POLITECNICO DI TORINO

Master's Degree in ICT for Smart Societies

Master's Degree Thesis

Machine Learning Methodologies for Airfare Prediction

Supervisors

Prof. Alessandro ALIBERTI

Prof. Edoardo PATTI

Candidate

Xin YAO

October 2022

Abstract

With the booming tourism industry, more and more people are choosing airplanes as a means of transportation for long-distance travel. Accurate low-price forecasting of air tickets helps the aviation industry to match demand and supply flexibly and make full use of aviation resources. Airline companies use dynamic pricing strategies to determine the price of airline tickets to maximize profits when selling airline tickets. Passengers who choose airplanes as a means of transportation want to purchase tickets at the lowest selling price for the flight of their choice. However, airline tickets are a special commodity that is time-sensitive and scarce, and the price of airline tickets is affected by various factors, such as the departure time of the plane, the number of hours of advance purchase, and the airline flight, so it is difficult for consumers to know the best time to buy a ticket. Deep learning algorithms have made great achievements in various fields in recent years, however, most prior work on airfare prediction problems is based on traditional machine learning methods, thus the performance of deep learning on this problem remains unclear. In this thesis, we did a systematic comparison of various traditional machine learning methods (i.e., Ridge Regression, Lasso Regression, K-Nearest Neighbor, Decision Tree, XGBoost, Random Forest) and deep learning methods (e.g., Fully Connected Networks, Convolutional Neural Networks, Transformer) on the problem of airfare prediction. Inspired by the observation that ensemble models like XGBoost and Random Forest achieve better performance than other traditional machine learning methods, we proposed a Bayesian neural network for airfare prediction, which is the first method that utilizes Bayesian Inference for the airfare prediction task. We evaluate the performance of different methods on an open dataset of 10,683 domestic routes in India from March 2019 to June 2019. The experimental results show that deep learning-based methods achieve better results than traditional methods in RMSE and R^2, while Bayesian neural networks can achieve better performance than other machine learning methods.

Keywords: Airfare prediction, Regression, Machine Learning, Deep Learning, Bayesian Neural Network

Acknowledgements

I would like to convey my deep appreciation to my thesis advisors Prof. Alessandro Aliberti and Prof. Edoardo Patti for their constant guidance and encouragement throughout the thesis, and for always steering me in the right direction. I would like to express my sincerest appreciation to my friends and family for their endless support and encouragement in my life. And to Jiaxi, for always being by my side.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

As the world economy has expanded and the aviation sector has rapidly developed, more and more regular people are choosing to travel by plane. Technical research in the area of civil aviation has been encouraged by the industry's quick expansion and the rise in passenger traffic.

The airline sector is one of those that use sophisticated pricing techniques, and even adjacent seats in the same service class may be marketed at wildly different costs for tickets on the same aircraft. This is so that airlines may optimize their earnings using a variety of intricate pricing schemes in the yield management system. In order to meet demand, airlines are also working to set up fair flight schedules. Airlines have focused their efforts recently on research that includes demand forecasting and pricing discrimination. As a result, even though the growth of the Internet today has made it possible for consumers to access more useful information, airlines can still create information asymmetry by keeping useful information like the number of available seats a trade secret and adjusting ticket prices on the fly to maximize profits. Airline search volume is crucial information for airlines as well as significant information for pricing and scheduling flights. The prediction service may incur significant computational costs and overhead if it is used too frequently. Therefore, creating an appropriate prediction service invocation approach is a critical issue that must be resolved. However, there are few research on this topic, and applications involving power systems have had comparable issues. However, these techniques cannot be effectively implemented in this issue situation due to the specificity of data in the sector of airline tickets.

Even while airlines have developed their theoretical understanding of air ticket pricing and revenue management, there is still a dearth of research on customer purchase behavior. There are principally two causes for this: The first is that most airlines do not publicly publish their pricing methods since airline ticket pricing techniques are very sensitive business information. Second, many models can only be trained using partial pricing data downloaded from the Internet via crawler programs since there aren't enough publicly accessible datasets for academics to test their hypotheses on. In an effort to provide customers with better buying tactics, several recent research has put forth some suggestions based on scant data. From the standpoint of forecasting ticket costs, this research manually constructed regression models. They also employed techniques that specifically target the day the cheap price first shows. The little amount of training data, however, prevent these tactics from generalizing well, and the suggested techniques cannot be used in practical situations. Predicting the price of an airline ticket is a common time series prediction issue, but because the data is unique, the model is prone to error. Additionally, because there are several factors that impact airline ticket costs, it is essential to create a flawless model.

1.2 Contributions

The main contributions of this thesis are:
(1) We did a systematic comparison of traditional machine learning methods and deep learning methods on the problem of airfare prediction.
(2) Inspired by the observation that ensemble models like XGBoost and Random Forest achieve better performance than other traditional machine learning methods, We propose a novel Bayesian airfare prediction network, which learns the data distribution from the training data and utilize the ensemble idea to boost the performance.
(3) We demonstrate the superior performance of the proposed Bayesian airfare prediction network on a public dataset.

1.3 Thesis Organization

The thesis is organized as follows:

- Chapter 1 introduces the background, motivation, goals, and contributions of the thesis.
- Chapter 2 will summarize the previous work on the topic of airfare prediction.
- Chapter 3 will introduce the traditional machine learning methods and recent popular deep learning methods we adopt in our thesis.

- Chapter 4 will show the dataset, experimental details, and results.
- Chapter 5 will give concluding remarks, summarize the major contributions of our work, and present the future outlooks.

Chapter 2

Related Work

In recent years, different forecasting methods have been proposed for air ticket price forecasting. In this part, some related work in machine learning will be introduced, as artificial intelligence has flourished, which inspired the idea of this article. In this chapter, we first present the various previous types of studies that are related to our study topic. Secondly, we will present the regression methods which have been used in previous studies and give a brief review of the disadvantages of current models in this problem.

By utilizing openly accessible datasets and a cutting-edge machine learning framework, Tianyi Wang and his colleagues tackled the issue of market segment-level flight price prediction [1]. The DB1B and T-100 databases, which are gathered and managed by the Office of Airline Information under the United States Bureau of Transportation Statistics, are two particular public datasets from which their suggested framework gathers information (BTS). The DB1B dataset has been used in several studies that analyze the structure and dynamics of O-D for the core of the air travel industry [2], estimate demand [3], and evaluate the factors that affect aircraft features and flight frequency. The T-100dataset covers big certified carriers with Certificates of Public Convenience and Necessity and contains air passenger volumes for domestic and international markets in the United States. Their suggested architecture aims to provide a thorough profile of each market and use machine learning techniques to forecast the typical airfare at the level of market segments.

A regression model put up by Groves and Gini uses the history diagram to forecast the ideal time to book airline tickets [4]. From February 22, 2011, to June 23, 2011, they gathered their data, which included over 140.000 records in total [5]. Their model consists of two phases. To start, they predicted the day price using a regression model. The second step was to create a reliable rule based on the dependable threshold. If the price is less than the value, which is the predicted price less the threshold, passengers should purchase the ticket. Travelers should wait if not. Their findings demonstrated that their methodology can successfully reduce the average cost when the purchase date is more than two months out from the departure date. Additionally, their technology allows users to input preferences like the number of pauses they are willing to take. The same year, Wohlfarth et al. presented the MPP (Marked Point Process) preprocess technique [6]. It is concentrating on predicting when the price will decrease or decline. They used a clustering technique and a tree model to generate predictions after reducing the size of the feature collection. Their information was gathered from nine providers of airline tickets, with a focus on six roundtrips. They selected 3, 7, or 14 days as the duration of stay to cover the most typical stay.

Tanisha Patel used Python libraries like Pandas, NumPy, Matplotlib, seaborn, and Sklearn to implement the machine learning life cycle and construct a simple web application that predicts travel pricing by utilizing machine learning algorithms on historical flight data [7]. The first stage in gathering historical flight data for the model to anticipate pricing is data selection. More than 10,000 records of information on flights and their costs are included in their dataset. Source, destination, departure date, departure time, number of stops, arrival time, pricing, and a few more parameters are among the features of the dataset. They cleaned the dataset during the exploratory data analysis stage by deleting duplicate and null values. The accuracy of the model would suffer if these values weren't eliminated. They learned other details, such as how the data was distributed. The following stage is data pre-processing, during which we discovered that the majority of the data was already in string format. Each feature's data is retrieved, including the day and month from the journey's date in integer format and the hours and minutes from the departure time. Because they were categorical features, source and destination had to be turned into values. For this one, categorical variables are transformed into model-identifiable values using hot-encoding and label-encoding approaches. In the feature selection process, significant features that are more closely connected to the pricing are chosen. Before making our model suitable for prediction, there are several characteristics that need to be deleted since they may impair the model's accuracy, such as excess information and routes. The following phase entails using a machine algorithm to create a model after choosing the attributes that are most connected to pricing. As their dataset consists of labeled data, they will employ supervised machine learning methods. Additionally, because their dataset comprises continuous values in the features, they will employ regression algorithms in supervised learning. To explain the link between dependent and independent variables, regression models are utilized.

Papadakis [8] used a wider range of features, some of which I believe may not be available to consumers, like the number of unsold seats, and features based on historical data, like the most recent price for the same ticket, to transform the problem into a supervised classification problem with the aid of the Ripple Down Rule Learner algorithm. Lu [9] used a neural network as one of their models in a manner similar to this and used historical data elements including the highest and lowest price to date. A well-known and widely cited publication in this field, To Buy or Not to Buy: Mining Airfare Data [10] to Minimize Ticket Purchase Pricing, investigates how airline costs change by examining pricing changes on a particular aircraft number and route. The authors quantified their results by simulating the predicted total savings from applying each strategy [11].

In Selim BUYRUKOLU's study [12], 1814 one-way flights from Greece to Germany made up the dataset. Based on Mean Squared Error-MSE values (Ridge: 160103, Lasso: 159280, Elastic Net: 174203), and Mean Absolute Error-MAE values, the created Ridge, Lasso, and Elastic Net techniques were successful in producing compelling findings for flight pricing analysis (Ridge:147.74, Lasso:146.43, Elastic Net:346.86). How closely a regression line resembles a group of points is explained by MSE and MAE. They measure the "errors" by measuring the distances between the points and the regression line. Both MSE and MAE calculate the squares and absolutes of the errors. The forecast is more accurate the lower they are. Lasso regression has the lowest MSE and MAE values in our scenario, making it superior to the ridge and elastic net. Based on the lasso regression, two unrelated features—overnight and arrival time—are removed. Future predictions of flight costs can be made with greater datasets.

In [13], they suggested utilizing LR, Naive Bayes, SoftMax regression, and SVMs to create a prediction model and divide the ticket price into five categories (i.e., 60% to 80%, 80% to 100%, 100% to 120%, and so on) in order to compare the relative values with the average ticket price. The models were developed using more than 9,000 data points, comprising six characteristics (e.g., the start of the departure week, the date of the price quote, the number of stops on the itinerary, etc.). The LR model was used by the authors to produce the best training error rate, which was close to 22.9%. Their SVM regression model was unable to deliver a successful outcome. Instead, the prices were divided into two categories: "greater" and "lower" than the average, using an SVM classification model [1].

In [14], four LR models were tested in order to determine the optimal fit model, with the goal of giving the passenger fair advice on whether to purchase the ticket or wait longer for a lower price. The authors proposed to anticipate the cheapest ticket prices, which are referred to as the "true bargains," using linear quantile mixed models. However, this research is only applicable to one ticket class, and economy, and only on direct flights with a single stopover from John F. Kennedy Airport to the San Francisco Airport. In order to aid the consumers' decision-making process, Wohlfarth et al. [15] incorporated clustering as a preliminary stage with many cutting-edge supervised learning algorithms (classification tree (CART) and RF). To group flights in the price series that exhibit similar behavior, their architecture employs the K-Means method. They then employ CART to understand insightful rules and RF to convey knowledge about the significance of each aspect. The authors also noted that one aspect, the number of seats remaining, is a crucial aspect of ticket price prediction. Numerous more factors influence the competitive market in addition to flight-specific aspects. For instance, a travel agency's accumulated expenditures, which are brought on by over-purchasing or missed orders, can be decreased by accurately anticipating the market demand.

In [16], it describes how the author used Artificial Neural Networks (ANN) and Genetic Algorithms (GA) to forecast the income from sales of airline tickets. The input characteristics included the price of crude oil on the world market, the weighted index of Taiwan's stock market, the monthly unemployment rate in Taiwan, and more. In order to boost the performance of the ANNs, the GA specifically chooses the best input characteristics. With a mean absolute percentage error of 9.11%, the model performed well. More sophisticated machine learning models have been taken into consideration to enhance the forecast of flight prices as of 2017 [17][18]. Eight machine learning models, including ANNs, RF, SVM, and LR, were used by Tziridis et al. [17] to forecast ticket prices and evaluate their effectiveness. The most accurate regression model has an 88% accuracy rate. The Bagging Regression Tree, which is reliable and unaffected by utilizing various input feature sets, is judged to be the best model in their comparison.

Deep Regressor Stacking was suggested in [17] as a way to make forecasts that were more accurate. The suggested solution is a brand-new multi-target strategy that uses RF and SVM as regressors and is easily adaptable to other problem domains with a similar set of issues. Since airline ticket data is rarely categorized and prepared for direct analysis, gathering and processing such data is always labor-intensive. For the majority of analyses reported in the literature, researchers either seek private data from collaborative groups or web crawl the data to test the effectiveness of their models on various datasets. It is so challenging to reproduce the study and make performance comparisons among the models. The T100 and DB1A/1B databases include publicly accessible fare data for American airlines. However, these datasets are rarely employed independently to produce scientific study outputs due to the weak correlation between the pricing and individual flight details [19]. However, it is more probable that academics who are interested in, say, examining price dispersion may think about looking into the data from those databases [20].

The Official Airline Guide (OAG) and DB1B data are utilized to estimate the airline pricing in Rama-dissertation Murthy's [21]. Additionally, the author uses Sabre AirPrice data that SABRE gave, however, they only provide information on their online users. The conclusions from the data might be skewed because this internet user data does not reflect the whole consumer market. Our suggested system, in contrast to prior and current work, is able to solve the price prediction problem by utilizing just public data sources with a minimum of characteristics. Additionally, as it is not constrained by any particular market segment as is the case with the prior work, the suggested framework may be used to estimate the cost of air travel in any market.

Chapter 3

Methods

3.1 Machine Learning for Regression

3.1.1 Ridge Regression

We just assume that A and B have been centered, therefore the regression does not require a constant term:
(1) A is an n by p matrix with centered columns,
(2) B is a centered n-vector.

