**Project Title**

A Project Report Submitted in Partial Fulfilment of the Requirements

For the Degree of

**BACHELOR OF TECHNOLOGY**

In

**ELECTRONICS AND COMMUNICATION ENGINEERING**

By

***Tushar Mehrotra***

***XYZ***

***XYZ***

***XYZ***

Under the Supervision of

***PROF. ABC***

(Head of Department)

**Department of Information and Technology**

A picture containing food, drawing

Description automatically generated

Galgotias University

**Greater Noida, Uttar Pradesh, India**

**February, 2020**

**Candidate’s Declaration**

I hereby declare that the work, which is being presented in the Major Project entitled ‘**PROJECT NAME’** in partial fulfilment for the award of Degree of “Bachelor of Technology” and submitted to the department of Information and Technology Engineering, of Galgotias College of Engineering, Greater Noida is a record of my own investigations carried under the Guidance of **Dr ProfessorName** Department of Information and Technology Engineering, Galgotias College of Engineering. I have not submitted the matter presented in this report anywhere for the award of any other degree.

XYZ

XYZ

XYZ

XYZ

Information and Technology Engineering,

Galgotias College of Engineering

Counter Signed by

Name(s) of Supervisor(s)

.....................................(Prof. Professor Name)

.....................................(Er. NameOfProjectMentor)

**CERTIFICATE**

Certified that XYZ (Roll no. \*\*\*\*\*\*\*\*\*\*\*\*), XYZ (Roll no. \*\*\*\*\*\*\*\*\*\*\*\*), XYZ (Roll no. \*\*\*\*\*\*\*\*\*\*\*\*) and XYZ (Roll no. \*\*\*\*\*\*\*\*\*\*\*\*) have carried out the project work presented in this report entitled ‘**Project Name**’, for the award of Bachelor of Technology in Information and Technology from the Galgotias College of Engineering , Greater Noida under my supervision. The project report embodies result of the studies carried out by the students themselves. Its contents do not form the basis for award of any other degree to the students or to anybody else.

Signature:\_\_\_\_\_\_\_\_\_\_\_\_ Signature:\_\_\_\_\_\_\_\_\_\_\_\_\_

Co-guide:

*Guide:*

*Prof. XYZ*

*Head of Department*

Information & Technology

*Galgotias College of Eng.*

*Greater Noida, Uttar Pradesh*

Er. XYZ

Information & Technology faculty

Galgotias College of Eng.

Greater Noida, Uttar Pradesh

Place: Lucknow

Date: 25/02/2020

**ACKNOWLEDGEMENT**

I take this opportunity to express my gratitude to all those people who have been directly and indirectly with me during the competition of this project. I pay thank to our guide **Prof. XYZ** and our co-guide **Er. XYZ**, who has given guidance and a light to me during this major project. Their versatile knowledge about **PROJECT NAME** has eased me in the critical times during the span of this major project. I acknowledge here out debt to those who contributed significantly to one or more steps. I take full responsibility for any remaining sins of omission and commission.

XYZ

XYZ

XYZ

XYZ

B.Tech IV Year

(Information & Technology engineering.)

**Abstract**

Breast Cancer is the most often identified cancer among women and major reason for increasing mortality rate among women. As the diagnosis of this disease manually takes long hours and the lesser availability of systems, there is a need to develop the automatic diagnosis system for early detection of cancer. Data mining techniques contribute a lot in the development of such system. For the classification of benign and malignant tumor we have used classification techniques of machine learning in which the machine is learned from the past data and can predict the category of new input. This project is a relative study on the implementation of models using Logistic Regression, Support Vector Machine (SVM) , K Nearest Neighbor (KNN) and various other Ensemble and Boosting technique is done on the dataset taken from the UCI repository. With respect to the results of accuracy, precision, sensitivity, specificity and False Positive Rate the efficiency of each algorithm is measured and compared. These techniques are coded in python and executed in Jupyter Notebook, the Scientific Python Development Environment. Our experiments have shown that SVM is the best for predictive analysis with an accuracy of 92.7%. We infer from our study that SVM is the well suited algorithm for prediction.

INDEX

[LIST OF FIGURES 7](#_Toc36496251)

[1.1 Introduction 8](#_Toc36496252)

[2.1 Feature Selection 9](#_Toc36496254)

[2.2 Feature Projection 9](#_Toc36496255)

[2.3 Model Selection 9](#_Toc36496256)

[3.1. Linear Regression 10](#_Toc36496257)

[5.1 Bagging 16](#_Toc36496258)

[6.1 Boosting 18](#_Toc36496259)

[6.2 Adaptative boosting and Light Gradient boosting 19](#_Toc36496260)

[6.3 Gradient Boost 21](#_Toc36496261)

[6.4 XGBOOST 24](#_Toc36496262)

[7.1 Support Vector machine 26](#_Toc36496263)

[7.2 Cost Function and Gradient Updates 26](#_Toc36496264)

[CONCLUSION 29](#_Toc36496265)

[REFRENCE 30](#_Toc36496266)

### LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| Figure no. | Figure name | Page no. |
| Figure 3.2 | Logistic regression graph | 12 |
| Figure 3.3 | Odds Ratio | 13 |
| Figure 3.5 | Formula for Least square | 14 |
| Figure 3.6 | Log likelihood formula | 14 |
| Figure 4.2 | KNN-Scatter Plot | 16 |
| Figure4.3 | KNN-Methodology of working | 17 |
| Figure 6.1 | Graph of bias variance trade-off | 21 |
| Figure 6.2 | Infographic of Ensemble models | 23 |
| Figure 7.1 | Ensemble technique for bagging classifier | 25 |
| Figure 8.1 | Ensemble technique for Random Forest | 27 |
| Figure 9.1 | Infographic for Boosting Technique | 30 |
| Figure 9.2 | Adaptive boosting technique | 30 |
| Figure 9.3 | Infographic for Adaptive Boosting Technique | 32 |
| Figure 9.4 | Working of Gradient boosting | 35 |
| Figure 9.5 | XGBoost Formula | 37 |
| Figure 10.1 | Infographic figure of Support Vector Machine. | 39 |
| Figure 10.2 | Cost function for SVM | 21 |
| Figure 10.3 | Feature Importance as per our graph | 21 |

Chapter 1

# 1.1 Introduction

# Machine learning develops algorithms and builds models from data and uses them to predict on new data. The main difference with traditional algorithm is that a model is built from inputs data rather than just execute a series of instructions. Supervised learning uses data with result labeled, while unsupervised learning using unlabeled data. There are a few common machine learning algorithms, such as regression, classification, neural network and deep learning. Reinforcement learning and representation learning are heavily used for deep learning. How to use machine learning algorithms to predict house price? It is a challenge to get as closely as possible result based on the model built. For a specific house price, it is determined by location, size, house type, city, country, tax rules, economic cycle, population movement, interest rate, and many other factors which could affect demand and supply. For local house price prediction, there are many useful regression algorithms to use. For example, support vector machines (SVM), Lasso (least absolute shrinkage and selection operator) [2], Gradient boosting [3], Ridge, Random forest. We will investigate and explore them in Part III. After examining data, we find that the data quality is a key factor to predict the house prices. Data input feature density estimation is important for regression. Hence, normality test for each feature is to confirm whether it is well-modeled by a normal distribution and to explore possible transformation to a normal distribution. Homoscedasticity verifications are also considered, hence regression algorithms with parameter more than 10000 iterations are applied. But the result is determined by the homoscedasticity between training data and test data. Linearity of each feature is the statistic fundamental of regression algorithm; therefore, many transformations are applied to enhance the linearity of input features.

