Introduction

- Time series prediction performance measures provide a summary of the skill and capability of the forecast model that made the predictions.
- There are many different performance measures to choose from. It can be confusing to know which measure to use and how to interpret the results.
- Time series generally focus on the prediction of real values, called regression problems.

5.5.2 Forecast Error (or Residual Forecast Error)

- The forecast error is calculated as the expected value minus the predicted value.
- This is called the residual error of the prediction.
- The forecast error can be calculated for each prediction, providing a time series of forecast errors.

5.5.3 Mean Forecast Error (or Forecast Bias)

- Mean forecast error is calculated as the average of the forecast error values.
- Forecast errors can be positive and negative. This means that when the average of these values is calculated, an ideal mean forecast error would be zero.
- A mean forecast error value other than zero suggests a tendency of the model to over forecast (positive error) or under forecast (negative error). As such, the mean forecast error is also called the forecast bias.
- The forecast error can be calculated directly as the mean of the forecast values
- Forecast errors can be positive and negative. This means that when the average of these values is calculated, an ideal mean forecast error would be zero.

- A mean forecast error value other than zero suggests a tendency of the model to over forecast (positive error) or under forecast (negative error). As such, the mean forecast error is also called the forecast bias.

- The forecast error can be calculated directly as the mean of the forecast values

5.5.4 Mean Absolute Error

- The mean absolute error, or MAE, is calculated as the average of the forecast error values, where all of the forecast values are forced to be positive.

- Forcing values to be positive is called making them absolute. This is signified by the absolute function *abs()* or shown mathematically as two pipe characters around the value: |*value*|.

- Where *abs()* makes values positive, *forecast_error* is one or a sequence of forecast errors, and *mean()* calculates the average value.

- We can use the mean_absolute_error() function from the scikit-learn library to calculate the mean absolute error for a list of predictions. The example below demonstrates this function.

- These error values are in the original units of the predicted values. A mean absolute error of zero indicates no error.

5.5.5 Mean Squared Error

- The mean squared error, or MSE, is calculated as the average of the squared forecast error values. Squaring the forecast error values forces them to be positive; it also has the effect of putting more weight on large errors.

- Very large or outlier forecast errors are squared, which in turn has the effect of dragging the mean of the squared forecast errors out resulting in a larger mean squared error score. In effect, the score gives worse performance to those models that make large wrong forecasts.

- The error values are in squared units of the predicted values. A mean squared error of zero indicates perfect skill, or no error.

5.5.6 Root Mean Squared Error

- The mean squared error described above is in the squared units of the predictions.
- It can be transformed back into the original units of the predictions by taking the square root of the mean squared error score. This is called the root mean squared error, or RMSE.
- The RMES error values are in the same units as the predictions. As with the mean squared error, an RMSE of zero indicates no error.

6. TESTING AND VALIDATION

6.1 INTRODUCTION

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

According to ANSI/IEEE 1059 standard, Testing can be defined as - A process of analysing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

6.1.1 Who does testing

It depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called **Unit Testing**. In most cases, the following professionals are involved in testing a system within their respective capacities:

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

Different companies have different designations for people who test the software based on their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, QA Analyst, etc.

It is not possible to test the software at any time during its cycle. The next two sections state when testing should be started and when to end it during the SDLC.

6.1.2 When to start testing

An early start to testing reduces the cost and time to rework and produce error-free software that is delivered to the client. However in Software Development Life Cycle (SDLC), testing can be started from the Requirements Gathering phase and continued till the deployment of the software. It also depends on the development model that is being used. For example, in the Waterfall model, formal testing is conducted in the testing phase; but in the incremental model, testing is performed at the end of every increment/iteration and the whole application is tested at the end.

Testing is done in different forms at every phase of SDLC:

- During the requirement gathering phase, the analysis and verification of requirements are also considered as testing.
- Reviewing the design in the design phase with the intent to improve the design is also considered as testing.
- Testing performed by a developer on completion of the code is also categorized as testing.

6.1.3 When to stop testing

It is difficult to determine when to stop testing, as testing is a never-ending process and no one can claim that a software is 100% tested. The following aspects are to be considered for stopping the testing process:

- Testing Deadlines
- Completion of test case execution
- Completion of functional and code coverage to a certain point
- Bug rate falls below a certain level and no high-priority bugs are identified
- Management decision

6.1.4 Verification & Validation

These two terms are very confusing for most people, who use them interchangeably. The following table highlights the differences between verification and validation.