Before computing the inverse of the matrix $A'A$, Hoerl and Kennard (1970) suggested that potential instability in the LS estimator could be reduced by adding a small constant value λ to the diagonal entries.

$$\hat{\beta} = (A'A)^{-1} X'B \tag{3.1}$$

The result is the ridge regression estimator

$$\hat{\beta}_{ridge} = (A'A + \lambda I_p)^{-1} A'B \tag{3.2}$$

The parameters (β's) are subject to a specific type of constraint in ridge regression: $\hat{\beta}_{\text{ridge}}$ is chosen to minimize the penalized sum of squares:

$$\sum_{i=1}^{n} \left(b_i - \sum_{j=1}^{p} a_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{3.3}$$

which is equivalent to minimization of $\sum_{i=1}^{n} \left(b_i - \sum_{j=1}^{p} a_{ij}\beta_j \right)^2$ subject to, for some $c > 0, \sum_{j=1}^{p} \beta_j^2 < c$, i.e. constraining the sum of the squared coefficients.

Ridge regression thus imposes additional restrictions on the linear model's parameters, β_j's. In this instance, we are adding a penalty term to the β''s in addition to reducing the residual sum of squares. This penalty term is equal to the squared norm of the β vector times the predetermined constant λ. This implies that the optimization function is penalized if the β_j's assume big values. To reduce the penalty term, we would prefer to accept lower β_j's or β_j's that are near zero.

In ridge regression, the means of the variables—both dependent and independent—are subtracted, and their standard deviations are divided. Since we need the means to indicate whether the variables in a particular formula are standardized or not, this is a challenge for notation. To keep the presentation simple, we will make the following general observation before ignoring standardization and its confusing terminology.

In terms of standardization, all ridge regression calculations are based on standardized variables. Before being shown, the computed regression coefficients are scaled back to their initial value. However, the ridge trace employs a traditional scale.

A ridge estimator is a shrinkage technique used in ridge regression. In order to provide a result that is more in line with the real population parameters, shrinkage estimators create new estimators that have been shrunk in size. A least squares estimate can be shrunk using a ridge estimator in multicollinear data to improve the estimate.

Regularization in ridge regression includes the application of a penalty to coefficients. When the coefficients are affected by the same factor, shrinkage occurs. This means that no coefficient will be left out when the model is constructed.

Ridge regression's geometric interpretation:

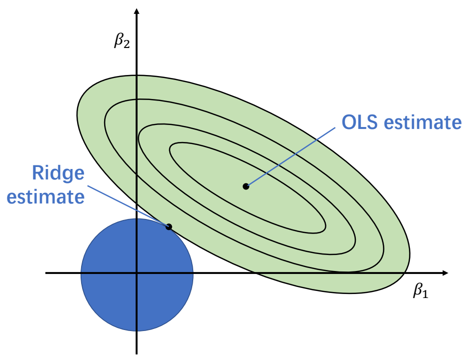


Figure 3.1: Ridge regression

For $p = 2$, the constraint in ridge regression corresponds to a circle,

$$\sum_{j=1}^{p} \beta_j^2 < c \tag{3.4}$$

In ridge regression, we want to simultaneously reduce the size of the circle and the ellipse. The intersection of the ellipse with the circle provides the ridge estimate.

The punishment period and RSS are subject to trade-offs. Perhaps using a big β might improve the residual sum of squares, but doing so would raise the penalty term. Given that lesser β values have a poorer residual sum of squares, you could actually prefer them. The penalty term is similar to a constraint on the β's from an optimization standpoint. Although the norm of the β_j's is now constrained to be less than a certain constant c, the function is still the residual sum of squares. A correlation exists between λ and c. You prefer the β_j's nearer to zero more the greater the λ is. When $\lambda = 0$, the worst case scenario, you would just be performing a standard linear regression. On the opposite extreme, you set all the $\beta's$ to zero as λ gets closer to infinity.

We are aware that the Ordinary Least Square Method (OLS) handles every variable equally. Therefore, the OLS model gets more complex as more variables are included.

In the image below, the OLS model is on the right side with a low bias and a high variance. The OLS model's position is stationary and fixed, however, ridge regression can cause a shift in position.

As we adjust the lambda parameter in ridge regression, the model coefficients will vary.

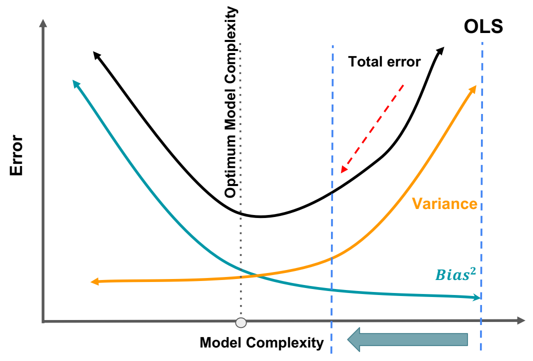


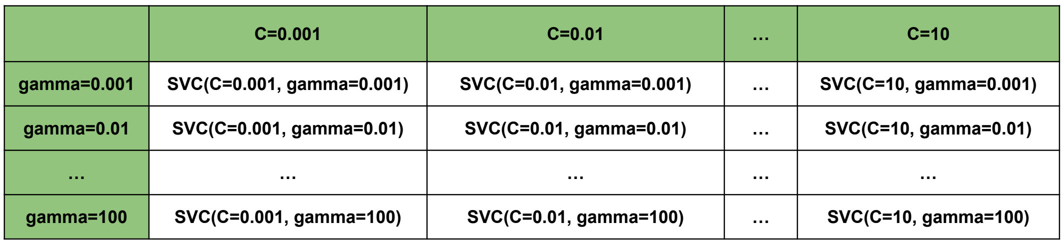
Figure 3.2: Bias

Grid Search

A technique for fine-tuning parameters; exhaustive search: by looping over and attempting each option, the best-performing parameter is the end result. The idea is similar to determining the highest value in an array.

The Grid Search reference method's main drawback is that it takes a long time; the more candidates and parameters there are, the longer it takes. Therefore, a broad range is often established first, followed by refinement.

Table 3.1: Parameters table



	C=0.001	C=0.01	...	C=10
gamma=0.001	SVC(C=0.001, gamma=0.001)	SVC(C=0.01, gamma=0.001)	...	SVC(C=10, gamma=0.001)
gamma=0.01	SVC(C=0.001, gamma=0.01)	SVC(C=0.01, gamma=0.01)	...	SVC(C=10, gamma=0.01)
...	...	...	...	...
gamma=100	SVC(C=0.001, gamma=100)	SVC(C=0.01, gamma=100)	...	SVC(C=10, gamma=100)

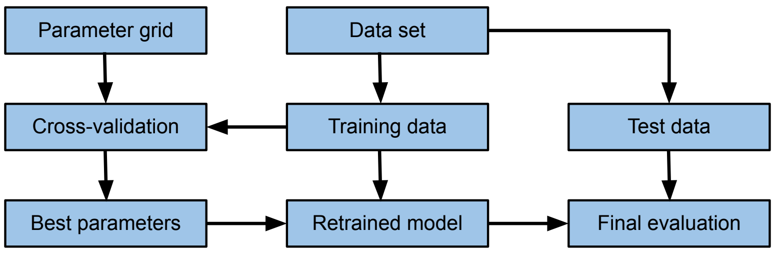


Figure 3.3: Overview of the parameter selection and model assessment procedures using GridSearchCV

Cross validation is used to lessen the chance since the outcomes of the first data partitioning have a significant impact on the grid search method's ultimate performance. Grid search with cross validation is a common parameter assessment technique that combines grid search and cross validation. As a result, sklearn created a class called GridSearchCV that implements fit, predict, score, and other functions. It is used as an estimator using the fit method, which finds the best parameters, and instantiates an estimator with the best parameters.

In general, cross-validation seeks to fulfill:
(1) The training set's percentage should be sufficient, usually larger than 50%.
(2) Equal samples from the training set and test set should be used.

Cross-validation is mainly divided into the following categories:

(1)**Least-one-out cross-validation (LOOCV)**
LOOCV is also n-CV if the dataset has n samples, in which case each sample is utilized as a separate test set and the remaining n-1 samples are used as the training set.

Advantages:
(a) Almost all samples from each round are utilized in the training mode, which results in a distribution that is the most similar to the original sample and more accurate estimates of the generalization error.
(b) The experimental data won't be impacted by random events, allowing the experiment to be repeated.

The high computational cost of LOOCV makes it impossible to apply in reality either each training model runs extremely quickly or the amount of time needed for calculation may be decreased via parallelization. This is because the number of modifications necessary is equal to the total number of samples.

(2)K-folder cross-validation
K subsets, with the first subset serving as a test set and the remaining subsets serving as the training set. Every time the cross-validation is performed, a subset is chosen as the test set. The average cross-validation recognition rate over the course of k repetitions is then used to calculate the outcome.

Benefits: Each sample is verified just once, and all samples are utilized as test and training sets.
Understanding that LOOCV is a unique K-fold Cross Validation (K=N) is not difficult.
As we can see, LOOCV and 10-fold CV really produce estimates of the test MSE that are very close. However, 10-fold CV has a lot lower computational cost and runs much faster than LOOCV. Compared to LOOCV, the computational cost is substantially lower and takes much less time.

3)K * 2 folder cross-validation
K* 2 folder cross-validation is a variant of k-folder cross-validation in which each folder is evenly divided into two sets, s0, and s1, and the first set is trained with

the s1 test before the second set is trained with the s0 test.
Advantage: The test and training sets are both big enough to serve as both training and test sets for each sample.

In this paper, we set a set of parameters: [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000] in the ridge regression model to perform the airline ticket prediction task, using the Grid search with validation method, setting K = 10. It doesn't work well in our dataset, we can see it in Table 4.3.

3.1.2 Lasso Regression

Lasso regression is a type of regularization. It is preferred over regression approaches for more precise prediction. This model makes advantage of shrinkage. Shrinkage is the process by which data values are shrunk towards a central point known as the mean. The lasso method promotes basic, sparse models (i.e. models with fewer parameters). This form of regression is ideal for models with high degrees of multicollinearity or for automating some aspects of model selection, such as variable selection/parameter removal. The L1 regularization technique is used in Lasso Regression.

Regularization
Regularization is a key concept that is utilized to avoid data overfitting, especially when the learned and test data differ greatly. Regularization is carried out by adding a "penalty" term to the best fit produced from the training data in order to attain a lower variance with the tested data. It also limits the effect of predictor variables over the output variable by compressing their coefficients.

In regularization, we typically preserve the same number of features while decreasing the magnitude of the coefficients. To solve this difficulty, we may utilize several sorts of regression approaches that use regularization to lower the size of the coefficients. Ridge Regression and Lasso Regression are the two basic regularization techniques. They differ in how they apply a penalty on the coefficients. When a regression model employs the L1 Regularization approach, it is referred to as Lasso Regression. Ridge Regression is employed when the L2 regularization approach is applied. In the next parts, we will go deeper into these topics.

L1 regularization adds a penalty proportional to the absolute value of the coefficient's magnitude. This kind of regularization can provide sparse models with few coefficients. Some coefficients may reach 0 and so be removed from the model. Greater penalties result in coefficient values closer to zero (ideal for producing simpler models). L2 regularization, on the other hand, does not result in the eradication of sparse models or coefficients. As a result, Lasso Regression is easier to read than Ridge Regression.

Mathematical equation
LASSO regression is an L1 penalized model in which we simply add the weights' L1 norm to our least-squares cost function:

$$\begin{aligned} J(w) &= \sum_{i=1}^{n} (y_i - \widehat{y_i})^2 + \alpha \sum_{j=1}^{m} |w_j| \\ \widehat{y_i} &= w_0 + \sum_{j=1}^{m} X_{ij} w_j \end{aligned} \tag{3.5}$$

We enhance the regularization strength and lower the weights of our model by raising the value of the hyperparameter α. Please keep in mind that the intercept term w_0 is not regularized. It is also worth noting that $\alpha = 0$ corresponds to normal regression analysis. Certain weights can become zero depending on the regularization strength, making the LASSO approach a particularly strong tool for dimensionality reduction.

Similar to what we did in the Ridge regression model, we set a set of parameters: [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000] in the lasso regression model to perform the airline ticket prediction task, using the Grid search with validation method, setting K = 10. The performance is similar to the Ridge regression model, it doesn't work well in our dataset, as we can see in Table 4.3.

3.1.3 K-Nearest Neighbors

The k-nearest neighbor's algorithm (KNN) in statistics was created by Evelyn Fix and Joseph Hodges in 1951 [22] and subsequently improved by Thomas Cover. It is a non-parametric supervised learning technique. [23] Regression and classification are two uses for it. The input in both situations consists of a data set's k closest training samples. Whether KNN is applied for classification or regression determines the results:
(1) The outcome of KNN classification is a class membership. The class that an object is allocated to based on the majority vote of its k closest neighbors is determined by the item's neighbors (k is a positive integer, typically small). The item is simply put into the class of its one nearest neighbor if k = 1.
(2) The result of KNN regression is the object's property value. The average of the values of the k closest neighbors makes up this number.

With KNN, all computation is postponed until after the function has been evaluated and the function is only locally approximated. Since this technique depends on distance for classification, normalizing the training data can significantly increase accuracy if the features reflect several physical units or have distinct sizes. [24][25]

Assigning weights to neighbor contributions may be a helpful strategy for both classification and regression, making the closer neighbors contribute more to the average than the farther neighbors. As an illustration, a typical weighting method assigns each neighbors a weight of 1/d, where d is the distance between the neighbors.

When using KNN classification or regression, the neighbors are chosen from a collection of objects for which the class or object property value is known. Although there is no need for an explicit training phase, this may be considered as the algorithm's training set.

The KNN method has the feature of being sensitive to the local structure of the data.

The k-nearest neighbor algorithm's objective is to locate a query point's closest neighbors so that we may categorize that location. KNN needs a few things in order to do this:

Determine your distance metrics
The distance between the query point and the other data points must be determined in order to discover which data points are closest to a certain query point. These distance measurements aid in the creation of decision borders, which divide query points into several zones. Decision boundaries are frequently represented using Voronoi diagrams.

Despite the fact that there are several distance measurements available, this article will only discuss the following:
Euclidean distance (p=2): This distance metric, which can only be applied to real-valued vectors, is the most often used one. The straight line between the query location and the other point being measured is calculated using the formula below.

$$d(a,b) = \sqrt{\sum_{i=1}^{n} (b_i - a_i)^2} \tag{3.6}$$

Manhattan distance (p=1): The absolute value between two places is measured using this common distance metric. It is also known as taxi distance or city block distance since it is frequently represented by a grid and shows how one may travel between two addresses using city streets.

$$ManhattanDistance = d(a,b) = \left(\sum_{i=1}^{m} |a_i - b_i|\right) \tag{3.7}$$

Minkowski distance: The generalized version of the Manhattan and Euclidean distance metrics is this one. Other distance measures can be created using the parameter p in the formula below. This formula denotes Manhattan distance when p is equal to one and Euclidean distance when p is equal to two.

$$MinkowskiDistance = \left(\sum_{i=1}^{n} |a_i - b_i|\right)^{1/p} \tag{3.8}$$

Hamming distance: This method helps to locate the locations where two vectors do not match and is commonly used with string or Boolean vectors. Because of this, it is also known as the overlap metric. This may be modeled using the formula below:

$$HammingDistance = D_H = \left(\sum_{i=1}^{k} |a_i - b_i|\right) a = b \quad D = 0; a \neq y \quad D \neq 1 \tag{3.9}$$

The k parameter in the KNN method specifies how many neighbors will be examined to establish a particular query point's categorization. The instance will be placed in the same class as its lone nearest neighbor, for instance, if k=1. In order to avoid either overfitting or underfitting, several values of k must be considered while defining it.