Chapter 2

We obtained the Housing Pricing dataset from UCI repository and used Jupyter Notebook as the platform for the purpose of coding. Our methodology involves use of classification techniques like Support Vector Machine (SVM), Linear Regression and various ensemble and boosting technique. We used proper imputation for dealing with the missing values.

## Feature Selection

Feature selection is finding the subset of original features by different approaches based on the information they provide, accuracy, prediction errors.

## Feature Projection

Feature projection is transformation of high-dimensional space data to a lower dimensional space (with few attributes). Both linear and nonlinear reduction techniques can be used in accordance with the type of relationships among the features in the dataset. The dataset used in this research is a multidimensional dataset with 32 attributes, which are related to cell parameters. Selection of features by the application of feature selection is a complex task. Moreover, it cannot give the most accurate features. Therefore, we have applied a feature projection technique, PCA to derive two principal components from the dataset.

## Model Selection

The most exciting phase in building any machine learning model is selection of algorithm. We can use more than one kind of data mining techniques to large datasets. But, at high level all those different algorithms can be classified in two groups: supervised learning and unsupervised learning. Supervised learning is the method in which the machine is trained on the data which the input and output are well labeled. The model can learn on the training data and can process the future data to predict outcome. They are grouped to Regression and Classification techniques. A regression problem is when the result is a real or continuous value, such as “salary” or “weight”. A classification problem is when the result is a category like filtering emails “spam” or “not spam”. Unsupervised Learning: Unsupervised learning is giving away information to the machine that is neither classified nor labeled and allowing the algorithm to analyze the given information without providing any directions. In unsupervised learning algorithm the machine is trained from the data which is not labeled or classified making the algorithm to work without proper instructions. In our dataset we have the outcome variable or Dependent variable i.e. Y having only two set of values, either M (Malign) or B(Benign). So, Classification algorithm of supervised learning is applied on it. We have chosen different types of classification algorithms in Machine Learning.

1. Linear Regression Methods like (LASSO and ElsaticNet)

2. Support Vector Machines

3. Ensemble models like Random Forest

4. Boosting Techniques like AdaBoost, Gradient Boosting etc.

5. Hybrid Model from Conventional ML models

Chapter 3

# 3.1. Linear Regression

It is a linear model that establishes the relationship between a dependent variable *y(Target),*and one or more independent variables denoted *X(Inputs)*.

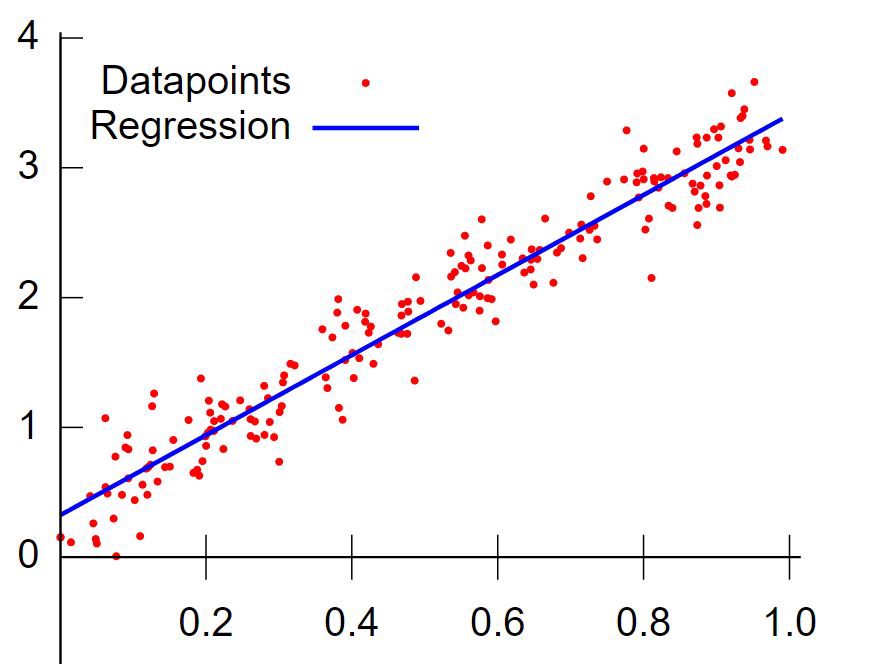


Fig 3.1 Linear Regression

**3.2 Preparing Data for Linear Regression**

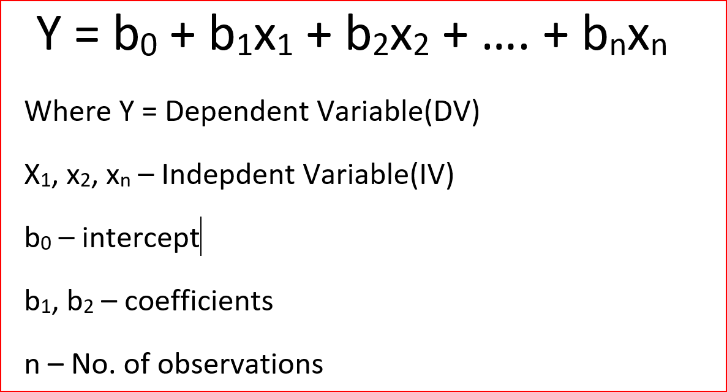
Linear regression is been studied at great length, and there is a lot of literature on how your data must be structured to make best use of the model.

As such, there is a lot of sophistication when talking about these requirements and expectations which can be intimidating. In practice, you can use these rules more as rules of thumb when using Ordinary Least Squares Regression, the most common implementation of linear regression.

Try different preparations of your data using these heuristics and see what works best for your problem.

* **Linear Assumption**. Linear regression assumes that the relationship between your input and output is linear. It does not support anything else. This may be obvious, but it is good to remember when you have a lot of attributes. You may need to transform data to make the relationship linear (e.g. log transform for an exponential relationship).
* **Remove Noise**. Linear regression assumes that your input and output variables are not noisy. Consider using data cleaning operations that let you better expose and clarify the signal in your data. This is most important for the output variable and you want to remove outliers in the output variable (y) if possible.
* **Remove Collinearity**. Linear regression will over-fit your data when you have highly correlated input variables. Consider calculating pairwise correlations for your input data and removing the most correlated.
* **Gaussian Distributions**. Linear regression will make more reliable predictions if your input and output variables have a Gaussian distribution. You may get some benefit using transforms (e.g. log or BoxCox) on your variables to make their distribution more Gaussian looking.
* **Rescale Inputs**: Linear regression will often make more reliable predictions if you rescale input variables using standardization or normalization.

**3.3 Equation for Linear Regression**



fif

fi

FiLi

3.2 Linear Regression Formula

**3.4 Gradient Descent**

Gradient descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. In machine learning, we use gradient descent to update the [parameters](https://ml-cheatsheet.readthedocs.io/en/latest/glossary.html#glossary-parameters) of our model. Parameters refer to coefficients in [Linear Regression](https://ml-cheatsheet.readthedocs.io/en/latest/linear_regression.html) and [weights](https://ml-cheatsheet.readthedocs.io/en/latest/nn_concepts.html#nn-weights) in neural networks.

**3.4.1 Introduction**

Consider the 3-dimensional graph below in the context of a cost function. Our goal is to move from the mountain in the top right corner (high cost) to the dark blue sea in the bottom left (low cost). The arrows represent the direction of steepest descent (negative gradient) from any given point–the direction that decreases the cost function as quickly as possible.