S.N.	Verification	Validation
1	Verification addresses the concern: "Are you building it right?"	Validation addresses the concern: "Are you building the right thing?"
2	Ensures that the software system meets all the functionality.	Ensures that the functionalities meet the intended behavior.
3	Verification takes place first and includes the checking for documentation, code, etc.	Validation occurs after verification and mainly involves the checking of the overall product.
4	Done by developers.	Done by testers.
5	It has static activities, as it includes collecting reviews, walkthroughs, and inspections to verify a software.	It has dynamic activities, as it includes executing the software against the requirements.
6	It is an objective process and no subjective decision should be needed to verify a software.	It is a subjective process and involves subjective decisions on how well a software works.

6.2 TYPES OF TESTING

6.2.1 Manual Testing

Manual testing includes testing a software manually, i.e., without using any automated tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behaviour or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.

Testers use test plans, test cases, or test scenarios to test a software to ensure the completeness of testing. Manual testing also includes exploratory testing, as testers explore the software to identify errors in it.

6.2.2 Automation Testing

Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves automation of a manual process. Automation Testing is used to re-run the test scenarios that were performed manually, quickly, and repeatedly.

Apart from regression testing, automation testing is also used to test the application from load, performance, and stress point of view. It increases the test coverage, improves accuracy, and saves time and money in comparison to manual testing.

6.2.2.1 What is Automate?

It is not possible to automate everything in a software. The areas at which a user can make transactions such as the login form or registration forms, any area where large number of users can access the software simultaneously should be automated.

Furthermore, all GUI items, connections with databases, field validations, etc. can be efficiently tested by automating the manual process.

6.2.2.2 When to Automate?

Test Automation should be used by considering the following aspects of a software:

- Large and critical projects
- Projects that require testing the same areas frequently
- Requirements not changing frequently
- Accessing the application for load and performance with many virtual users
- Stable software with respect to manual testing
- Availability of time

6.2.2.3 How to Automate?

Automation is done by using a supportive computer language like VB scripting and an automated software application. There are many tools available that can be used to write automation scripts. Before mentioning the tools, let us identify the process that can be used to automate the testing process:

- Identifying areas within a software for automation
- Selection of appropriate tool for test automation
- Writing test scripts
- Development of test suits
- Execution of scripts
- Create result reports
- Identify any potential bug or performance issues

6.2.2.4 Software Testing Tools

The following tools can be used for automation testing:

- HP Quick Test Professional
- Selenium
- IBM Rational Functional Tester
- SilkTest
- TestComplete
- Testing Anywhere
- WinRunner
- LoadRunner
- Visual Studio Test Professional
- WATIR

6.3 METHODS OF TESTING

6.3.1 Black-Box Testing

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

The following table lists the advantages and disadvantages of black-box testing.

Advantages	Disadvantages
• Well suited and efficient for large code segments. • Code access is not required. • Clearly separates user's perspective from the developer's perspective through visibly defined roles. • Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems.	• Limited coverage, since only a selected number of test scenarios is actually performed. • Inefficient testing, due to the fact that the tester only has limited knowledge about an application. • Blind coverage, since the tester cannot target specific code segments or error-prone areas. • The test cases are difficult to design.

6.3.2 White-Box Testing

White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called **glass testing** or **open-box testing**. To perform **white-box** testing on an application, a tester needs to know the internal workings of the code.

The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

The following table lists the advantages and disadvantages of white-box testing.

Advantages	Disadvantages
• As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively. • It helps in optimizing the code. • Extra lines of code can be removed which can bring in hidden defects. • Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.	• Since a skilled tester is needed to perform white-box testing, the costs are increased. • Sometimes it is impossible to investigate every nook and corner to find out hidden errors that may create problems, as many paths will go untested. • It is difficult to maintain white-box testing, as it requires specialized tools like code analysers and debugging tools.

6.3.3 Grey-Box Testing

Grey-box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. In software testing, the phrase the more you know, the better carries a lot of weight while testing an application.

Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black-box testing, where the tester only tests the application's user interface; in grey-box testing, the tester has access to design documents and the database. Having this knowledge, a tester can prepare better test data and test scenarios while making a test plan.

Advantages	Disadvantages
• Offers combined benefits of black-box and white-box testing wherever possible. • Grey box testers don't rely on the source code; instead they rely on interface definition and functional specifications. • Based on the limited information available, a grey-box tester can design excellent test scenarios especially around communication protocols and data type handling. • The test is done from the point of view of the user and not the designer.	• Since the access to source code is not available, the ability to go over the code and test coverage is limited. • The tests can be redundant if the software designer has already run a test case. • Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested.

6.3.4 A Comparison of Testing Methods

The following table lists the points that differentiate black-box testing, grey-box testing, and white-box testing.