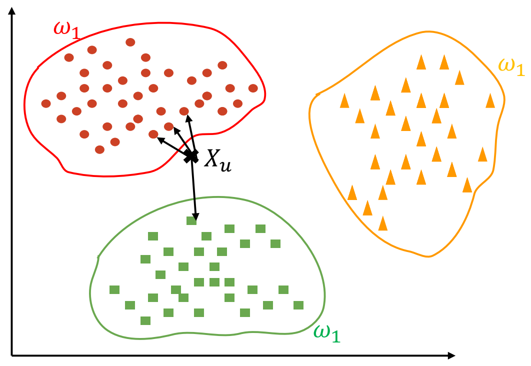


Figure 3.4: Geometrical description of KNN

(1) If k is set too low, the noise will have an impact on the forecast; for instance, if k is set to 1, there will be deviations once the nearest point is noisy. Lowering k also makes the entire model more complicated and more prone to overfitting. The

approximation error of learning will rise if the value of k is selected too high, which is analogous to forecasting using training examples in a broader neighborhood. The total complexity of the model decreases as k increases.
(2) If k == N, it will take all examples, i.e., the most points under a certain categorization in the instances, which has no real-world use for the prediction.

The procedure for taking k: The typical approach is to start with k=1 and use the test set to estimate the classifier's error rate. Every time k rises in value by 1, allowing for one more nearest neighbor, the procedure is repeated. The k that produces the lowest error rate is chosen.

Similar to any machine learning algorithm, KNN has advantages and disadvantages. It might or might not be the best option, depending on the project and application.

Advantages
(1) Simple to use: The method is one of the first classifiers that a novice data scientist will learn due to its clarity and accuracy.
(2) Readily adapts: Because all training data is kept in memory, the algorithm can easily adapt when fresh training examples are introduced.
(3) Few hyperparameters: Compared to other machine learning algorithms, KNN just needs a k value and a distance metric.

Disadvantages
(1) Is not scalable: KNN is a lazy algorithm, which means it uses more memory and data storage than other classifiers. Both in terms of time and money, this may be expensive. Business costs will increase with additional memory and storage, and processing more data may take longer. Different data structures, such as the Ball-Tree, have been developed to alleviate computational inefficiencies; nevertheless, depending on the business challenge, a different classifier may be the best option.
(2) Dimensionality curse: The KNN method frequently suffers from the dimensionality curse, which causes it to underperform with high-dimensional data inputs. The peaking phenomenon describes a situation in which, when the algorithm reaches the optimal number of features, adding more features causes a rise in classification mistakes, particularly when the sample size is less.
(3) Prone to overfitting: KNN is more vulnerable to overfitting as a result of the "curse of dimensionality." The choice of features and dimensionality reduction strategies are used to avoid this, but the value of k can also affect how the model behaves. Higher values of k tend to "smooth out" the prediction values because they average the values over a larger region or neighborhood, whereas lower values of k might overfit the data. However, if k is set too large, the data may not be well suited.

In this study, we establish the number of neighbors in the KNN regression model, starting at 1, and enable one more nearest neighbor to be added each time K grows in value by 1 until it reaches 30 for the ticket prediction job. The outcomes are as follows when there are 10 validations.

3.1.4 Support Vector Regression

What is Support Vector Machine?

Finding a hyperplane in N-dimensional space (N is the number of features) that categorizes the data points clearly is the goal of the support vector machine method.

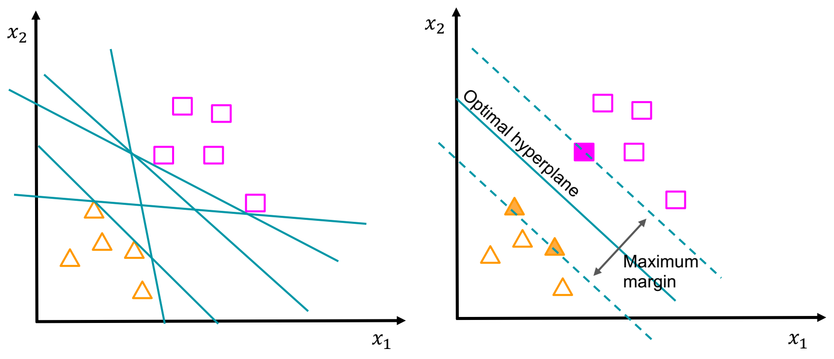


Figure 3.5: Possible hyperplanes

There are a variety of different hyperplanes that might be used to split the two classes of data points. Finding a plane with the greatest margin—that is, the greatest separation between data points from both classes—is our goal. Maximizing the margin distance adds some support, increasing the confidence with which future data points may be categorized.

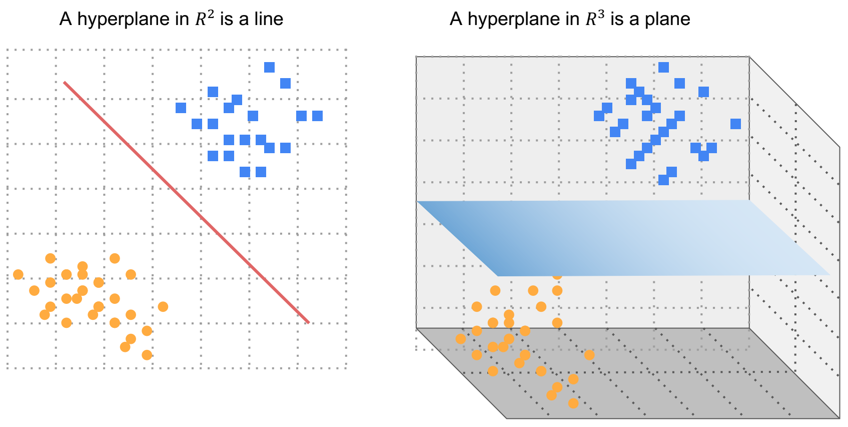

A hyperplane in R^2 is a line
A hyperplane in R^3 is a plane

Figure 3.6: Hyperplanes in 2D and 3D feature space

Decision boundaries known as hyperplanes assist in categorizing the data points. Different classifications can be given to the data points that lie on each side of the hyperplane. Additionally, the amount of features affects how big the hyperplane is. The hyperplane is essentially a line if there are just two input characteristics. The hyperplane turns into a two-dimensional plane if there are three input characteristics. When there are more than three characteristics, it gets harder to imagine.

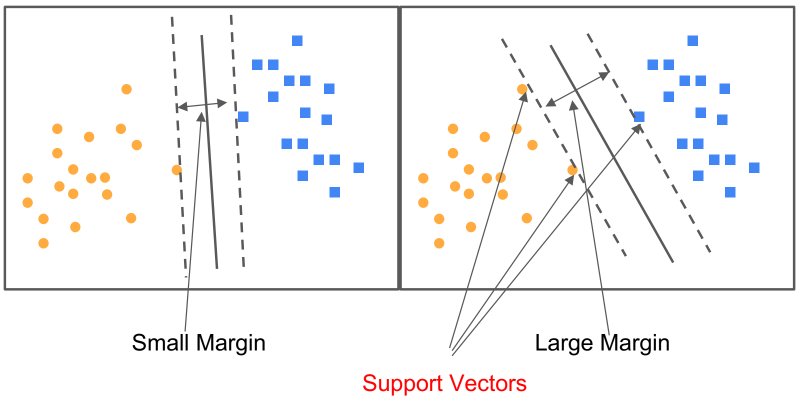


Figure 3.7: Different sizes of margin

Support vectors are data points that are closer to the hyperplane and have an impact on the hyperplane's location and orientation. By utilizing these support vectors, we increase the classifier's margin. The hyperplane's location will vary if the support vectors are deleted. These are the ideas that aid in the development of our SVM.

Large Margin Intuition

In logistic regression, we take the output of the linear function and use the sigmoid function to compress the result inside the range [0,1]. We provide a label of 1 if the squished value exceeds a threshold value (0.5), else we assign a label of 0. In SVM, the output of the linear function is taken into consideration. If the output is more than 1, it is associated with one class, and if it is less than 1, it is associated with a different class. We get this reinforcing range of values ([-1,1]) that serves as a margin since the threshold values in SVM are altered to 1 and -1.

Cost Function and Gradient Updates

The goal of the SVM method is to increase the distance between the data points and the hyperplane. Hinged loss is the loss function that aids in maximizing the margin.

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases} \qquad c(x, y, f(x)) = (1 - y * f(x))_+ \tag{3.10}$$

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

If the projected value and the actual value have the same sign, there is no cost. If not, we next determine the loss value. The cost function additionally receives a regularization parameter from us. The regularization parameter's goal is to strike a compromise between margin maximization and loss. The cost functions appear as follows when the regularization parameter has been added.

$$\min_w \lambda \|w\|^2 + \sum_{i=1}^{n} \left(1 - y_i \langle x_i, w \rangle\right)_+ \tag{3.11}$$

Loss function for SVM

Now that we know the loss function, we can find the gradients by taking partial derivatives with respect to the weights. We may modify our weights using the gradients.

$$\begin{gathered} \frac{\delta}{\delta w_k} \lambda \|w\|^2 = 2\lambda w_k \\ \frac{\delta}{\delta w_k} \left(1 - y_i \langle x_i, w \rangle\right)_+ = \begin{cases} 0, & \text{if } y_i \langle x_i, w \rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases} \end{gathered} \tag{3.12}$$

Gradient Update - Misclassification

We only need to update the gradient from the regularization parameter when there is no misclassification, which is to say when our model predicts the class of our data point accurately.

$$w = w - \alpha \cdot (2\lambda w) \tag{3.13}$$

Gradient Update - No misclassification

In order to execute a gradient update when there is a misclassification, or when our model incorrectly predicts the class of a data point, we add the loss along with the regularization parameter.

$$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w) \tag{3.14}$$

Gradient Update - Misclassification

Regression is performed using support vectors

SVMs may be utilized to tackle regression problems, after all. We are aware that the decision boundary seen in the previous picture may be utilized to distinguish between the two groups. There are two alternative solutions to this problem, which is referred to as discrete one: class 0 for everything above the line and class 1 for anything below the line. Classification issues are effective illustrations of challenges with discrete machine learning.

There is no such thing as a result being "above the line" or "below the line" in regression since one input value is mapped to real numerical output, a number. Instead, we must produce the result using the border itself. However, since a precise, maximum-margin fit is extremely challenging in those circumstances, the problem would exponentially worsen if we attempted to discover a perfect boundary for our continuous data.

This emphasizes the importance of the boundary's accuracy and temporal complexity, although Support Vector Machines may be used to conduct what is known as Support Vector Regression (SVR). As a result of the requirement that it learns to calculate continuous outputs, an error-accepting region known as the error tube is captured around the maximum-margin decision boundary. Finding a tube that is as short as feasible while without significantly sacrificing model complexity or training time is the aim of SVR.

Imagine that all of the samples in the above image are just samples and represent some $x \to y$ mapping from one continuous input to a continuous output value rather than being members of any specific class. It goes without saying that you want the regressed function to fall somewhere in the middle of the samples when executing a regression job. With support vectors towards the center of your dataset, Support Vector Machines will regress a function that translates those inputs to outputs, making them a strong fit for (linear, and if not linear, using some kernel function with the kernel technique) regression issues.

Epsilon-SVR and nu-SVR

In reality, there are two varieties of support vector regression: epsilon-based SVR($\epsilon - SVR$)and nu-SVR ($\nu - SVR$). The degree of control they provide you over the regression problem is how they vary (StackExchange, n.d.):

You can control the overall number of support vectors utilized with $\nu - SVR$, but not necessarily the amount of tolerable error (often yielding smaller but possibly worse models).

When utilizing $\epsilon - SVR$, you can adjust the allowable level of error but not always the total number of support vectors (often yielding better but large models).

In this paper, we set a set of regularization parameters C: [1e0, 1e1, 1e2, 1e3] in the support vector regression model to perform the airline ticket prediction task, using the Grid search with validation method. The strength of the regularization is inversely proportional to C and must be strictly positive. The penalty is a squared l2 penalty. And we also use a set of Gaussian kernel coefficients in our model, to get good performance. This model is time-consuming but it doesn't work well in our dataset, we can see it in Table 4.3.

3.1.5 XGBoost

XGBoost, also known as Extreme Gradient Boosting, is a supervised learning technique that uses an ensemble approach based on the Gradient boosting algorithm. It is a scalable end-to-end tree-boosting system, widely used by data scientists to achieve state-of-the-art results on many machine learning challenges. It can solve both classification and regression problems and achieve better results with minimal effort.

XGBoost's key features:
(1)Regularization: To minimize overfitting, XGBoost provides a variety of regularization penalties. Penalty regularizations result in successful training, allowing the model to generalize well.
(2)Missing Value: It is built in such a way that it can deal with missing values. It identifies and apprehends trends in missing values.

(3)Flexibility: It provides assistance for objective functions. They are the function used to assess the model's effectiveness, and they may also manage user-defined validation measures.
(4)Cross-Validation: Built-in and comes out-of-the-box.
(5)Save and load: It provides the ability to save the data matrix and reload it later, saving resources and time.

How does the XGBoost algorithm work?
Think about a function or an estimate. To begin, we create a series based on the function gradients. The equation below represents a specific type of gradient descent. Because it depicts the Loss function to minimize, it indicates the direction in which the function drops. is the rate of change fitted to the loss function; it corresponds to the gradient descent learning rate. is intended to accurately replicate the loss's behavior.

$$F_{x_t+1} = F_{x_t} + \epsilon_{x_t} \frac{\partial F}{\partial x} (x_t) \tag{3.15}$$

To iterate over the model and discover the best definition, we must represent the entire formula as a sequence and construct an effective function that will converge to the function's minimum. This function will act as an error measure, assisting us in reducing loss and maintaining performance over time. The series converges to the function's minimum. This notation specifies the error function used while evaluating a gradient-boosting regression.

$$f(x, \theta) = \sum l \left(F \left((X_i, \theta), y_i \right) \right) \tag{3.16}$$

In this study, we set the depth of the tree to be 3, the learning rate of the model generated by each iteration is 0.1, the number of sub-models is 100, and the loss function is set to squared loss.

3.1.6 Decision Tree

A decision tree is a supervised machine-learning approach that uses a tree structure resembling a flowchart to represent decisions, outcomes, and predictions. Such a tree is created using an algorithm (series of if-else statements) that determines how to divide, categorize, and display a dataset in accordance with certain criteria.

Each internal node in a decision tree represents a test on a dataset feature (such as the outcome of a coin toss, heads or tails), and each leaf node in a decision tree represents an outcome (such as the choice made after simulating all features), and branches in a decision tree represent the decision rules or feature conjunctions that result in the corresponding class labels.