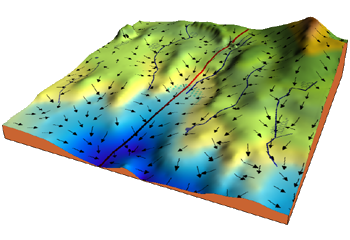


Figure 3.3 Gradient Descent Graphic elaboration

Starting at the top of the mountain, we take our first step downhill in the direction specified by the negative gradient. Next, we recalculate the negative gradient (passing in the coordinates of our new point) and take another step in the direction it specifies. We continue this process iteratively until we get to the bottom of our graph, or to a point where we can no longer move downhill–a local minimum.

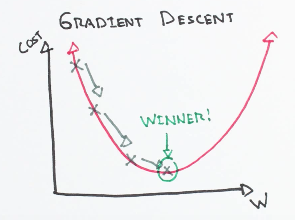


Figure 3.4 Gradient Descent Infographic

**3.4.2 Learning rate**

The size of these steps is called the learning rate. With a high learning rate, we can cover more ground each step, but we risk overshooting the lowest point since the slope of the hill is constantly changing. With a very low learning rate, we can confidently move in the direction of the negative gradient since we are recalculating it so frequently. A low learning rate is more precise, but calculating the gradient is time-consuming, so it will take us a very long time to get to the bottom.

**3.4.3 Cost function**

A [Loss Functions](https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html#cost-function) tells us “how good” our model is at making predictions for a given set of parameters. The cost function has its own curve and its own gradients. The slope of this curve tells us how to update our parameters to make the model more accurate.

**3.4.4 Step-by-step**

Now let’s run gradient descent using our new cost function. There are two parameters in our cost function we can control: mm (weight) and bb (bias). Since we need to consider the impact each one has on the final prediction, we need to use partial derivatives. We calculate the partial derivatives of the cost function with respect to each parameter and store the results in a gradient.

Given the cost function:

f(m,b)=1N∑i=1n(yi−(mxi+b))2f(m,b)=1N∑i=1n(yi−(mxi+b))2

The gradient can be calculated as:

f′(m,b)=⎡⎣dfdmdfdb⎤⎦=[1N∑−2xi(yi−(mxi+b))1N∑−2(yi−(mxi+b))]f′(m,b)=[dfdmdfdb]=[1N∑−2xi(yi−(mxi+b))1N∑−2(yi−(mxi+b))]

To solve for the gradient, we iterate through our data points using our new mm and bb values and compute the partial derivatives. This new gradient tells us the slope of our cost function at our current position (current parameter values) and the direction we should move to update our parameters. The size of our update is controlled by the learning rate.

Chapter 4

**4.1 Ridge and Lasso Regression: L1 and L2 Regularization**

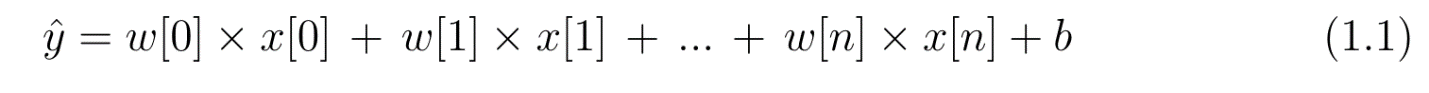
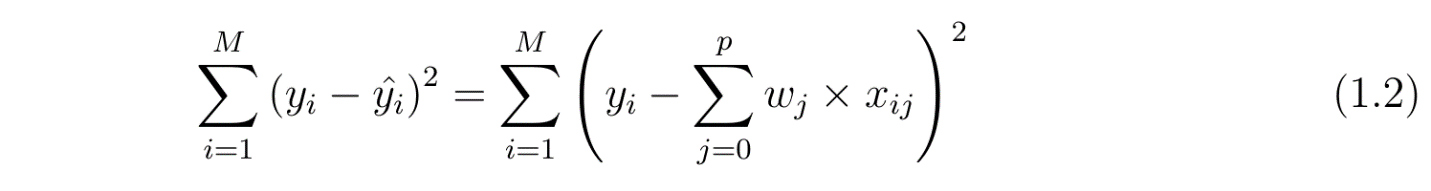


Figure 11.1 Linear model with n features for output prediction

In the equation above, shown the linear model based on the n number of features. Considering only a single feature that *w[0]* will be slope and *b*will represent intercept. Linear regression looks for optimizing *w* and *b* such that it minimizes the cost function. The cost function can be written as

Figure 11.2 Cost function for simple linear model

In the equation above I have assumed the dataset has M instances and p features. Once we use linear regression on a dataset divided in to training and test set, calculating the scores on training and test set can give us a rough idea about whether the model is suffering from over-fitting or under-fitting. The chosen linear model can be just right also, if you’re lucky enough! If we have very few features on a dataset and the score is poor for both training and test set, then it’s a problem of under-fitting. On the other hand, if we have large number of features and test score is relatively poor than the training score then it’s the problem of over-generalization or over-fitting. **Ridge and Lasso regression are some of the simple techniques to reduce model complexity and prevent over-fitting which may result from simple linear regression**.

**4.2 Ridge Regression**

In ridge regression, the cost function is altered by adding a penalty equivalent to square of the magnitude of the coefficients.

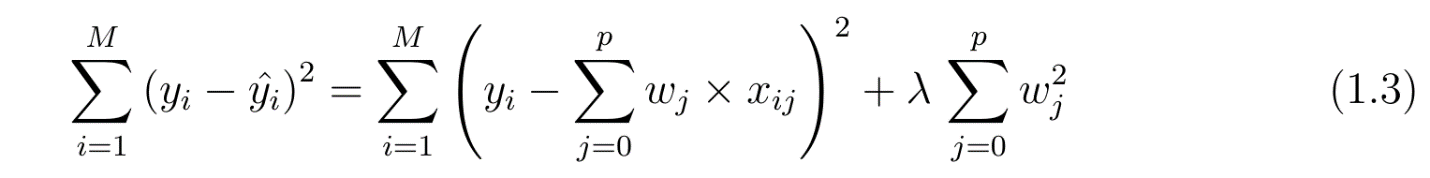


Figure 11.2 Cost function for ridge regression

This is equivalent to saying minimizing the cost function in equation 1.2 under the condition as below

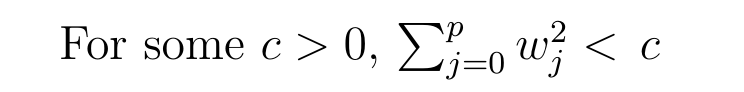


Figure 11.3 Constrain on Ridge regression coefficients

So, ridge regression puts constraint on the coefficients (w). The penalty term (lambda) regularizes the coefficients such that if the coefficients take large values the optimization function is penalized. So**, ridge regression shrinks the coefficients and it helps to reduce the model complexity and multi-collinearity.** Going back to eq. one can see that when λ → 0, the cost function becomes like the linear regression cost function (eq. 1.2). So, lower the constraint (low λ) on the features, the model will resemble linear regression model.