Black-Box Testing	Grey-Box Testing	White-Box Testing
The internal workings of an application need not be known.	The tester has limited knowledge of the internal workings of the application.	Tester has full knowledge of the internal workings of the application.
Also known as closed-box testing, data-driven testing, or functional testing.	Also known as translucent testing, as the tester has limited knowledge of the insides of the application.	Also known as clear-box testing, structural testing, or code-based testing.
Performed by end-users and by testers and developers.	Performed by end-users and by testers and developers.	Normally done by testers and developers.
Testing is based on external expectations - Internal behaviour of the application is unknown.	Testing is done based on high-level database diagrams and data flow diagrams.	Internal workings are fully known, and the tester can design test data accordingly.
It is exhaustive and the least time-consuming.	Partly time-consuming and exhaustive.	The most exhaustive and time-consuming type of testing.

Not suited for algorithm testing.	Not suited for algorithm testing.	Suited for algorithm testing.
This can only be done by trial-and-error method.	Data domains and internal boundaries can be tested, if known.	Data domains and internal boundaries can be better tested.

6.4 LEVELS OF TESTING

There are different levels during the process of testing. In this chapter, a brief description is provided about these levels.

Levels of testing include different methodologies that can be used while conducting software testing. The main levels of software testing are:

- Functional Testing
- Non-functional Testing

6.4.1 Functional Testing

This is a type of black-box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional testing of a software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

There are five steps that are involved while testing an application for functionality.

Steps	Description
I	The determination of the functionality that the intended application is meant to perform.
II	The creation of test data based on the specifications of the application.
III	The output based on the test data and the specifications of the application.
IV	The writing of test scenarios and the execution of test cases.

V	The comparison of actual and expected results based on the executed test cases.

An effective testing practice will see the above steps applied to the testing policies of every organization and hence it will make sure that the organization maintains the strictest of standards when it comes to software quality.

Unit Testing

This type of testing is performed by developers before the setup is handed over to the testing team to formally execute the test cases. Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is different from the test data of the quality assurance team.

The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

Integration Testing

Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

S.N.	Integration Testing Method
1	Bottom-up integration This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.
2	Top-down integration In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter.

In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple

tests of the complete application, preferably in scenarios designed to mimic actual situations.

System Testing

System testing tests the system. Once all the components are integrated, the application is tested rigorously to see that it meets the specified Quality Standards. This type of testing is performed by a specialized testing team.

System testing is important because of the following reasons:

- System testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.
- The application is tested thoroughly to verify that it meets the functional and technical specifications.
- The application is tested in an environment that is very close to the production environment where the application will be deployed.
- System testing enables us to test, verify, and validate both the business requirements as well as the application architecture.

Regression Testing

Whenever a change in a software application is made, it is quite possible that other areas within the application have been affected by this change. Regression testing is performed to verify that a fixed bug hasn't resulted in another functionality or business rule violation. The intent of regression testing is to ensure that a change, such as a bug fix should not result in another fault being uncovered in the application.

Regression testing is important because of the following reasons:

- Minimize the gaps in testing when an application with changes made has to be tested.
- Testing the new changes to verify that the changes made did not affect any other area of the application.
- Mitigates risks when regression testing is performed on the application.
- Test coverage is increased without compromising timelines.
- Increase speed to market the product.

Acceptance Testing

This is arguably the most important type of testing, as it is conducted by the Quality Assurance Team who will gauge whether the application meets the intended specifications and satisfies the client's requirement. The QA team will have a set of pre-written scenarios and test cases that will be used to test the application.

More ideas will be shared about the application and more tests can be performed on it to gauge its accuracy and the reasons why the project was initiated. Acceptance tests are not only intended to point out simple spelling mistakes, cosmetic errors, or interface gaps, but also to point out any bugs in the application that will result in system crashes or major errors in the application.

By performing acceptance tests on an application, the testing team will reduce how the application will perform in production. There are also legal and contractual requirements for acceptance of the system.

Alpha Testing

This test is the first stage of testing and will be performed amongst the teams (developer and QA teams). Unit testing, integration testing and system testing when combined is known as alpha testing. During this phase, the following aspects will be tested in the application:

- Spelling Mistakes
- Broken Links
- Cloudy Directions

The Application will be tested on machines with the lowest specification to test loading times and any latency problems.

Beta Testing

This test is performed after alpha testing has been successfully performed. In beta testing, a sample of the intended audience tests the application. Beta testing is also known as pre-release testing. Beta test versions of software are ideally distributed to a wide audience on the Web, partly to give the program a "real-world" test and partly to

provide a preview of the next release. In this phase, the audience will be testing the following:

- Users will install, run the application and send their feedback to the project team.
- Typographical errors, confusing application flow, and even crashes.
- Getting the feedback, the project team can fix the problems before releasing the software to the actual users.
- The more issues you fix that solve real user problems, the higher the quality of your application will be.
- Having a higher-quality application when you release it to the public will increase customer satisfaction.

6.4.2 Non-Functional Testing

This section is based upon testing an application from its non-functional attributes. Non-functional testing involves testing a software from the requirements which are non-functional in nature but important such as performance, security, user interface, etc.