Regression and classification problems are frequently solved using decision trees. In classification issues, target variables storing discrete values are used by tree models to label or classify an entity. However, in regression issues, the objective variable has continuous values (actual numbers), and tree models are employed to predict results for unobserved data.

For machine learning and data mining, decision trees that employ a predictive modeling method are frequently used. By taking into account data from the sample population, the model generates precise judgments about the sample's goal value (represented by leaves) (illustrated via branches).

How does a decision tree work?

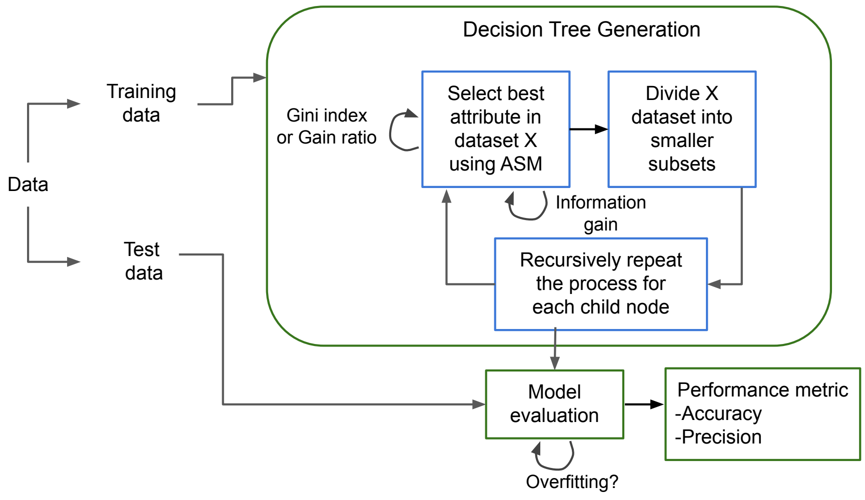


Figure 3.8: The working mechanism of the decision tree

By using orthogonal splits to create decision areas, tree-based approaches impose if-else conditions on features. When creating such tree-based solutions, it is essential to comprehend how these splitting conditions are created and how many times the decision space has to be divided.

The selection of the features to divide, the values of the feature split, and the point at which to cease splitting are all critical issues that good decision trees handle. Let's examine each requirement in further depth.

Splitting features

A top-down greedy method is used by decision trees to determine the appropriate feature split. In greedy approaches, all points in the same decision area are divided, and further splits are done methodically. The branch (sub-tree) that results has a higher metric value than the preceding tree.

For many classification and regression problems, common cost functions include:
For classification problems:

Entropy: Entropy quantifies the degree of uncertainty in the processed information and determines its unpredictability. The more entropy there is, the harder it is to draw inferences from a situation. The general goal is to reduce entropy and create decision zones that are more homogenous and contain data points that are members of the same class. Entropy is given by the formula,

$$\text{Entropy } = -\sum_{i=1}^{n} {p_i}^{*} \log_2 (p_i) \tag{3.17}$$

Where p = probability of an element or class in the data

Gini index: The metric calculates the odds that a randomly chosen data point would be incorrectly labeled by a certain node. The Gini index serves as the cost function for assessing feature splits in a dataset.
The Gini index is given by the formula,

$$Gini = 1 - \sum_{i=1}^{n} (p_i)^2 \tag{3.18}$$

Where p is the likelihood that an item will be placed in a specific class

Information gain (IG): The Gini index or reduction in entropy as a result of a feature split is measured using the IG metric. When tree-based algorithms utilize Entropy or the Gini index as a criterion, informative splits are obtained. In other words, a split like that only decreases the needs by a certain percentage.

The formula provides information gain,
Information Gain = Entropy (X: before splitting) – Entropy (each feature: after splitting)

For regression problems:

Residual sum of squares: For each data point in a decision region, the metric is equal to the total of the squared differences between the observation (target class) and the mean response. To reduce the residual sum of squares, feature splits are chosen.

In order to maximize information gain or decrease the residual sum of squares, the feature splits are designed so that the feature value is higher. The procedure is repeated to perform more best splits.

The tree tends to get increasingly complicated as the splitting process goes on, and the algorithm unavoidably learns noise in addition to signals from the dataset. Due to overfitting, the decision tree can only be used for the training dataset and is unable to generalize to other untrained or untested datasets. As a result, decision trees employ pruning procedures.

By removing tree portions with poor predicting ability, pruning techniques lower the overfitting component. Removing the weak or irrelevant rules makes the decision tree simpler. There are two methods to do this:
(1) Limit the decision tree's maximum depth
(2) Impose a minimum sample size per decision space restriction.

Cost complexity pruning is another technique for pruning. Here, the sub-trees are removed by adding a new term to the cost function. Simply said, the traditional recursive splitting strategy is used to create a large decision tree at first. After the tree has been created, cost complexity pruning is used to determine the optimal order for sub-trees and to remove unimportant sub-trees based on weights. In Lasso Regression, when the model complexity is regularized by penalizing weights, this technique is evident.

The decision tree's operation is made simpler by the following algorithm:
Step I: Begin the decision tree with X as the root node. X contains the entire dataset in this case.
Step II: Using the 'attribute selection measure (ASM),' choose the optimal attribute in dataset X to partition it.
Step III: Subdivide X into subsets with the best qualities' potential values.
Step IV: Create a tree node containing the best attribute.
Step V: Recursively generate new decision trees using the subsets of the dataset X established in step III. Continue the process until you can no longer categorize the nodes. The final node is referred to as a leaf node.

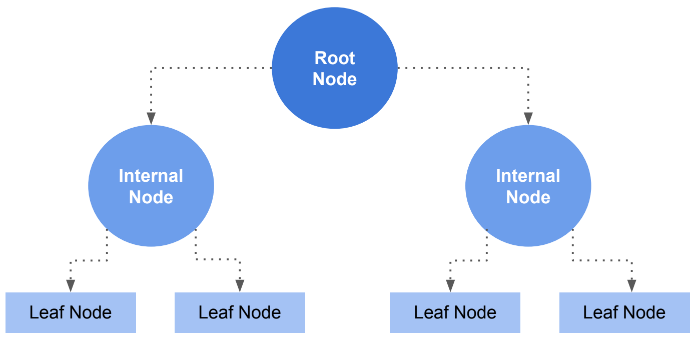


Figure 3.9: The elements in a decision tree

The attribute selection measure in the preceding procedure refers to a form of heuristic used to determine the splitting criterion that optimally splits a given dataset (X) into different subsets. In other words, it specifies how datasets or subsets at a certain node will be divided.

Different strategies may be used by decision trees to split and subdivide a node into additional sub-nodes. Technically, the decision tree splits the nodes using all of the relevant factors, but it ultimately selects the split that produces the most homogenous sub-nodes. Here, choosing a method is greatly influenced by the nature of the target variable.

Let's examine some of the popular algorithms that decision trees employ.

(1) Iterative Dichotomiser 3 (ID3)
With the entire dataset "X" as the root node, the Iterative Dichotomiser 3 method creates decision trees. It then repeats the instructions for each attribute and divides the data into subgroups using metrics like entropy or information gain. After splitting, the method iteratively considers the properties not taken into account in the first splits for each subgroup.
When continuous variables are taken into account, the ID3 method typically overfits the data, and dividing the data can be time-consuming. The ID3 algorithm is employed in several machine learning and natural language processing fields.

(2) C4.5
An improved version of the ID3 algorithm is C4.5. It regards samples with classification as data. To separate the nodes, the method employs normalized information gain. Additionally, the character with the greatest information gain determines the final data split.

Contrary to the ID3 method, C4.5 effectively handles both discrete and continuous properties. Additionally, after creating the final decision tree, the algorithm goes through a pruning procedure in which all the branches with little significance or relevance are eliminated.

(3) Classification and regression trees (CART)
Both classification and regression issues can be resolved with the CART method. Additionally, it divides the datasets using the Gini index measure, as opposed to the ID3 and C4.5 algorithms, which employ information gain or entropy and gain ratio.
The goal of the greedy strategy used in the CART splitting procedure is to minimize the cost function. The purity of the leaf nodes is calculated for classification tasks using the Gini index as a cost function. To select the most accurate forecast, the method uses the total squared error as the cost function for regression.

(4) Chi-square automatic interaction detector (CHAID)
All sorts of variables, including nominal, ordinal, and continuous ones, are revealed by the CHAID method. The CHAID method builds a tree that shows the optimum way to combine variables in order to reveal the result for the specified dependent variable.
The CHAID algorithm uses each categorical predictor individually to build a tree, taking into account all potential combinations, then repeats the procedure until no more splitting is possible. This suggests that the ideal result is ultimately attained. Finding a root node for the tree that reflects the target or dependent variable is the first step in the decision tree creation process. The target variable is further separated into numerous parent nodes. Finally, using statistical techniques, these nodes are split into child nodes.
The merging of variables in the CHAID analysis is based on tests; for instance, the 'F-test' is utilized if the dependent variable is continuous. If the dependent variable is categorical, the "chi-square test" is also employed.

(5) Multivariate adaptive regression splines (MARS)
When there is non-linear data in a regression problem, MARS techniques are frequently used. This adaptive spline approach divides the data into sections and applies a linear regression model to each section separately.
MARS serves as a building block for nonlinear modeling and is closely related to multiple regression models. The method, which is a CART adaption, enables the addition of additional terms to the current model.

Advantages
(1) Decision trees are simple to learn and use because of their Boolean logic and

visual representations. A decision tree's hierarchical structure also makes it simple to understand which traits are most crucial, which is not necessarily the case with other methods, such as neural networks.
(2) Hardly any data preprocessing is necessary: Decision trees are more adaptable than other classifiers due to a variety of qualities. It may work with a variety of data formats, such as discrete or continuous values, and it can employ thresholds to turn continuous values into categorical values. It also has the ability to handle missing value values, which may be difficult for other classifiers like Naive Bayes.
(3) More adaptable: Compared to certain other algorithms, decision trees are more adaptable since they may be used for both classification and regression problems. The method will only pick one of the characteristics to split on if two variables are highly linked, as it is indifferent to the underlying connections between attributes.

Disadvantages
(1) Complex decision trees are prone to overfitting and poor generalization to fresh data. Pre- or post-pruning procedures can be used to prevent this situation. When there is insufficient data, pre-pruning stops the growth of the tree, but post-pruning eliminates subtrees with insufficient data after the tree has been built.
(2) High variance estimators: Small differences in the data themselves might result in highly diverse decision trees. Decision trees' variance can be decreased via bagging, which is the averaging of estimates. This method has drawbacks since it might provide predictors that are strongly linked.
(3) More expensive: Compared to other algorithms, decision trees may be more expensive to train since they employ a greedy search strategy during building.
(4) Insufficient support for scikit-learn: Python-based Scikit-learn is a well-known machine learning package. Although there is a Decision Tree module in this library, categorical variables are not yet supported by the implementation.

In the Decision tree regression model, we used the mean squared error function to measure the quality of a split, which is equal to variance reduction as a feature selection criterion and minimizes the L2 loss using the mean of each terminal node. And we used the "best" strategy to choose the split at each node. Meanwhile, when we set the maximum depth of the tree, nodes are expanded until all leaves are pure or until all leaves contain less than *min_samples_split* samples. And then we considered *min_samples_split* as the minimum number required to split an internal node.

3.1.7 Random Forest Regression

A general model made up of several decision trees is known as a random forest. Each decision tree's predictions are averaged to provide the forecasts. A random forest model is a collection of decision tree models, just like a forest is a collection of trees. As a result, random forests are a robust modeling approach that is far more effective than individual decision trees.

A random forest trains each tree using a portion of the data. Instead of depending just on one decision tree to determine the outcome, the core idea behind it is to merge many decision trees. Since each decision tree is perfectly trained for a particular sample of data, the variance of the results is low even though each decision tree has a high variance. However, when we combine all the decision trees simultaneously, the output is not dependent on just one decision tree but rather on several. The average of all decision tree outputs is used as the final result for the regression issue.

4 Steps to Construct a Random Forest

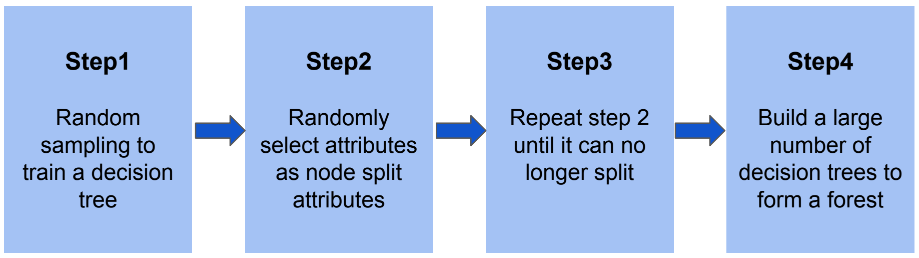

Step1
Random sampling to train a decision tree
Step2
Randomly select attributes as node split attributes
Step3
Repeat step 2 until it can no longer split
Step4
Build a large number of decision trees to form a forest

Figure 3.10: Steps to build a random forest

(1) To obtain N samples, a sample with a sample size of N is pulled N times with put-back, yielding 1 sample each time. As the samples at the decision tree's root node, these chosen N samples are utilized to train the tree.
(2) When each node of the decision tree has to be divided and each sample has M attributes, m attributes are randomly picked from these M attributes, meeting the criterion m M. The splitting attribute of the node is then chosen from among these m attributes using a technique (such as knowledge gain).
(3) Each node is divided in accordance with step 2 of the development of the decision tree (it is easy to understand that if the next selected attribute is the same attribute that was used in the splitting of its parent node, the node has already reached the leaf node and does not need to continue splitting). It continues until splitting is impossible. Take note that no pruning is done at any point while the decision tree is being formed.
(4) Create a huge number of decision trees in accordance with steps 1 through 3 to

create a random forest.

Using many models that have been trained on the same data and averaging their findings to provide a more accurate prediction or classification is known as ensemble learning. The assumption behind ensemble learning is that the faults of each model—in this example, a decision tree—are distinct from one another and independent of one another.

Bootstrapping involves randomly selecting subsets of a dataset over a certain number of repetitions and variables. To provide a more potent outcome, these findings are then averaged. An example of an applied ensemble model is bootstrapping.

The bootstrapping Random Forest approach combines ensemble learning techniques with the decision tree framework to generate numerous randomly chosen decision trees from the data. The outputs are averaged to get a new result, which frequently produces accurate predictions and classifications.

Random Forest Regression Model:
Our random forest regression model will be trained using the sklearn module, especially the Random Forest Regression function. Numerous possible parameters that we may choose from for our model are listed in the Random Forest Regression documentation.

The following list of crucial variables is highlighted:
n_estimators — the number of decision trees that the model will use.
criterion — the criteria (loss function) used to determine model results can be chosen using this variable. Loss functions like mean squared error (MSE) and mean absolute error is available to us (MAE). MSE is the default value.
max_depth — this determines each tree's maximum feasible depth.
max_features — the most factors the model will take into account when deciding on a split.
bootstrap — The model adheres to bootstrapping principles since the default setting for this is True (defined earlier).
max_samples — This option presupposes that bootstrapping is set to True; otherwise, it has no effect. This number determines the greatest size of each sample for each tree when True is present.
Other important parameters are *min_samples_split*, *min_samples_leaf*, *n_jobs*, and others that can be read in the sklearn's.