**4.3 Lasso Regression**

The cost function for Lasso (least absolute shrinkage and selection operator) regression can be written as

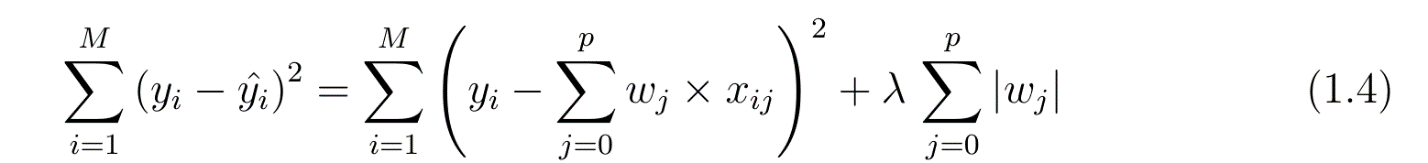


Figure 11.4 Cost function for Lasso regression

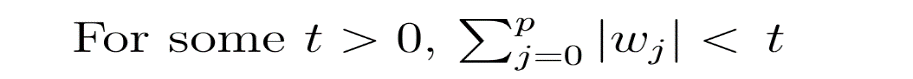


Figure 11.5 Lasso regression coefficients

Just like Ridge regression cost function, for lambda =0, the equation above reduces to equation 1.2. The only difference is instead of taking the square of the coefficients, magnitudes are considered*.* This type of regularization (L1) can lead to zero coefficients i.e. some of the features are completely neglected for the evaluation of output. **So, Lasso regression not only helps in reducing over-fitting but it can help us in feature selection.** Just like Ridge regression the regularization parameter (lambda) can be controlled. So, feature selection using Lasso regression can be depicted well by changing the regularization parameter.

lasso = make\_pipeline(RobustScaler(), Lasso(alpha =0.0005, random\_state=1))

score = rmsle\_cv(lasso)

print("\nLasso score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

**Lasso score: 0.1115 (0.0074)**

CHAPTER 5

# 5.1 Bagging

When training a model, no matter if we are dealing with a classification or a regression problem, we obtain a function that takes an input, returns an output and that is defined with respect to the training dataset. Due to the theoretical variance of the training dataset (we remind that a dataset is an observed sample coming from a true unknown underlying distribution), the fitted model is also subject to variability: **if another dataset had been observed, we would have obtained a different model**.

The idea of bagging is then simple: we want to fit several independent models and “average” their predictions in order to obtain a model with a lower variance. However, we can’t, in practice, fit fully independent models because it would require too much data. So, we rely on the good “approximate properties” of bootstrap samples (representativity and independence) to fit models that are almost independent.

First, we create multiple bootstrap samples so that each new bootstrap sample will act as another (almost) independent dataset drawn from true distribution. Then, we can **fit a weak learner for each of these samples and finally aggregate them such that we kind of “average” their outputs** and, so, obtain an ensemble model with less variance that its components. Roughly speaking, as the bootstrap samples are approximatively independent and identically distributed (i.i.d.), so are the learned base models. Then, “averaging” weak learners outputs do not change the expected answer but reduce its variance (just like averaging i.i.d. random variables preserve expected value but reduce variance).

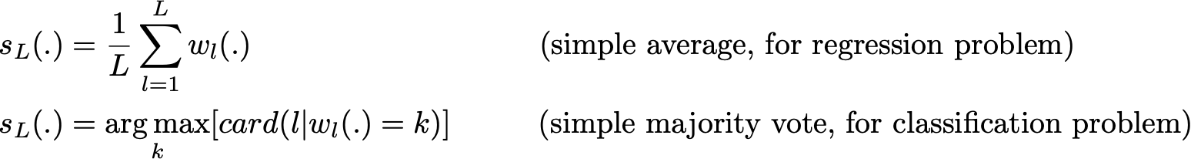
So, assuming that we have L bootstrap samples (approximations of L independent datasets) of size B denoted



we can fit L almost independent weak learners (one on each dataset)



and then aggregate them into some kind of averaging process in order to get an ensemble model with a lower variance. For example, we can define our strong model such that



There are several possible ways to aggregate the multiple models fitted in parallel. For a regression problem, the outputs of individual models can literally be averaged to obtain the output of the ensemble model. For classification problem the class outputted by each model can be seen as a vote and the class that receives the majority of the votes is returned by the ensemble model (this is called **hard-voting**). Still for a classification problem, we can also consider the probabilities of each classes returned by all the models, average these probabilities and keep the class with the highest average probability (this is called **soft-voting**). Averages or votes can either be simple or weighted if any relevant weights can be used.

Finally, we can mention that one of the big advantages of bagging is that **it can be parallelised**. As the different models are fitted independently from each others, intensive parallelisation techniques can be used if required.

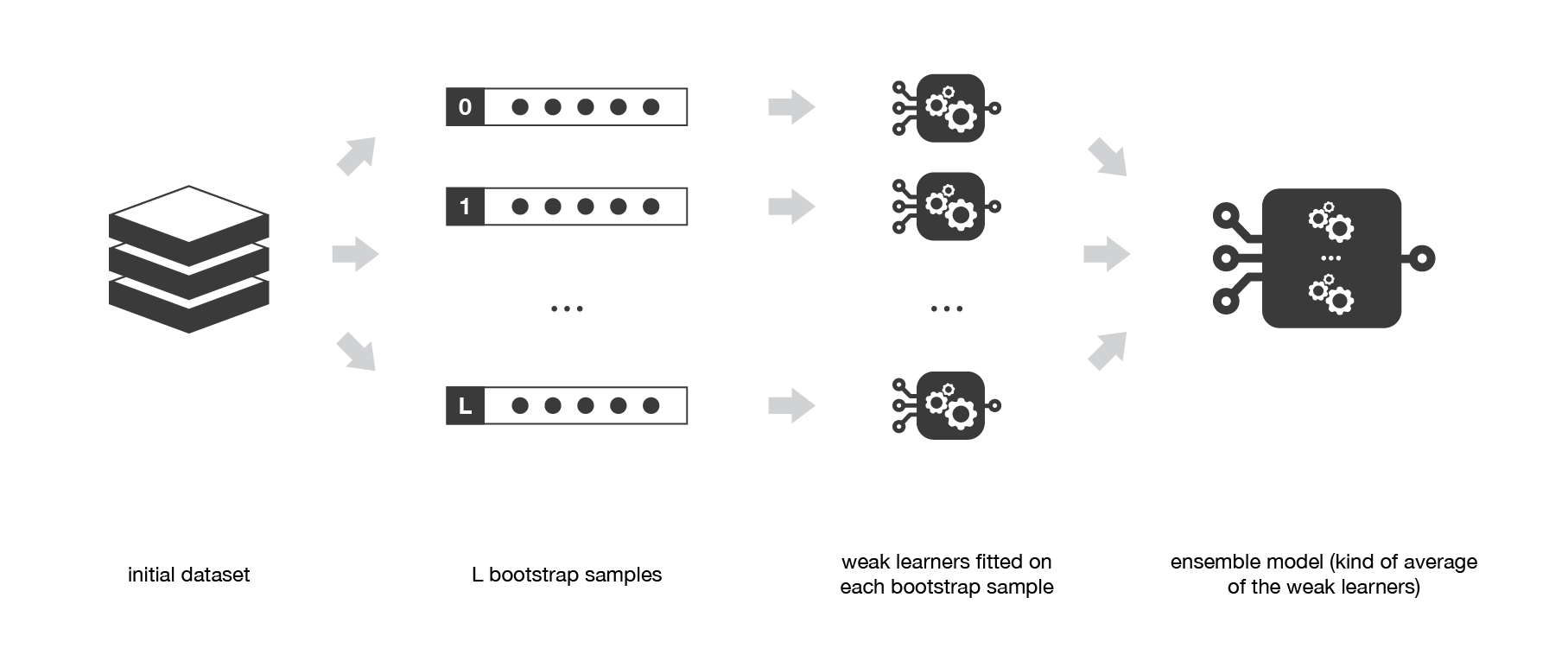


Figure7.1 Ensemble technique for bagging classifier

CHAPTER 6

## 6.1 Boosting

Boosting methods work in the same spirit as bagging methods: we build a family of models that are aggregated to obtain a strong learner that performs better. However, unlike bagging that mainly aims at reducing variance, boosting is a technique that consists in fitting sequentially multiple weak learners in a very adaptative way: each model in the sequence is fitted giving more importance to observations in the dataset that were badly handled by the previous models in the sequence. Intuitively, each new model **focus its efforts on the most difficult observations** to fit up to now, so that we obtain, at the end of the process, a strong learner with lower bias (even if we can notice that boosting can also have the effect of reducing variance). Boosting, like bagging, can be used for regression as well as for classification problems.