Some of the important and commonly used non-functional testing types are discussed below.

Performance Testing

It is mostly used to identify any bottlenecks or performance issues rather than finding bugs in a software. There are different causes that contribute in lowering the performance of a software:

- Network delay
- Client-side processing
- Database transaction processing
- Load balancing between servers
- Data rendering
- Performance testing is considered as one of the important and mandatory testing type in terms of the following aspects:
- Speed (i.e. Response Time, data rendering and accessing)
- Capacity
- Stability
- Scalability

Performance testing can be either qualitative or quantitative and can be divided into different sub-types such as Load testing and Stress testing.

Load Testing

It is a process of testing the behaviour of a software by applying maximum load in terms of software accessing and manipulating large input data. It can be done at both normal and peak load conditions. This type of testing identifies the maximum capacity of software and its behaviour at peak time.

Most of the time, load testing is performed with the help of automated tools such as Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test, etc.

Virtual users (VUsers) are defined in the automated testing tool and the script is executed to verify the load testing for the software. The number of users can be increased or decreased concurrently or incrementally based upon the requirements.

Stress Testing

Stress testing includes testing the behaviour of a software under abnormal conditions. For example, it may include taking away some resources or applying a load beyond the actual load limit.

The aim of stress testing is to test the software by applying the load to the system and taking over the resources used by the software to identify the breaking point. This testing can be performed by testing different scenarios such as:

- Shutdown or restart of network ports randomly
- Turning the database on or off
- Running different processes that consume resources such as CPU, memory, server, etc.

Usability Testing

Usability testing is a black-box technique and is used to identify any error(s) and improvements in the software by observing the users through their usage and operation.

According to Nielsen, usability can be defined in terms of five factors, i.e. efficiency of use, learn-ability, memory-ability, errors/safety, and satisfaction. According to him, the usability of a product will be good and the system is usable if it possesses the above factors.

Nigel Bevan and Macleod considered that usability is the quality requirement that can be measured as the outcome of interactions with a computer system. This requirement can be fulfilled, and the end-user will be satisfied if the intended goals are achieved effectively with the use of proper resources.

Molich in 2000 stated that a user-friendly system should fulfil the following five goals, i.e., easy to Learn, easy to remember, efficient to use, satisfactory to use, and easy to understand.

In addition to the different definitions of usability, there are some standards and quality models and methods that define usability in the form of attributes and sub-attributes such as ISO-9126, ISO-9241-11, ISO-13407, and IEEE std.610.12, etc.

UI vs Usability Testing

UI testing involves testing the Graphical User Interface of the Software. UI testing ensures that the GUI functions according to the requirements and tested in terms of colour, alignment, size, and other properties.

On the other hand, usability testing ensures a good and user-friendly GUI that can be easily handled. UI testing can be considered as a sub-part of usability testing.

Security Testing

Security testing involves testing a software to identify any flaws and gaps from security and vulnerability point of view. Listed below are the main aspects that security testing should ensure:

- Confidentiality
- Integrity
- Authentication
- Availability
- Authorization
- Non-repudiation
- Software is secure against known and unknown vulnerabilities
- Software data is secure
- Software is according to all security regulations
- Input checking and validation
- SQL insertion attacks
- Injection flaws
- Session management issues
- Cross-site scripting attacks
- Buffer overflows vulnerabilities.

Portability Testing

Portability testing includes testing a software with the aim to ensure its reusability and that it can be moved from another software as well. Following are the strategies that can be used for portability testing:

- Transferring an installed software from one computer to another.
- Building executable (.exe) to run the software on different platforms.

Portability testing can be considered as one of the sub-parts of system testing, as this testing type includes overall testing of a software with respect to its usage over different environments. Computer hardware, operating systems, and browsers are the major focus of portability testing. Some of the pre-conditions for portability testing are as follows:

- Software should be designed and coded, keeping in mind the portability requirements.
- Unit testing has been performed on the associated components.
- Integration testing has been performed.
- Test environment has been established.

7. CONCLUSION

In the last decade, the importance of social media grew unbelievably. Here, we presented a proof-of-concept industrial application of social media analysis. We aimed to predict the number of feedbacks that blog documents receive. Our software prototype allowed to crawl data and perform experiments. The results show that state-of-the art regression models perform well, they outperform naive models substantially.

On the other hand, the results show that there is room for improvement, while developing new models for the blog feedback prediction problem seems to be a non-trivial task: with widely-used techniques, ensemble methods, we only achieved marginal improvement.

8. REFERENCES

- https://www.analyticsvidhya.com/
- https://www.wikipedia.org/
- https://machinelearningmastery.com/
- http://scikit-learn.org/stable/index.html
- https://www.python.org/
- https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d