The random forest approach has a variety of significant benefits and drawbacks when used for classification or regression tasks. Some of them consist of:

Key Benefits

(1) Less chance of overfitting: Decision trees have a propensity to closely match all the samples contained in training data, which increases the possibility of overfitting. The classifier won't, however, overfit the model when there are a large number of decision trees in a random forest since the averaging of uncorrelated trees reduces the total variance and prediction error.
(2) Flexibility: The Random forest is a well-liked approach among data scientists since it can accurately handle both classification and regression jobs. The random forest classifier benefits from feature bagging by maintaining accuracy even when some of the data is missing, which makes it a useful tool for guessing missing values.
(3) Simple evaluation of feature contribution: Random forest makes it simple to assess variable contribution. There are several methods for determining feature relevance. To gauge how much the model's accuracy declines when a particular variable is removed, the Gini importance and mean drop in impurity (MDI) are frequently utilized. A different significance metric is permutation importance, often known as mean decrease accuracy (MDA). By randomly permuting the feature values in OOB samples, MDA can determine the average reduction in accuracy.

Key Challenges

(1) Process that takes a long time: Because random forest methods can handle big data sets, they can make predictions that are more accurate. However, because they must compute data for each decision tree, they can take a long time to process data.
(2) More resources are needed: Because random forests analyze bigger data sets, more resources are needed to store that data.
(3) More complex: When compared to a forest of decision trees, a single one's prediction is simpler to understand.
Numerous sectors have used the random forest algorithm to help them make better business decisions. Examples of use cases are:
(4) Finance: This method is favored over others since it takes less time to handle and pre-process data. It may be used to assess high-risk consumers for fraud and issues with option pricing.
(5) Healthcare: The random forest approach is used in computational biology to solve issues including classifying gene expression, finding biomarkers, and annotating sequences. Doctors can therefore estimate pharmacological reactions to certain drugs.
(6) E-commerce: Cross-selling may be accomplished by using recommendation engines.

Here, we use the random size search with a cross-validation approach to discover the best parameters, set the number of cross-validations to 5, and utilize these parameters to construct a random forest regression model. The minimum number of samples needed to split an internal node is set to [2, 5, 10], while the minimum number of samples needed to be at a leaf node is set to [1, 2, 4]. We set the number of trees in the forest to [100, 200, 300, 400, 500].

RandomizedSearchCV

We employ the Random Search with a cross-validation approach to determine the ideal parameters for the Random Forest regression model:

The various kernel functions (Kernals) in SVMs and K values in KNN algorithms are examples of so-called model settings, often known as hyperparameters of the model. The number of hyperparameters, etc., is often limitless. The combination of hyperparameters can be adjusted using a heuristic search approach, in addition, to manually validating numerous pre-defined combinations of hyperparameters in a constrained amount of time. Grid search is the name of this heuristic hyperparameter search technique.

If there are only a few hyperparameters (three or four or fewer) while searching for them, grid search, an exhaustive search method, can be used. However, if we continue to utilize grid search and there are a lot of hyperparameters, the search time would grow tremendously.

In order to find lower values, a random search strategy that randomly explores tens or hundreds of points in the hyperparameter space has been developed. The random search technique produces somewhat better results than the sparse grid method, according to studies, and it is faster than the sparse grid method described above.

While RandomizedSearchCV employs a technique very similar to the class GridSearchCV, he does so by selecting a predetermined number of random combinations of one random value for each hyperparameter rather than trying all possible possibilities.

This technique has two benefits:
(1) The random search will examine 1000 possible values of each hyperparameter if you run it 1000 times (instead of searching only a few values of each hyperparameter, like the grid search).

(2) By adjusting the number of searches, you can simply manage how much computation goes into the hyperparameter search.

In reality, RandomizedSearchCV and GridSearchCV are identical; however, RandomizedSearchCV samples the parameter space randomly, and for parameters with continuous variables, it samples the parameters as a distribution. Grid search is unable to accomplish this, and its ability to do so depends on the value chosen for the n iter argument.

3.2 Deep Neural Network

Neural networks are a collection of algorithms that are intended to identify patterns and are loosely based on the human brain. They categorize or group raw input to understand sensory data using a form of machine perception. All real-world data, including pictures, sounds, texts, and time series, must be converted into vectors in order for them to detect the patterns, which are numerical and included within.

We can categorize and cluster data using neural networks. They may be viewed as a layer of grouping and classification on top of the data you manage and store. They aid in organizing unlabeled data into groups based on similarities between example inputs, and when given a labeled training set, they categorize data. You might conceive of deep neural networks as parts of broader machine-learning systems that involve algorithms for reinforcement learning, classification, and regression (since neural networks can also extract features that are supplied to other algorithms for clustering and classification).

What defines a Neural Network?

The inner workings of biological brains serve as the basis for the construction of neural networks. By sending input data through numerous layers of what are known as perceptrons (think "neurons"), each of which transforms the input using a different set of functions, these models mimic the actions of linked neurons. The simplest component of a neural network, the perceptron, will be broken down in this section.

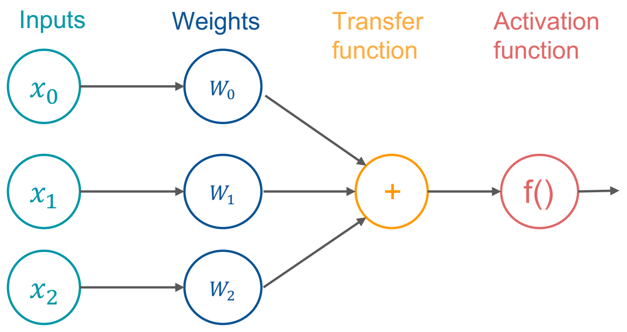


Figure 3.11: The structure of a perceptron

Scalar multiplication, a summation, and finally a transformation utilizing a unique equation known as an activation function are the three primary mathematical processes that commonly make up a perceptron (above). We may group several perceptrons together to simulate a brain, which is known as a neural network, as each perceptron represents a single neuron in the brain.

Input: Our features' measurements make up the inputs.

Weights: Scalar multiplications are represented by weights. It is their responsibility to evaluate the significance and directionality of each input. The transfer function, the next part of the perceptron, will then get these values.

Transfer Function: Unlike the other components, the transfer function accepts a variety of inputs. In order to apply the activation function, the transfer function must merge many inputs into a single output value. Typically, this is accomplished by adding up all of the transfer function's inputs.

However, the value undergoes an activation function transformation before being delivered as the perceptron's final output.

The number from the transfer function will be changed into a value that dramatizes the input using an activation function. The activation function is frequently non-linear. The perceptron may be made more complicated by adding non-linearity, which prevents the output from changing linearly with the inputs. Here are two typical activation processes.

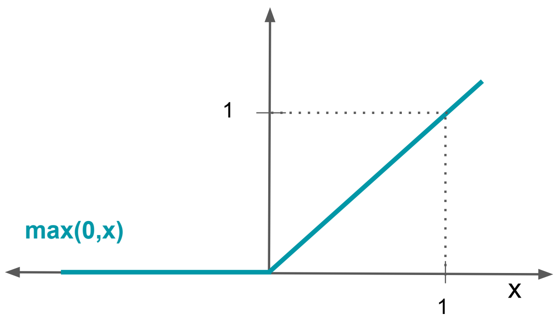


Figure 3.12: Geometric representation of the RELU function

ReLU is a straightforward function that selects the maximum by comparing zero with the input. In other words, any negative input results in zero, and positive inputs have no impact. This is handy for reducing linearity without having to perform any laborious computations or in circumstances where negative numbers don't make much sense.

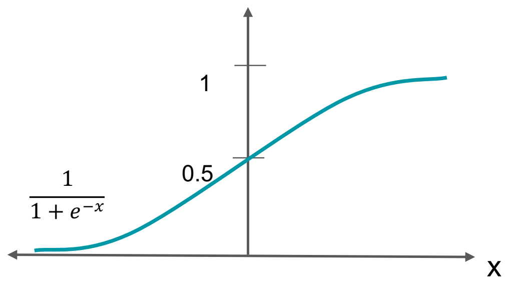


Figure 3.13: Geometric representation of the sigmoid function

The sigmoid function is effective in dividing values into several thresholds. It is especially helpful for values like z-scores, where numbers close to the mean (zero) should be carefully examined since a tiny change close to the mean may have a huge impact on certain behavior, but where values far from the mean likely signal the same thing about the data.

The input is made more dramatic by the activation function, which is a non-linear function. In other words, inputs that are closer to zero tend to be more impacted than ones that are far from zero. This will enable us to select more defined decision limits.

The activation function is always the same since we choose it before training our model. In the course of testing countless models, we do not toggle this parameter. It only occurs with the weights.

Bias: Every perceptron uses the same input, which never changes. It is multiplied by the weight in the same way as the other inputs and serves to enable independent up- and down-shifting of the value prior to the activation function. Due to the fact that they are not required to additionally attempt to balance the entire sum to be close to 0, the other weights—for the actual inputs, not the bias—can be more precise.

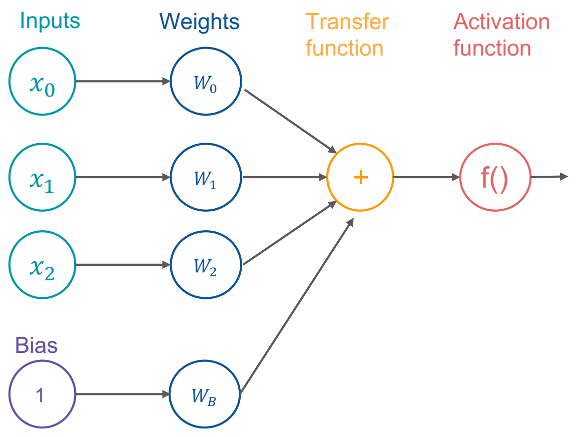


Figure 3.14: The structure of a perceptron with bias

To be more specific, bias might shift graphs like the left graph to something like the following graph:

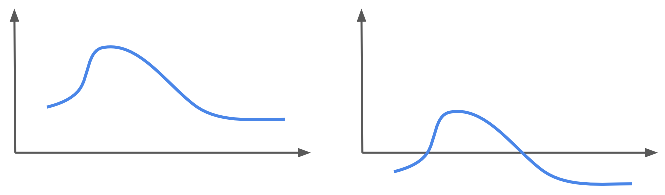


Figure 3.15: The impact of bias

We've now created a model that imitates the brain's neurons, we can create complex multi-dimensional equations by altering a few weights. They can be summarized by the following equation:

$$f\left(\mathrm{x}_0 \cdot \mathrm{w}_0 + \mathrm{x}_1 \cdot \mathrm{w}_1 + \mathrm{x}_2 \cdot \mathrm{w}_2 + 1 \cdot \mathrm{w}_\mathrm{B}\right) \tag{3.19}$$

From this point forward, every single green perceptron contains every element we have seen thus far: inputs, bias, weights, transfer function, and an activation function.

Multi-Layer Perceptrons

We only connect the output of one perceptron to the input of another to depict a network of perceptrons. Many of these perceptrons are linked together in chains that flow from one end to the other. This is referred to as a Multi-Layer Perceptron (MLP), and as the name implies, there are several interconnected layers of perceptrons in it. We will examine fully-connected MLPs for simplicity, in which every perceptron in one layer is coupled to every perceptron in the following layer.

A layer is nothing more than a line of disconnected perceptrons. In an MLP, perceptrons are connected to all other perceptrons in the layers above and below them but not to any other perceptrons in the same layer. Let's examine an MLP that has two input values, two hidden layers, and a single output value. Say there are two perceptrons in the first hidden layer and three in the second.

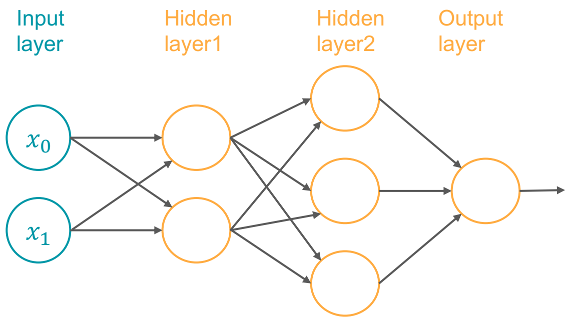


Figure 3.16: An MLP with two input values, 2 hidden layers, and an output of a single value

Here, each perceptron will take in the inputs (arrows heading in the direction of the circle), carry out the actions outlined in the previous section, and then advance the output (arrow pointing out of the circle). This is repeated several times to produce equations that are progressively more complicated while still taking the same data into account repeatedly to provide reliable predictions. Because we are unsure of the significance of the equations it chooses, the strategy we're presenting is sometimes referred to as a "black box" approach.

The layers between the input layer and the output layer are referred to as "hidden" layers because, once the values are fed from the input, it is not beneficial to see how they are converted until they leave the final output node. This is due to the fact that our model's performance is never assessed using these interim variables (i.e. getting error values for predictions made on sample data). We may construct equations that are even more complex than those produced by a single perceptron by combining numerous of these perceptrons.

3.2.1 Fully Connected Neural Network

An information system called the fully-connected ANN is created by conceptually abstracting, streamlining, and modeling the essential elements of the genuine neural network in the human brain [26].

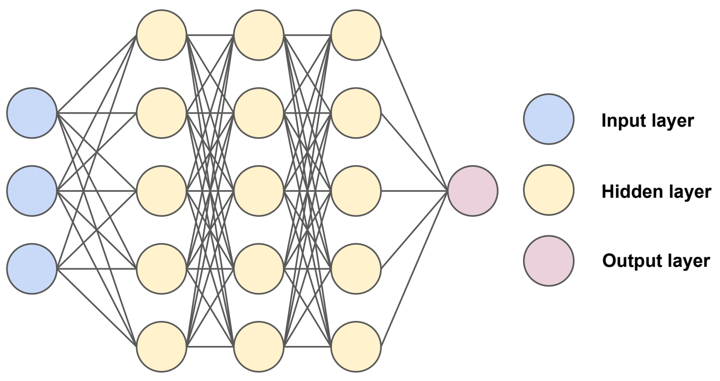


Figure 3.17: Fully Connected Neural Network

The BP neural network is a multi-layer feedforward neural network that was trained using the error backpropagation technique. It is presently the most popular fully-connected ANN. Usually, a differentiable function, the transfer function utilized by the BP neural network's neurons may actualize any nonlinear mapping between the input and the output. Consequently, a wide range of applications exists for the BP neural network in pattern recognition, risk assessment, intelligent prediction, etc. [27].

The BP neural network excels at multi-dimensional function mapping and can classify patterns of any complexity. Its objective function is the square of the network error, and the gradient descent method is used to get its least value [28]. This is how the calculations are done in detail: (1) Random values are used to initialize the connection weight value and threshold. (2) The parameters chosen by the input mode and output mode are used to calculate the output of each unit of the hidden layer and the output layer. The ReLU is utilized in this work as the hidden layer's activation function. The output layer's activation function uses the Tanh.