Being **mainly focused at reducing bias**, the base models that are often considered for boosting are models with low variance but high bias. For example, if we want to use trees as our base models, we will choose most of the time shallow decision trees with only a few depths. Another important reason that motivates the use of low variance but high bias models as weak learners for boosting is that these models are in general less computationally expensive to fit (few degrees of freedom when parametrised). Indeed, as computations to fit the different models **can’t be done in parallel** (unlike bagging), it could become too expensive to fit sequentially several complex models.

Once the weak learners have been chosen, we still need to define how they will be sequentially fitted (what information from previous models do we take into account when fitting current model?) and how they will be aggregated (how do we aggregate the current model to the previous ones?). We will discuss these questions in the two following subsections, describing more especially two important boosting algorithms: adaboost and gradient boosting.

In a nutshell, these two meta-algorithms differ on how they create and aggregate the weak learners during the sequential process. Adaptive boosting updates the weights attached to each of the training dataset observations whereas gradient boosting updates the value of these observations. This main difference comes from the way both methods try to solve the optimisation problem of finding the best model that can be written as a weighted sum of weak learners.

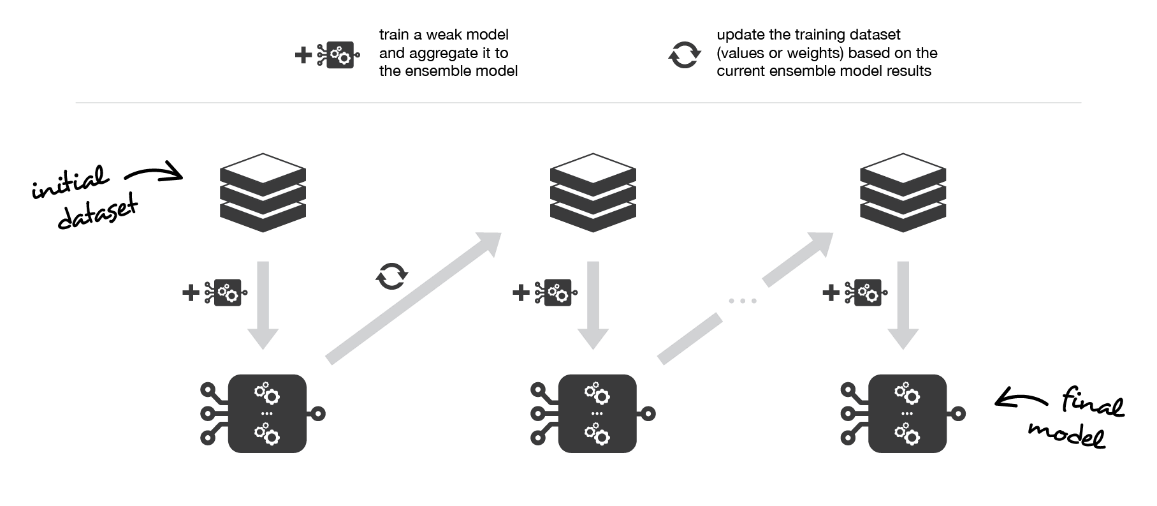


Figure 9.1 Infographic for Boosting Technique

## 6.2 Adaptative boosting and Light Gradient boosting

In adaptative boosting (often called “adaboost”), we try to define our ensemble model as a weighted sum of L weak learners

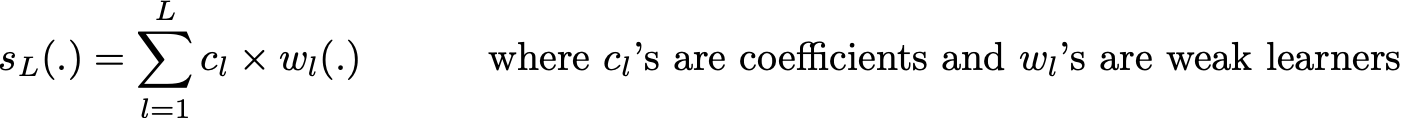
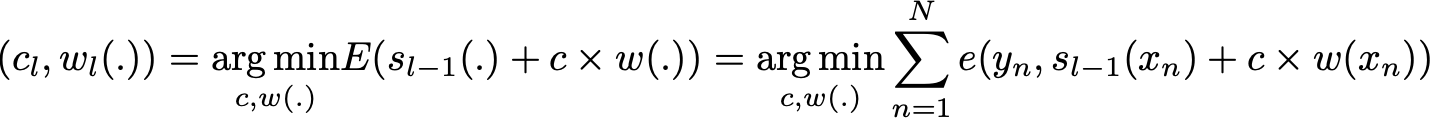


Figure 9.2 Adaptive boosting technique

Finding the best ensemble model with this form is a **difficult optimisation problem**. Then, instead of trying to solve it in one single shot (finding all the coefficients and weak learners that give the best overall additive model), we make use of an **iterative optimisation process** that is much more tractable, even if it can lead to a sub-optimal solution. More especially, we add the weak learners one by one, looking at each iteration for the best possible pair (coefficient, weak learner) to add to the current ensemble model. In other words, we define recurrently the (s\_l)’s such that



where c\_l and w\_l are chosen such that s\_l is the model that fit the best the training data and, so, that is the best possible improvement over s\_(l-1). We can then denote



where E(.) is the fitting error of the given model and e(.,.) is the loss/error function. Thus, instead of optimising “globally” over all the L models in the sum, we approximate the optimum by optimising “locally” building and adding the weak learners to the strong model one by one.

More especially, when considering a binary classification, we can show that the adaboost algorithm can be re-written into a process that proceeds as follow. First, it **updates the observations weights** in the dataset and train a new weak learner with a special focus given to the observations misclassified by the current ensemble model. Second, it **adds the weak learner to the weighted sum** according to an update coefficient that expresse the performances of this weak model: the better a weak learner performs, the more it contributes to the strong learner.

So, assume that we are facing a binary classification problem, with N observations in our dataset and we want to use adaboost algorithm with a given family of weak models. At the very beginning of the algorithm (first model of the sequence), all the observations have the same weights 1/N. Then, we repeat L times (for the L learners in the sequence) the following steps:

* fit the best possible weak model with the current observations weights
* compute the value of the update coefficient that is some kind of scalar evaluation metric of the weak learner that indicates how much this weak learner should be taken into account into the ensemble model
* update the strong learner by adding the new weak learner multiplied by its update coefficient
* compute new observations weights that expresses which observations we would like to focus on at the next iteration (weights of observations wrongly predicted by the aggregated model increase and weights of the correctly predicted observations decrease)

Repeating these steps, we have then built sequentially our L models and aggregate them into a simple linear combination weighted by coefficients expressing the performance of each learner. Notice that there exist variants of the initial adaboost algorithm such that LogitBoost (classification) or L2Boost (regression) that mainly differ by their choice of loss function.

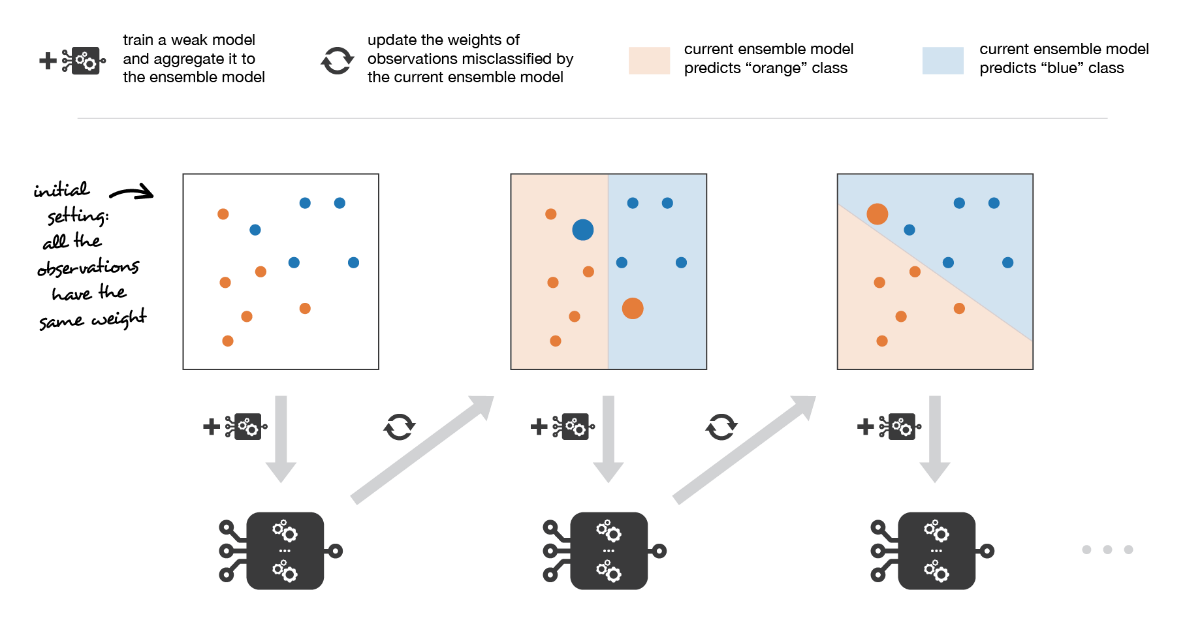


Figure 9.3 Infographic for Boosting Technique 2

Adaboost updates weights of the observations at each iteration. Weights of well classified observations decrease relatively to weights of misclassified observations. Models that perform better have higher weights in the final Following is the code snippet for the same: -

model\_lgb = lgb.LGBMRegressor(objective='regression',num\_leaves=5,

learning\_rate=0.05, n\_estimators=720,

max\_bin = 55, bagging\_fraction = 0.8,

bagging\_freq = 5, feature\_fraction = 0.2319,

feature\_fraction\_seed=9, bagging\_seed=9,

min\_data\_in\_leaf =6, min\_sum\_hessian\_in\_leaf = 11)

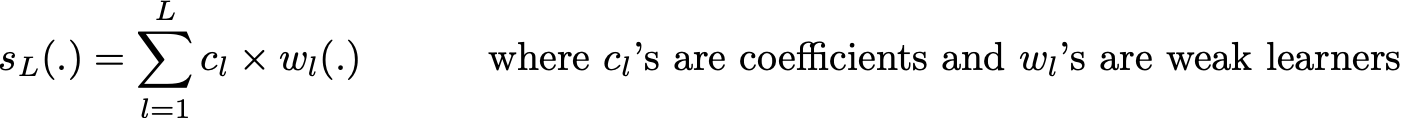
score = rmsle\_cv(model\_lgb)

print("LGBM score: {:.4f} ({:.4f})\n" .format(score.mean(), score.std()))

**LGBM score: 0.1438 (0.0060)**

## 6.3 Gradient Boost

In gradient boosting, the ensemble model we try to build is also a weighted sum of weak learners



Just as we mentioned for adaboost, finding the optimal model under this form is too difficult and an iterative approach is required. The main difference with adaptative boosting is in the definition of the sequential optimisation process. Indeed, gradient boosting **casts the problem into a gradient descent one**: at each iteration we fit a weak learner to the opposite of the gradient of the current fitting error with respect to the current ensemble model. Let’s try to clarify this last point. First, theoretical gradient descent process over the ensemble model can be written



where E(.) is the fitting error of the given model, c\_l is a coefficient corresponding to the step size and



is the opposite of the gradient of the fitting error with respect to the ensemble model at step l-1. This (pretty abstract) opposite of the gradient is a function that can, in practice, only be evaluated for observations in the training dataset (for which we know inputs and outputs): these evaluations are called **pseudo-residuals** attached to each observation. Moreover, even if we know for the observations the values of these pseudo-residuals, we don’t want to add to our ensemble model any kind of function: we only want to add a new instance of weak model. So, the natural thing to do is to **fit a weak learner to the pseudo-residuals** computed for each observation. Finally, the

coefficient c\_l is computed following a one-dimensional optimisation process (line-search to obtain the best step size c\_l).

So, assume that we want to use gradient boosting technique with a given family of weak models. At the very beginning of the algorithm (first model of the sequence), the pseudo-residuals are set equal to the observation values. Then, we repeat L times (for the L models of the sequence) the following steps:

* fit the best possible weak learner to pseudo-residuals (approximate the opposite of the gradient with respect to the current strong learner)
* compute the value of the optimal step size that defines by how much we update the ensemble model in the direction of the new weak learner
* update the ensemble model by adding the new weak learner multiplied by the step size (make a step of gradient descent)
* compute new pseudo-residuals that indicate, for each observation, in which direction we would like to update next the ensemble model predictions

Repeating these steps, we have then build sequentially our L models and aggregate them following a gradient descent approach. Notice that, while adaptative boosting tries to solve at each iteration exactly the “local” optimisation problem (find the best weak learner and its coefficient to add to the strong model), gradient boosting uses instead a gradient descent approach and can more easily be adapted to large number of loss functions. Thus, **gradient boosting can be considered as a generalization of adaboost to arbitrary differentiable loss functions**.