The ReLU can be defined as:

$$f(x) = \max(0, x) \tag{3.20}$$

The Tanh can be defined as:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.21}$$

The modification of the neuron threshold:

$$\theta_k(t+1) = \theta_k(t) + \eta_t \sigma_k, (t = 1,2,\ldots,p; k = 1,2,\ldots,l) \tag{3.22}$$

$$\theta_j(t+1) = \theta_k(t) + \eta_t \sigma_j, (t = 1,2,\ldots,p; j = 1,2,\ldots,l) \tag{3.23}$$

where θ_j and θ_k are the $j-th$ node's threshold values and the $k-th$ node's output layer, respectively; σ_j and σ_k are the $j-th$ node's hidden layer errors and the $k-th$ node's output layer errors, respectively. η_t is the $t-th$ training iteration's learning rate.

The weight update formula is as follows:

$$\omega_{jk}(t+1) = \omega_{jk}(t) + \Delta\omega_{jk}(t), (t = 1,2,\ldots,p) \tag{3.24}$$

$$v_{ij}(t+1) = v_{ij}(t) + \Delta v_{ij}(t), (t = 1,2,\ldots,p) \tag{3.25}$$

where t is the total number of training iterations, v_{ij} is the updating of weight values from the input layer to the hidden layer, and ω_{jk} is the updating of weight values from the hidden layer to the output layer.

To train the neural network, go back to step two and keep updating the learning input mode until the number of training instances approaches the desired number. Figure 3.18 depicts the fundamental flow of the aforementioned computation method.

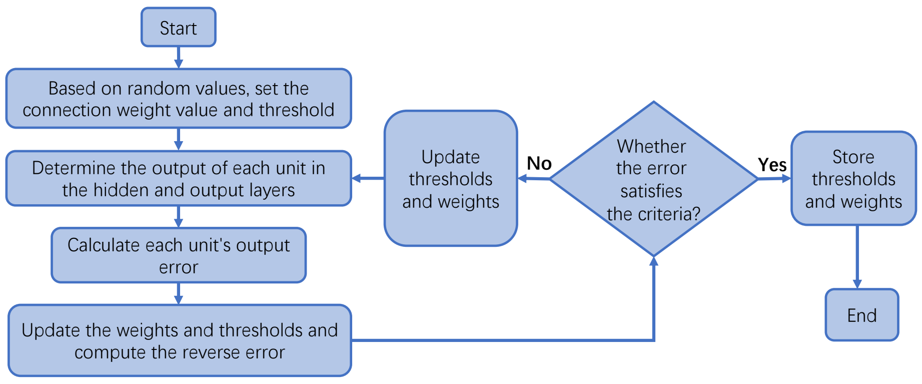

Start
Based on random values, set the connection weight value and threshold
Determine the output of each unit in the hidden and output layers
Calculate each unit's output error
Update the weights and thresholds and compute the reverse error
Update thresholds and weights
No
Whether the error satisfies the criteria?
Yes
Store thresholds and weights
End

Figure 3.18: The working mechanism of a Fully Connected Neural Network

In this part, we built a fully connected neural network, which contains seven linear layers, each of which is followed by batch normalization and ReLu. There are thirteen features in the airfare dataset, so we set the number of neurons in the input layer is thirteen. And the number of neurons in the hidden layer and output layer are 1024 and 1 respectively. We set the batch size as 128. The performance is as good as the random forest regression model's.

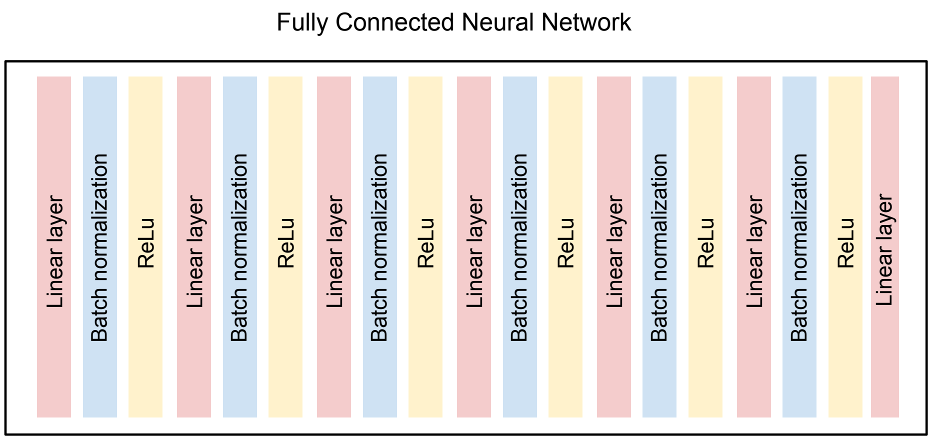


Figure 3.19: Fully Connected Neural Network

3.2.2 Convolutional Neural Network

Convolutional layers, pooling layers, and fully connected layers make up a convolutional neural network (CNN), which is often a feed-forward neural network.

Local connection, weight sharing, and pooling are the three structural features of convolutional neural networks. Convolutional neural networks exhibit some degree of translation, scaling, and rotation invariance as a result of these features. Convolutional neural networks contain fewer parameters as compared to feedforward neural networks.

CNN network structure

A convolutional neural network (CNN) is often a feed-forward neural network made up of fully connected, pooling, and convolutional layers. Three structural traits of convolutional neural networks are local connection, weight sharing, and pooling. Because of these features, convolutional neural networks exhibit some degree of translation, scaling, and rotation invariance. Convolutional neural networks have a smaller number of parameters than feedforward neural networks.

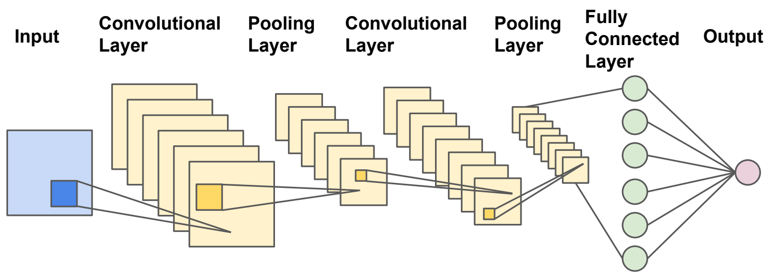


Figure 3.20: The composition of convolutional neural network

Convolution operation

Convolution is a powerful computer technology that is used by convolutional neural networks. One-dimensional convolution or two-dimensional convolution is frequently employed in signal processing or picture processing. It is important to transform one-dimensional data into a two-dimensional format since the picture is Convolution is enlarged. The two-dimensional convolution is defined as Y = W * X where the filter is W and the input data is X.

Feature map

Each feature map may be utilized as a class of extracted image features. A feature map is a feature that is obtained by the convolution of an image (or other feature maps). Multiple distinct feature maps can be employed at each layer to better reflect the features of the picture and enhance convolutional networks' ability to represent them.

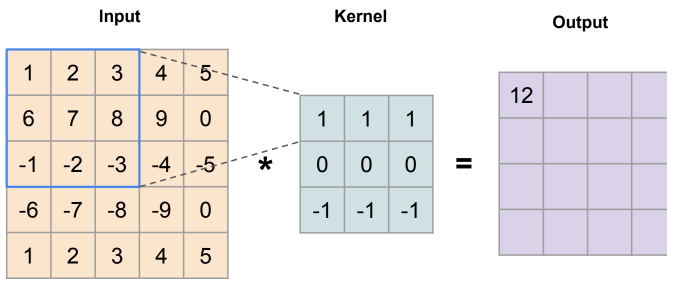


Figure 3.21: The working mechanism of a kernel

The feature map is the actual picture in the input layer. If it is a grayscale picture, there is a feature map; if it is a color image, there are feature maps of three RGB color channels; and if it is a grayscale image, there is a feature map; the depth of the input layer is D=1.

Convolutional layers

Local connectivity is the first characteristic. A locally connected network is created when each neuron in the convolutional layer, which is considered to be the Lth layer, is only linked to other neurons in a local window in the following layer, the L-1 layer. All of the neurons in the Lth layer share the same convolution kernel as a parameter. The sharing of weights is the second characteristic. Weight sharing may be thought of as a convolution kernel that only extracts one particular local feature from the input data, necessitating the usage of several distinct convolution kernels if you wish to extract other features.

The completely connected layer (a) in the fully connected neural network is seen in the image below. The completely linked layer's weight matrix has a large number of parameters, and training effectiveness will be low. The convolutional layer of the convolutional neural network is (b), as well. The weights are the same on connections of the same color in the convolutional layer, which significantly reduces the number of parameters in the weight matrix that may be produced in this fashion.

Pooling layer

The pooling layer's job is to do feature selection, cut down on the number of features, and subsequently cut down on the number of parameters.

The number of connections in the network can be greatly reduced by the convolutional layer, but the number of neurons in the feature map group is not much affected. If a classifier is added after, its input dimension is still quite high and it is simple to overfit. After the convolutional layer, a pooling layer can be added to minimize the feature dimension and prevent overfitting in order to address this issue.

Downsampling each region to obtain a result that represents this region as a whole is referred to as pooling.

There are two types of pooling functions that are frequently used: (1) Max Pooling, which chooses the maximal activity value of all the neurons in an area to serve as the region's representation; and (2) Average Pooling, also known as Mean Pooling.

In this part, we built a convolutional neural network, which contains seven convolutional layers, each of which is followed by batch normalization and ReLu. There are thirteen features in the airfare dataset, so we set the number of neurons in the input layer is thirteen. And the number of neurons in the hidden layer and output layer are 1024 and 1 respectively. We set the batch size as 128.

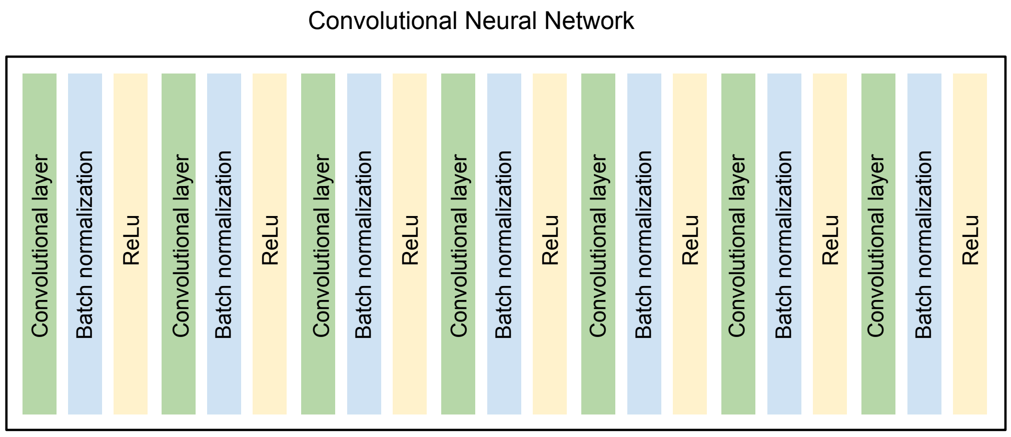


Figure 3.22: Convolutional Neural Network

3.2.3 Transformer

A Transformer is a model architecture that does not use recurrence and instead draws connections between input and output using an attention method. Prior to Transformers, the dominant sequence transduction models were based on complicated recurrent or convolutional neural networks with just an encoder and a decoder, as represented in Figure 3.23's left and right halves, respectively. The

Transformer similarly has an encoder and decoder, but by foregoing recurrence in favor of attention mechanisms, it may achieve substantially greater parallelization than RNNs and CNNs.

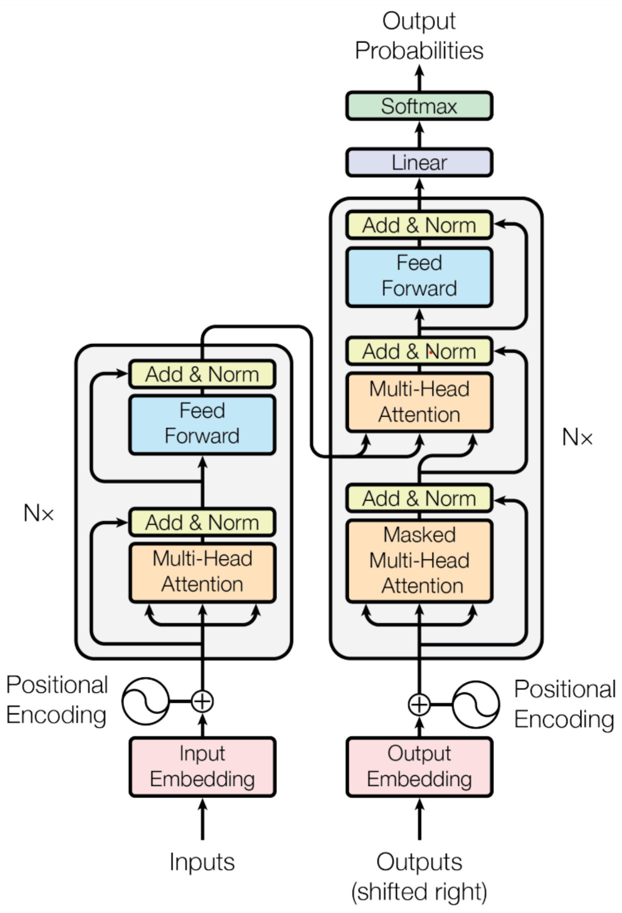


Figure 3.23: Architecture of a Transformer model.

Encoder
The encoder is made up of $N = 6$ identical layers. Each layer is divided into two sub-layers. The first is a multi-head self-attention mechanism, while the second is a basic, fully linked feed-forward network that is location-wise. Following layer normalization [29], we use a residual connection [30] around each of the two sub-layers. That is, the output of each sub-layer is $LayerNorm(x + Sublayer(x))$, where $Sublayer(x)$ is the sub-own layer's function. To assist these residual connections, all sub-layers and embedding layers in the model give outputs with dimension $d_model = 512$.

Decoder
The decoder is also made up of a stack of $N = 6$ identical layers. The decoder inserts a third sub-layer, which conducts multi-head attention over the encoder stack's output, in addition to the two sub-layers in each encoder layer. We use residual connections surrounding each sub-layer, similar to the encoder, followed by layer normalization. We additionally change the decoder stack's self-attention sub-layer to prevent positions from attending to the following positions. This masking, along with the fact that the output embeddings are offset by one place, ensures that predictions for position i can only rely on known outputs at locations less than i.

Attention
A query and a collection of key-value pairs are mapped to output by an attention function, where the query, keys, values, and output are all vectors. The result is generated as a weighted sum of the values, with the weight allocated to each value determined by the query's compatibility function with the relevant key.