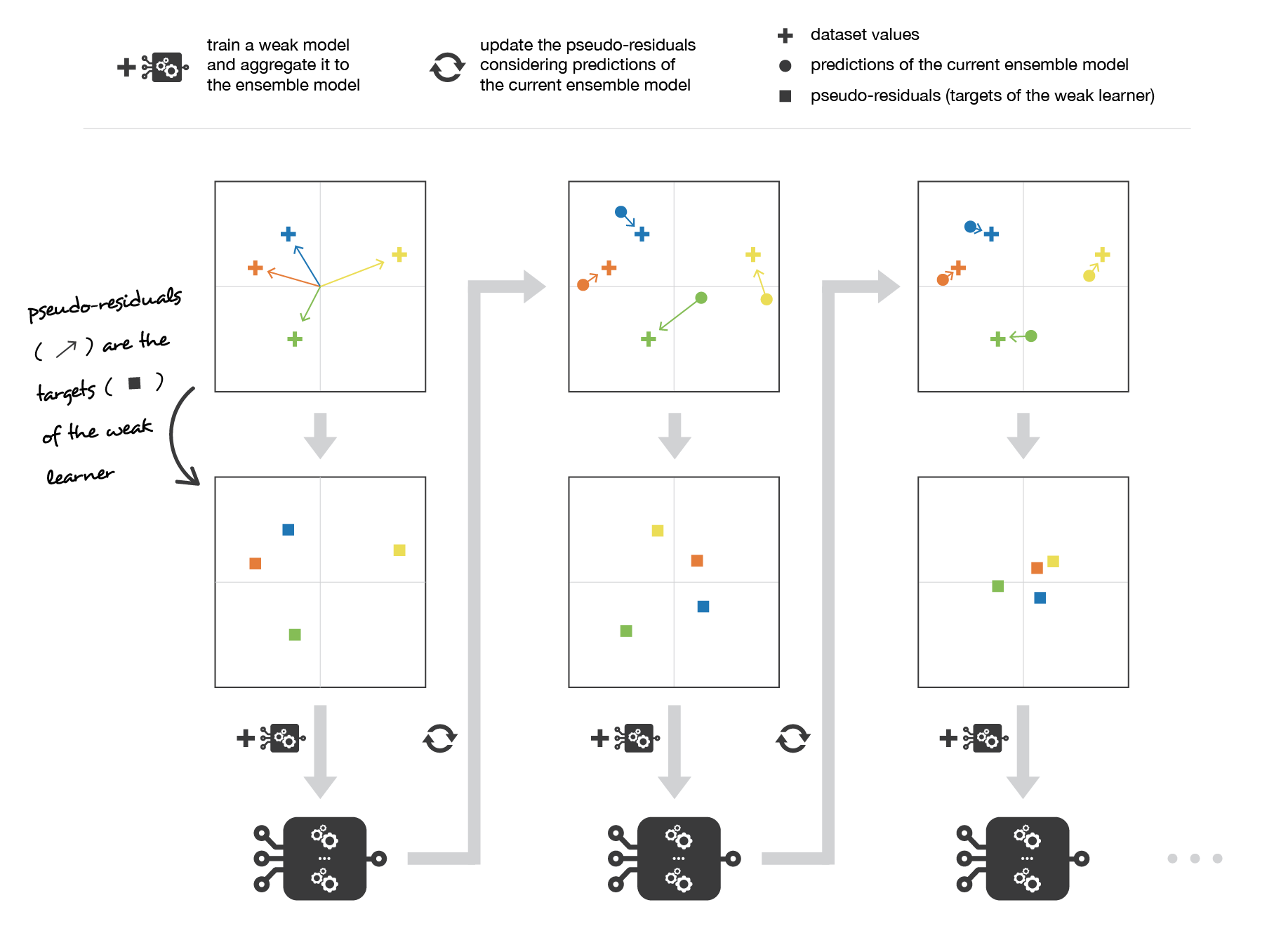


Figure 9.4 Working of Gradient boosting.

Gradient boosting updates values of the observations at each iteration. Weak learners are trained to fit the pseudo-residuals that indicate in which direction to correct the current ensemble model predictions to lower the error.

##### *Important Tuning Parameters for GBM:*

* learning\_rate - the effect of each tree on the outcome is shrunk by this factor.
* criterion - measure for quality of a split
* max\_depth - The maximum depth of the tree.
* max\_leaf\_nodes - Number of features to consider when looking for the best split
* min\_samples\_leaf - The minimum number of samples required to be a leaf node. This may have effect of smoothing the model.
* min\_sample\_split - The minimum number of samples required to split an internal node.
* n\_estimators - The number of trees in the forest
* max\_features - Number of features to consider when looking for the best split

Following is the code snippet for the same: -

GBoost = GradientBoostingRegressor(n\_estimators=3000, learning\_rate=0.05,

max\_depth=4, max\_features='sqrt',

min\_samples\_leaf=15, min\_samples\_split=10,

loss='huber', random\_state =5)

score = rmsle\_cv(GBoost)

print("Gradient Boosting score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

**Gradient Boosting score: 0.1177 (0.0080)**

## 6.4 XGBOOST

Extreme Gradient Boosting (XGBoost) is a more efficient version of gradient boosting framework containing both a linear model solver and tree learning algorithms.

The reason behind it’s efficiency is it’s capacity to do parallel computing on a single machine.

The problem with general boosting was

* Can’t extract the linear combination of features
* Small predictive power (high variance)

It used GBT approach that included

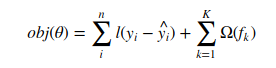
* control tree structure (maximum depth, minimum samples per leaf),
* control learning rate (shrinkage),
* reduce variance by introducing randomness (stochastic gradient boosting – using random subsamples of instances and features)

XGBoost improved it with some good features like-

* good bias-variance (simple-predictive) trade-off “out of the box”,
* great computation speed

XGBoost’s objective function is a sum of a specific loss function evaluated overall predictions and a sum of regularization term for all predictors (KK trees).

Mathematically,



FI

##### Figure 9.5 XGBoost Formula.

##### *Important Tuning Parameters for XGBOOST:*

* learning\_rate - Boosting Learning rate.
* criterion - measure for quality of a split
* max\_depth - The maximum depth of the tree.
* max\_leaf\_nodes - Number of features to consider when looking for the best split
* min\_samples\_leaf - The minimum number of samples required to be a leaf node. This may have effect of smoothing the model.
* min\_sample\_split - The minimum number of samples required to split an internal node.
* n\_estimators - The number of trees in the forest
* max\_features - Number of features to consider when looking for the best split
* gamma - Minimum loss reduction required to make a further partition on a leaf node of the tree.
* subsample - Denotes the fraction of observations to be randomly samples for each tree.
* colsample\_bytree - It is the subsample ratio of columns when constructing each tree. Subsampling occurs once for every tree constructed.
* reg\_alpa - L1 regularization term on weights. Increasing this value will make model more conservative.
* reg\_lambda - L2 regularization term on weights. Increasing this value will make model more conservative.
* scale\_pos\_weight - Control the balance of positive and negative weights, useful for unbalanced classes. A typical value to consider: sum(negative instances) / sum(positive instances).

Following is the code snippet for the same: -

model\_xgb = xgb.XGBRegressor(colsample\_bytree=0.4603, gamma=0.0468,

learning\_rate=0.05, max\_depth=3,

min\_child\_weight=1.7817, n\_estimators=2200,

reg\_alpha=0.4640, reg\_lambda=0.8571,

subsample=0.5213, silent=1,

random\_state =7, nthread = -1)

score = rmsle\_cv(model\_xgb)

print("Xgboost score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))

**Xgboost score: 0.1165 (0.0072)**

CHAPTER 7

## 7.1 Support Vector machine

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points.

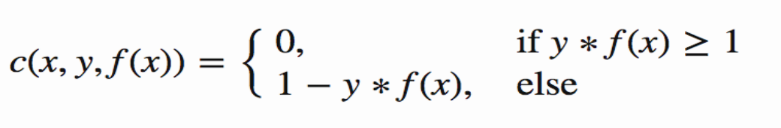


Figure 10.1 Infographic figure of Support Vector Machine.

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

## 7.2 Cost Function and Gradient Updates

In the SVM algorithm, we are looking to maximize the margin between the data points and the hyperplane. The loss function that helps maximize the margin is hinge loss.



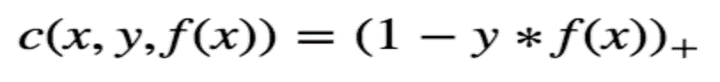
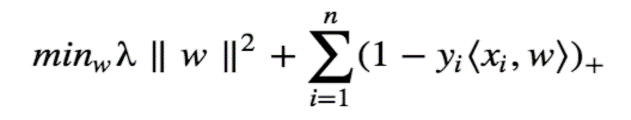


Figure 10.2 Cost function for SVM

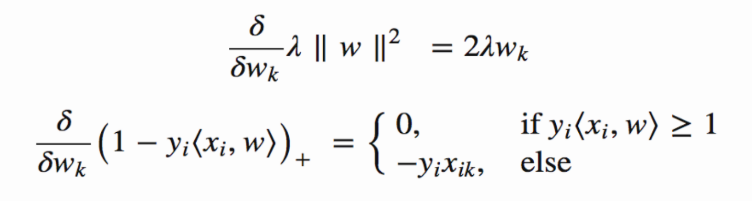
Hinge loss function (function on left can be represented as a function on the right)

The cost is 0 if the predicted value and the actual value are of the same sign. If they are not, we then calculate the loss value. We also add a regularization parameter the cost function. The objective of the regularization parameter is to balance the margin maximization and loss. After adding the regularization parameter, the cost functions looks as below.

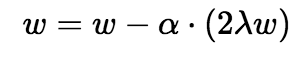


Loss function for SVM

Now that we have the loss function, we take partial derivatives with respect to the weights to find the gradients. Using the gradients, we can update our weights.

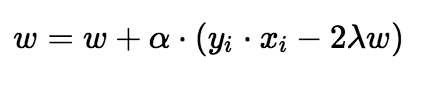


When there is no misclassification, i.e. our model correctly predicts the class of our data point, we only must update the gradient from the regularization parameter.



Gradient Update — No misclassification

When there is a misclassification, i.e our model make a mistake on the prediction of the class of our data point, we include the loss along with the regularization parameter to perform gradient update.



##### *Important Tuning Parameters for Support Vector Classifier:*

* C - C is the penalty parameter of the error term. It controls the tradeoff between smooth decision boundary and classifying the training points correctly
* class\_weight - Weights associated with classes in the form class\_label: Weight. If not given, all classes are supposed to have weight one.
* degree - degree is a parameter used when kernel is set to poly. It’s basically the degree of the polynomial used to find the hyperplane to split the data.
* gamma - kernel coefficient for rbf, ploy and sigmoid to handle non-linear classification
* kernel - This selects the type of hyperplane used to separate the data.
* tol - tolerance for stopping criterion.

Following is the code snippet for the same: -

model\_svr = SVR(C=0.8, epsilon=0.2, kernel='poly')

score = rmsle\_cv(model\_svr)

print("LGBM score: {:.4f} ({:.4f})\n" .format(score.mean(), score.std()))

**Result: - LGBM score: 0.1438 (0.0060)**

CHAPTER 8

**8.1 Hybrid Model**

In this approach, we add a meta-model on averaged base models and use the out-of-folds predictions of these base models to train our meta-model.

The procedure, for the training part, may be described as follows:

1. Split the total training set into two disjoint sets (here **train** and .**holdout** )
2. Train several base models on the first part (**train**)
3. Test these base models on the second part (**holdout**)
4. Use the predictions from 3) (called out-of-folds predictions) as the inputs, and the correct responses (target variable) as the outputs to train a higher level learner called **meta-model**.

The first three steps are done iteratively . If we take for example a 5-fold hybriding , we first split the training data into 5 folds. Then we will do 5 iterations. In each iteration, we train every base model on 4 folds and predict on the remaining fold (holdout fold).

So, we will be sure, after 5 iterations , that the entire data is used to get out-of-folds predictions that we will then use as new feature to train our meta-model in the step 4.

For the prediction part , We average the predictions of all base models on the test data and used them as **meta-features** on which, the final prediction is done with the meta-model.

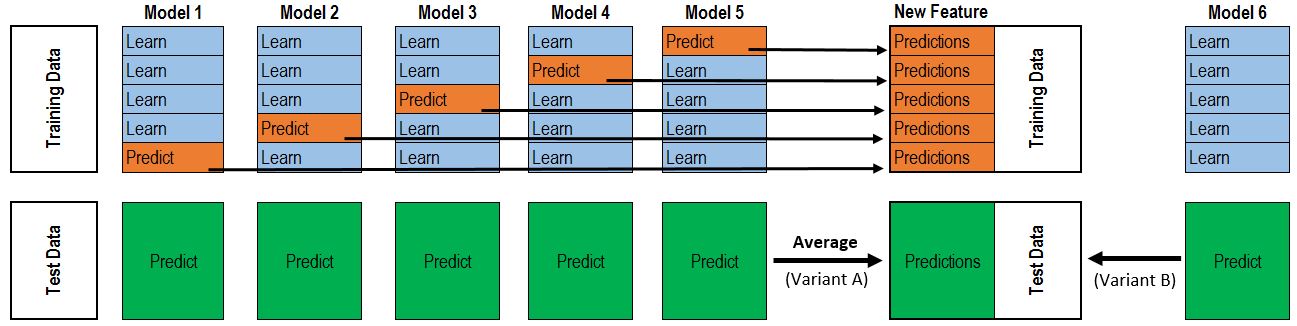


Fig 8.1 Informatic diagram for hybrid ensemble models.

**8.2 Hybriding Averaged models Score**

To make the two approaches comparable (by using the same number of models), we just average **Enet KRR and Gboost**, then we add **lasso as meta-model**.

The code snippet for same is: -

hybrid\_averaged\_models = HybridingAveragedModels(base\_models = (ENet, GBoost, KRR), meta\_model = lasso)

score = rmsle\_cv(hybrid\_averaged\_models)

print("Hybriding Averaged models score: {:.4f} ({:.4f})".format(score.mean(), score.std()))

**Hybriding Averaged models score: 0.1085 (0.0074)**

CHAPTER 9

# CONCLUSION

In this project, we propose a novel hybrid multi-regression model to predict the housing price. The accuracy of our model lies in predicting housing price based on specific house price it is determined by location, size, house type, city, country, tax rules, economic cycle, population movement, interest rate, and many other factors. Using statistical error analyses, we identified that a hybrid model outperformed other conventional machine learning algorithm. We obtained the minimum r.m.s.e.(or highest R squared value) with the hybrid model of 0.074 (scaled value) and has the most robustness to the input data

CHAPTER 10

# REFRENCE

1. Eli Beracha, Ben T Gilbert, Tyler Kjorstad, Kiplan womack, "On the Relation between Local Amenities and House Price Dynamics", Journal of Real estate Economics, Aug. 2016.
2. Stephen Law, "Defining Street-based Local Area and measuring its effect on house price using a hedonic price approach: The case study of Metropolitan London", Cities, vol. 60, Part A, pp. 166–179, Feb. 2017.
3. Alexander N. Bogin , William M. Doerner, "Property Renovations and Their Impact on House Price Index Construction".
4. David C. Ling, Joseph T.L. Ooi and Thao T.T. Le, "Explaining house price dynamics: Isolating the role of nonfundamentals", Journal of Money, Credit and Banking, vol. 47, Issue S1, pp. 87-125, March/April 2015.
5. Daniel P. McMillen, Christian L. Redfearn, "Estimation And Hypothesis Testing For Nonparametric Hedonic House Price Functions", Journal of Regional Science, vol. 50, Issue 3, pp. 712–733, Aug. 2010.
6. Binbin Lu, Martin Charlton, Paul Harris & A. Stewart Fotheringham, "Geographically weighted regression with a non-Euclidean distance metric: a case study using hedonic house price data", International Journal of Geographical Information Science, pp. 660-681, Jan 2014.
7. Marco Helbich, Wolfgang Brunauer, Eric Vaz, Peter Nijkamp, "Spatial Heterogeneity in Hedonic House Price Models: The Case of Austria", Urban Studies, vol. 51, Issue 2, Feb. 2014
8. Sean Holly, M. Hashem Pesarana, Takashi Yamagata, "A spatio-temporal model of house prices in the USA", Journal of Econometrics, vol. 158, Issue 1, pp. 160–173, Sep. 2010. [12] Joep Steegmans, Wolter Hassink, "Financial position and house price determination: An empirical study of income and wealth effects", Journal of Housing Economics, vol. 36, pp. 8-24, June 2017.