Multi-Head Attention
Instead of executing a single attention function with d_{model} -dimensional keys, values, and queries, we discovered that linearly projecting the queries, keys, and values h times using distinct, learned linear projections to d_k, d_k and d_v dimensions was more advantageous. We then apply the attention function in parallel on each of these projected versions of queries, keys, and values, providing d_v-dimensional output values. These are concatenated and projected again, yielding the final values shown in Figure 3.24.

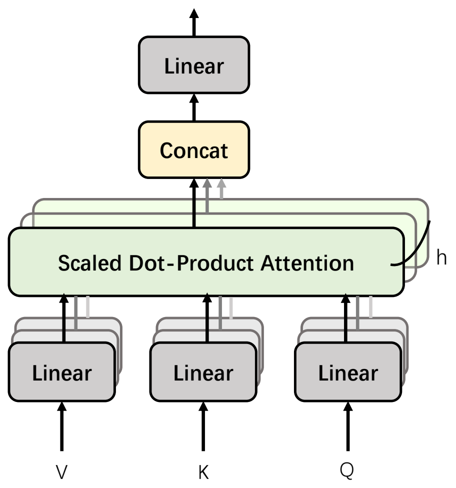


Figure 3.24: Multi-Head Attention

The model may attend to input from distinct representation subspaces at different points using multi-head attention. Averaging prevents this with a single attention head.

$$MultiHead(Q, K, V) = Concat\,(\,head_1, \ldots, head_{\,\mathrm{h}})\,W^O \tag{3.26}$$

$$\text{where head}_{\,i} = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$

In this thesis, we design a Transformer model which has 3-layer self-attention layers and set the number of heads as 4. Since our input data do not contain sequential information, we replicate it to a dimension of 16 to simulate series data. Next, we apply linear embedding to input data to enrich the features from 13 to 256. Then we feed these sequential high-dimensional features into the Transformer encoder, which is followed by a global average pooling to get the global features among the sequential features. Finally, we apply two fully connected layers, with the number of output nodes as 256 and 1, to get the final prediction.

3.2.4 Beyesian Neural Network

Bayesian neural networks are a popular type of neural network due to their ability to quantify the uncertainty in their predictive output.

With standard neural networks, the weights between the different layers of the network take single values. In a Bayesian neural network, the weights take on probability distributions. The process of finding these distributions is called marginalization. One important factor for training these networks is having a large enough set of training data to produce accurate probability distributions. This makes them more robust and allows them to generalize better with less overfitting.

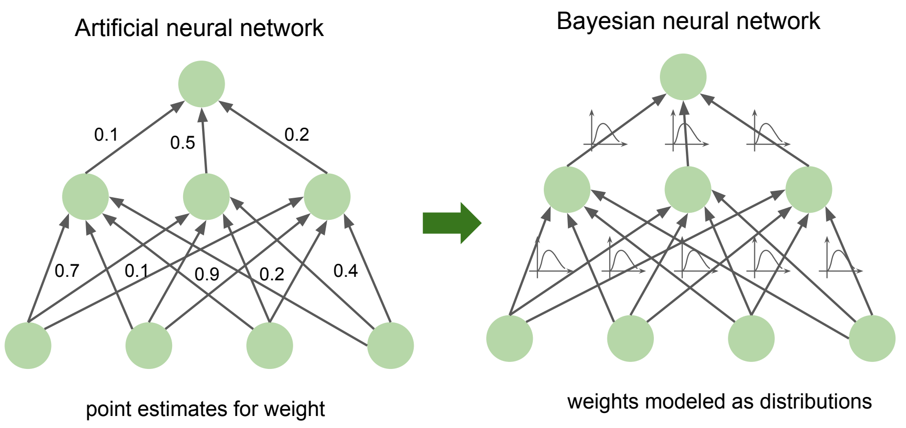

Artificial neural network
0.1
0.5
0.2
0.7
0.1
0.9
0.2
0.4
point estimates for weight
Bayesian neural network
weights modeled as distributions

Figure 3.25: Differences between Artificial neural network and Bayesian neural network

Variational Inference

We define $y = f(x)$ as a function that estimates the supplied inputs $\{x_1, \ldots, x_N\}$ and their related outputs $\{y_1, \ldots, y_N\}$ and gives a predicted output. A prior distribution is utilized across the space of functions $p(f)$ in Bayesian inference. This distribution reflects our prior assumption about which functions are most likely to have yielded our data.

To capture the process by which a given function observation is formed, a likelihood is defined as $p(Y \mid f, X)$. To calculate the posterior distribution given our dataset, we implement the Bayes rule: $p(Y \mid f, X)$. By integrating over all potential functions f, the new output for a new input point x^* may be anticipated.

$$p\left(y^* \mid x^*, X, Y\right) = \int p\left(y^* \mid f^*\right) p\left(f^* \mid x^*, X, Y\right) df^* \tag{3.27}$$

Because of the integration sign, equation 3.27 is unsolvable. We may approximate it by conditioning the model on a finite collection of random variables w. However, it is predicated on a modeling assumption that the model is dependent solely on these variables, and we use them as statistics in our approximation model. After that, the predictive distribution for a new input point x^* is given by:

$$p\left(y^* \mid x^*, X, Y\right) = \int p\left(y^* \mid f^*\right) p\left(f^* \mid x^*, w\right) p(w \mid X, Y) df^* dw \tag{3.28}$$

However, the distribution $p(w \mid X, Y)$ remains difficult, and we must approximate it using a calculated variational distribution $q(w)$. The estimated distribution should be as similar to the posterior distribution generated from the original model as feasible. As a result, we minimize the Kullback-Leibler (KL) divergence, which is an intuitive measure of similarity between two distributions: The approximate predictive distribution is obtained by $KL(q(w) \| p(w \mid X, Y))$.

$$q\left(y^{*} \mid x^{*}\right)=\int p\left(y^{*} \mid f^{*}\right) p\left(f^{*} \mid x^{*}, w\right) q(w) d f^{*} d w \tag{3.29}$$

Minimizing the Kullback-Leibler divergence corresponds to increasing the *log evidence lower bound.*

$$KL_{\mathrm{VI}}:=\int q(w) p(F \mid X, w) \log p(Y \mid F) d F d w-KL(q(w) \| p(w)) \tag{3.30}$$

with relation to the variational parameters that define $q(w)$. This is referred to as *variational inference*, and it is a standard strategy in Bayesian modeling.

Maximizing the KL divergence between the posterior and the prior over w produces a variational distribution that learns a good representation from the data (as determined by log-likelihood) and is closer to the prior distribution. In other words, it can help to avoid overfitting.

Local Reparametrisation Trick

One of the most general-purpose tools in mathematical statistics is the capacity to rephrase statistical problems in an equivalent but different form, or to reparameterize them. The type of *reparameterization* when the global uncertainty in the weights is transformed into a form of local uncertainty which is independent across cases is known as the *local reparameterization trick.* An alternate estimator with $\operatorname{Cov}\left[L_{i}, L_{j}\right]=0$ is used, such that the variance of the stochastic gradients scales as $1/M$. The new estimator is computationally efficient since it samples the intermediate variables rather than ϵ directly, but just $f(\epsilon)$, which affects $L_{\mathcal{D}}^{\mathrm{SGVB}}(\phi)$. As a result of translating the source of global noise to local noise ($\epsilon \rightarrow f(\epsilon)$), a local reparameterization may be used to generate a statistically efficient gradient estimator.

A simple example will help to understand the technique: We assume an input(X) of a random uniform function with a range of -1 to +1 and an output(Y) of a random normal distribution with mean X and standard deviation δ. The Mean Squared Loss is defined as $(Y-X)^{2}$. The issue arises during the backpropagation of the random normal distribution function. We reparameterize as we try to propagate across a stochastic node by adding X to the random normal function output and multiplying by δ. The model's behavior is unaffected by moving parameters outside the normal distribution.

There are several benefits of adopting Bayesian neural networks.
(1) They are more resilient and generalizable than other neural networks.
(2) They can quantify the uncertainty in their predicted output.
(3) They may be utilized for a wide range of practical applications.

There are certain drawbacks to utilizing Bayesian neural networks, which we will now go through.
(1) They can be more difficult to train than other neural networks and need an understanding of probability and statistics.
(2) They can be slower to converge than other neural networks and frequently require more data. Because the network's weights are distributions rather than single values, more data is necessary to correctly predict the weights.

In this part, we built a bayesian fully connected neural network, which contains two bayesian layers and five linear layers, each of which is followed by batch normalization and ReLu. There are thirteen features in the airfare dataset, so we set the number of neurons in the input layer is thirteen. And the number of neurons in the hidden layer and output layer are 1024 and 1 respectively. We set the batch size as 128.

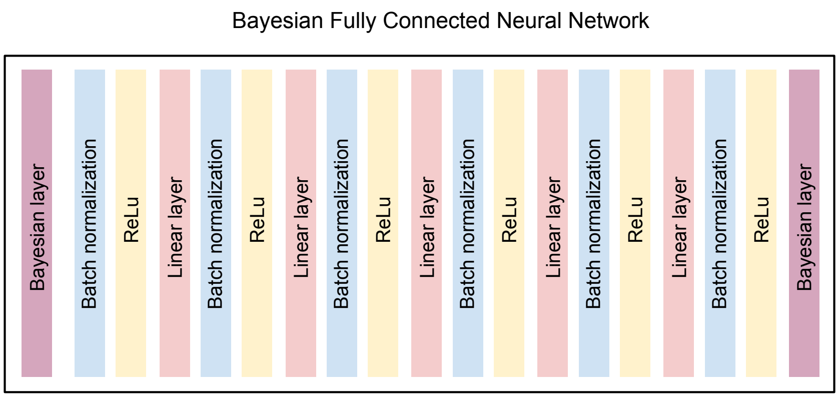


Figure 3.26: Bayesian Fully Connected Neural Network

Meanwhile, we built a Bayesian Convolutional Neural Network, which contains one bayesian layer and six convolutional layers, each of which is followed by batch normalization and ReLu. There are thirteen features in the airfare dataset, so we set the number of neurons in the input layer is thirteen. And the number of neurons in the hidden layer and output layer are 1024 and 1 respectively. We set the batch size to 128.

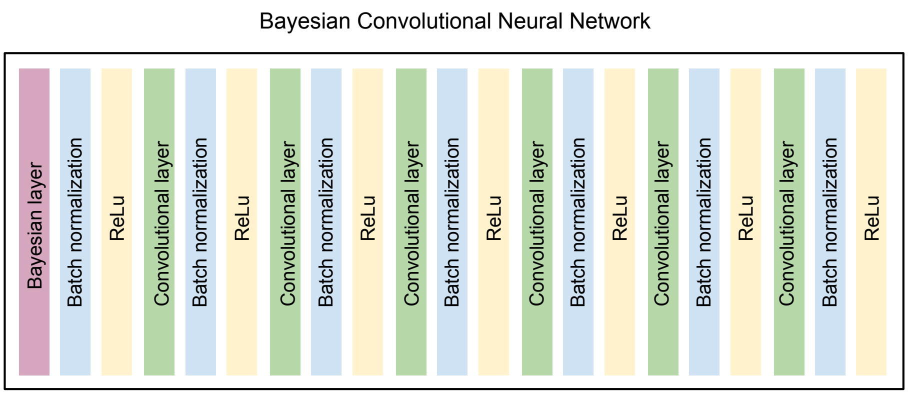


Figure 3.27: Bayesian Convolutional Neural Network

Chapter 4

Experiments

This chapter first introduces the experimental data set and preprocesses the data in a standardized manner. Subsequently, the factors that have an influence on future airline ticket prices are explored in terms of their own price series and external influence factors respectively, and the important characteristics of the random forest regressor model, KNN regressor model, ridge regressor model are used to screen the variables and provide good data support for the subsequent research At the same time, a neural network model was constructed to accomplish the task of predicting airline ticket prices.

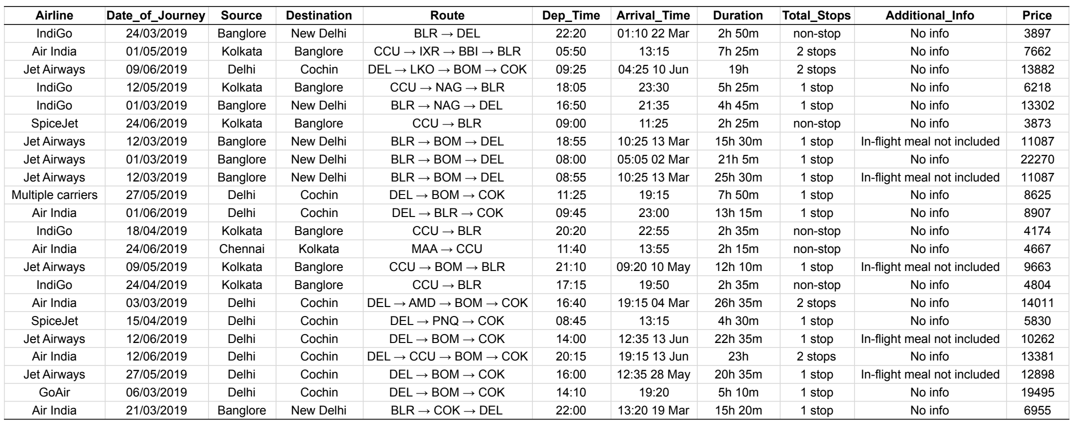
4.1 Dataset

4.1.1 Flight Price Dataset

The dataset used in this paper is from Kaggle, which contains a total of 10,683 routes between these cities within India: New Delhi, Bangalore, Cochin, Kolkata, Hyderabad, and Delhi from March 2019 to June 2019, and from this data, each raw data contains 11 fields of information, as shown in the chart.

The various features of the cleaned dataset are explained below:
(1) Airline: The name of the airline company is stored in the airline column. It is a categorical feature having 6 different airlines.
(2) Flight: The flight stores information regarding the plane's flight code. It is a categorical feature.
(3) Source City: City from which the flight takes off. It is a categorical feature having 6 unique cities.
(4) Departure Time: This is a derived categorical feature obtained created by grouping time periods into bins. It stores information about the departure time

Table 4.1: An example of data information in our adopted flight ticket dataset.



Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
Air India	01/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662
Jet Airways	09/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882
IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218
IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302
SpiceJet	24/06/2019	Kolkata	Banglore	CCU → BLR	09:00	11:25	2h 25m	non-stop	No info	3873
Jet Airways	12/03/2019	Banglore	New Delhi	BLR → BOM → DEL	18:55	10:25 13 Mar	15h 30m	1 stop	In-flight meal not included	11087
Jet Airways	01/03/2019	Banglore	New Delhi	BLR → BOM → DEL	08:00	05:05 02 Mar	21h 5m	1 stop	No info	22270
Jet Airways	12/03/2019	Banglore	New Delhi	BLR → BOM → DEL	08:55	10:25 13 Mar	25h 30m	1 stop	In-flight meal not included	11087
Multiple carriers	27/05/2019	Delhi	Cochin	DEL → BOM → COK	11:25	19:15	7h 50m	1 stop	No info	8625
Air India	01/06/2019	Delhi	Cochin	DEL → BLR → COK	09:45	23:00	13h 15m	1 stop	No info	8907
IndiGo	18/04/2019	Kolkata	Banglore	CCU → BLR	20:20	22:55	2h 35m	non-stop	No info	4174
Air India	24/06/2019	Chennai	Kolkata	MAA → CCU	11:40	13:55	2h 15m	non-stop	No info	4667
Jet Airways	09/05/2019	Kolkata	Banglore	CCU → BOM → BLR	21:10	09:20 10 May	12h 10m	1 stop	In-flight meal not included	9663
IndiGo	24/04/2019	Kolkata	Banglore	CCU → BLR	17:15	19:50	2h 35m	non-stop	No info	4804
Air India	03/03/2019	Delhi	Cochin	DEL → AMD → BOM → COK	16:40	19:15 04 Mar	26h 35m	2 stops	No info	14011
SpiceJet	15/04/2019	Delhi	Cochin	DEL → PNQ → COK	08:45	13:15	4h 30m	1 stop	No info	5830
Jet Airways	12/06/2019	Delhi	Cochin	DEL → BOM → COK	14:00	12:35 13 Jun	22h 35m	1 stop	In-flight meal not included	10262
Air India	12/06/2019	Delhi	Cochin	DEL → CCU → BOM → COK	20:15	19:15 13 Jun	23h	2 stops	No info	13381
Jet Airways	27/05/2019	Delhi	Cochin	DEL → BOM → COK	16:00	12:35 28 May	20h 35m	1 stop	In-flight meal not included	12898
GoAir	06/03/2019	Delhi	Cochin	DEL → BOM → COK	14:10	19:20	5h 10m	1 stop	No info	19495
Air India	21/03/2019	Banglore	New Delhi	BLR → COK → DEL	22:00	13:20 19 Mar	15h 20m	1 stop	No info	6955

and has 6 unique time labels.
(5) Stops: A categorical feature with 3 distinct values that stores the number of stops between the source and destination cities.
(6) Arrival Time: This is a derived categorical feature created by grouping time intervals into bins. It has six distinct time labels and keeps the information about the arrival time.
(7) Destination City: City where the flight will land. It is a categorical feature having 6 unique cities.
(8) Class: A categorical feature that contains information on seat class; it has two distinct values: Business and Economy.
(9) Duration: A continuous feature that displays the overall amount of time it takes to travel between cities in hours.
(10) Days Left: This is a derived characteristic that is calculated by subtracting the trip date from the booking date.
(11) Price: The target variable stores information on the ticket price.

4.1.2 Preprocessing

Missing data imputation: To facilitate the study, the terms are further treated in this paper. In the real world, it is very common for data to contain missing values because some information is not available or the data is not recorded, omitted, or lost due to human factors. However, when training a dataset with many missing values, the presence of missing values can greatly affect the performance of the machine learning model.

Table 4.2: An example of the data information after preprocessing.

Airline	Source	Destination	Route	Additional_Info	Duration	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Price
3	0	5	18	3	170	0	24	3	22	20	1	10	3897
1	3	0	84	3	445	2	1	5	5	50	13	15	7662
4	2	1	118	3	1140	2	9	6	9	25	4	25	13882
3	3	0	91	3	325	1	12	5	18	5	23	30	6218
3	0	5	29	3	285	1	1	3	16	50	21	35	13302
7	3	0	64	3	145	0	24	6	9	0	11	25	3873
4	0	5	5	1	930	1	12	3	18	55	10	25	11087
4	0	5	5	3	1265	1	1	3	8	0	5	5	22270
4	0	5	5	1	1530	1	12	3	8	55	10	25	11087
5	2	1	104	3	470	1	27	5	11	25	19	15	8625
1	2	1	103	3	795	1	1	6	9	45	23	0	8907
3	3	0	64	3	155	0	18	4	20	20	22	55	4174
1	1	4	127	3	135	0	24	6	11	40	13	55	4667
4	3	0	66	1	730	1	9	5	21	10	9	20	9663
3	3	0	64	3	155	0	24	4	17	15	19	50	4804
1	2	1	97	3	1595	2	3	3	16	40	19	15	14011
7	2	1	123	3	270	1	15	4	8	45	13	15	5830
4	2	1	104	1	1355	1	12	6	14	0	12	35	10262
1	2	1	105	3	1380	2	12	6	20	15	19	15	13381
4	2	1	104	1	1235	1	27	5	16	0	12	35	12898
2	2	1	104	3	310	1	6	3	14	10	19	20	19495
1	0	5	17	3	920	1	21	3	22	0	13	20	6955

Unit conversion: In order to handle the data well, we convert flight duration hours into minutes; split the date of the journey into journey day and journey month; split the departure time and arrival time into hours and minutes.

Then, in the analysis of additional information, we find that the percentage of "Change airports", "Business class", "2 Long layover", "Red-eye flight", and "1 Short layover" are few, and we call them collectively as " other".

Categorical data encoding. Finally, in order to use the machine learning model, we first convert categorical data into numerical. now that the data preparation work is done, we will build machine learning models.

4.2 Evaluation Metrics

4.2.1 Root mean square error

One of the methods most frequently used to assess the accuracy of forecasts is the root mean square error (RMSE). It illustrates the Euclidean distance between measured true values and forecasts. For each data point, determine the residual (difference between forecast and reality), its norm, and its standard deviation to calculate the residual mean square error (RMSE). Due to the fact that it requires and utilizes real measurements at each projected data point, RMSE is frequently utilized in supervised learning applications.

The standard deviation is a gauge of how evenly distributed a set of numbers is. The square root of the variance serves as its formula. The average of the squared deviations from the mean is known as a variance. In the standard deviation formula below, "x_i" stands for the number, "μ" for the number's average, and "N" stands for the total number of values.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{4.1}$$

Data points' distance from the regression line is gauged by residuals. By deducting the anticipated value from the actual value, we may calculate residuals, which are nothing more than prediction errors.

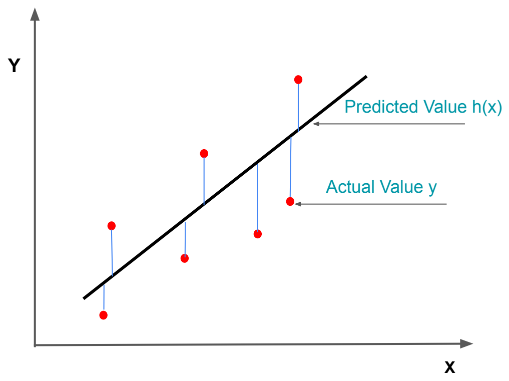


Figure 4.1: The geometrical meaning of residuals.

So instead of using the square root of the variance to get the RMSE, we will use the square root of the average of the squared residuals. Calculate the residual (difference between forecast and truth) for each data point, together with its norm, mean, and square root in order to determine the root-mean-square error (RMSE). Due to the fact that it requires and utilizes real measurements at each projected data point, RMSE is frequently utilized in supervised learning applications. A number of 0 (nearly never attained in practice) would represent a perfect fit to the data, and RMSE is always non-negative. A smaller RMSE is often preferable to a greater one. However, because the measure depends on the size of the numbers used, comparisons across other types of data would be incorrect. The average of the squared errors' square root is the RMSE. Each error has an impact on RMSE that is proportional to the amount of the squared error; as a result, greater errors have an outsized influence on RMSE. RMSE is hence vulnerable to outliers. [31][32]

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2} \tag{4.2}$$

When evaluating a model's performance in machine learning, whether during training, cross-validation, or monitoring after deployment, it is very beneficial to have a single number. One of the most used metrics for this is the root mean square error. It is an appropriate scoring method that is simple to comprehend and consistent with some of the most widely used statistical presumptions.

4.2.2 Mean absolute error

A model assessment statistic used with regression models is mean absolute error. The average of the absolute values of each prediction error over all test set occurrences is the mean absolute error of a model with respect to the test set. The difference between the instance's real value and the expected value represents each prediction mistake.

$$\text{mae} = \frac{\sum_{i=1}^{n} abs\left(y_i - \lambda\left(x_i\right)\right)}{n} \tag{4.3}$$

where y_i is the true target value for test instance y_i , $\lambda(x_i)$ is the predicted target value for test instance x_i , and n is the number of test instances.

The smaller the MAE, the better your model is in performing your (regression) analysis. A model is perfect if its mean absolute error is zero. In other words, the actual and anticipated values are identical for all observations.

The MAE has an advantage over other metrics in that its result is in the same units as the relevant variable. So, if your mean absolute error is 2, the absolute difference between the actual value and the anticipated value is 2 units.

The MAE's potential drawback is that both low and high mistakes are given equal weight. In other words, a difference of 1 unit is just as significant as a difference of 5 units between the actual and anticipated values. Use the Root Mean Squared Error (RMSE) if larger errors should be punished more severely than smaller ones.

4.2.3 Mean absolute percentage error

The mean absolute percentage error (MAPE), also known as mean absolute percentage deviation (MAPD), is a measure of predicting accuracy used in statistics, such as trend estimation, and is also used as a Loss function in Machine Learning. The MAPE (Mean Absolute Percent Error) calculates the error magnitude in percentage terms. It is determined as the average of the unsigned percentage errors, as seen in the following example:

$$\text{MAPE } = \frac{100}{n} \sum_{t=1}^{n} \frac{|A_t - F_t|}{A_t} \%, \tag{4.4}$$

where A_t are actuals and F_t corresponding forecasts or predictions.

It reflects the average of each entry's absolute percentage inaccuracy, indicating how accurate the anticipated amounts were in relation to the actual values. MAPE is sometimes useful for studying bigger collections of data, although it is impossible to calculate the MAPE of datasets with zero values. This is due to the computation requiring division by zero, which is not feasible. MAPE is a simple statistic that reflects the average deviation between anticipated and actual values as 10%, regardless of whether the deviation was positive or negative. There is, however, no industry standard for what constitutes a good MAPE.

4.2.4 Coefficient of determination R^2

The data points' dispersion around the fitted regression line is measured using R-squared. Multiple regression is also known as the coefficient of determination or the coefficient of multiple determination. Higher R-squared values for the same data set indicate less discrepancy between the fitted values and the observed data.

The amount of variance in the dependent variable that a linear model can explain is measured by its R-squared.

$$R^2 = \frac{\text{Variance explained by the model}}{\text{Total variance}} \tag{4.5}$$

R-squared ranges from 0% to 100% always:
(1) A model with a 0% explanatory power does not account for any variation in the response variable around its mean. Both the dependent variable and the regression model are predicted by the dependent variable's mean.
(2) 100% denotes a model that accounts for all of the response variable's variation around its mean.

The better the regression model matches your observations, typically, the bigger the R2. Plotting the fitted values against the observed values will show how R-squared values indicate the dispersion around the regression line.

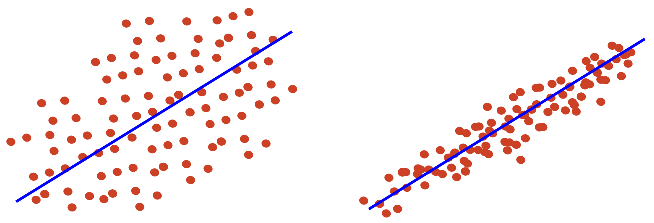


Figure 4.2: An illustration of the coefficient of determination R^2.

The regression model on the left has an R-squared of 15%, whereas the model on the right has an R-squared of 85%. The data points are nearer the regression line when a regression model explains more variation. An R2 of 100% is unheard of in regression models used in real-world applications. The fitted values are then equal to the observed values, and all of the observations lie precisely on the regression line.

4.3 Implementation Details

We implement a total of 12 methods for a thorough comparison. Specifically, we include 7 traditional machine learning regression algorithms (Lasso Regression, Ridge Regression, Support Vector Regression, K-Nearest Neighbors, XGBoost, Decision Tree, Random Forest) and 5 deep neural network methods (Transformer, Fully Connected Network, Bayesian Fully Connected Network, Convolutional Neural Network, Bayesian Convolutional Neural Network). To keep a fair comparison, we split the dataset into 70% training data and 30% test data. We use the same training and test data for all the implemented methods. We will introduce the implementation details of these two kinds of methods respectively.

Machine learning methods: The traditional machine learning regression methods are implemented based on the framework of scikit-learn. We train our model on normal CPUs and the training takes only several minutes for all the methods. The settings of hyper-parameters are given in last chapter.

Deep Learning methods: The deep learning methods are implemented based on the framework of Pytorch. To optimize the parameters of our network, we adopt the Adam solver with batch size 128. We set the learning rate as 0.01. We train our model on one NVIDIA GeForce GTX 3090 GPU. It takes about half an hour to get the Convolutional Neural Network model and the Fully Connected Network model. It takes about one hour to get the Bayesian model and Transformer model. The setting of hyper-parameters is introduced below. For Bayesian neural networks, we set the number of ensembles during training as 5, the weight of KL loss as 2×10^{-5}, and the number of ensembles during testing as 128.

4.4 Experimental Results

4.4.1 Comparison of different methods

Table 4.3 shows the numerical results of mean squared error(RMSE), mean absolute error (MAE), mean absolute percentage error (MAPE), and coefficient of determination R^2 among different methods. Specifically, we implement representative traditional machine learning methods (Lasso Regression, Ridge Regression, Support Vector Regression, K-Nearest Neighbors, XGBoost, Decision Tree, Random Forest) and deep neural networks (Transformer, Fully Connected Network, Bayesian Fully Connected Network, Convolutional Neural Network, Bayesian Convolutional Neural Network).

Table 4.3: Comparison of different methods on RMSE, MAE, MAPE, and R^2. The best two results are highlighted in red and blue.

Methods	RMSE (↓)	MAE (↓)	MAPE (↓)	R^2 (↑)
Lasso Regression	3299.82	2396.99	31.96%	0.466
Ridge Regression	3289.24	2407.41	32.02%	0.470
Support Vector Regression	3167.65	1913.42	20.70%	0.508
K-Nearest Neighbors	2902.66	1690.64	18.34%	0.587
XGBoost	1567.57	749.70	8.76%	0.880
Decision Tree	1937.03	704.95	8.10%	0.816
Random Forest	1566.37	645.78	7.57%	0.880
Transformer	1733.29	835.79	10.18%	0.853
Fully Connected Network	1564.42	710.15	8.15%	0.880
Bayesian FCN (ours)	1491.36	698.53	8.04%	0.891
Convolutional Neural Network	1486.25	756.20	8.82%	0.892
Bayesian CNN (ours)	1414.51	749.70	8.81%	0.902

It can be observed that for traditional machine learning methods, Decision Tree, XGBoost, and Random Forest achieve significantly better performance than Lasso Regression, Ridge Regression, Support Vector Regression, and KNN. Besides, Random Forest achieves the best performance in all metrics among all traditional machine learning methods.

For the deep learning-based methods, the performance of Fully Connected Network, Bayesian FCN, Convolutional Neural Network, and Bayesian CNN achieve better performance than all traditional methods in RMSE and R^2. We must highlight that with our proposed Bayesian layers, the performance of CNN and FCN can be both improved. Although Transformer is a recently popular method in various fields, the performance is the worst among all deep learning methods, which means it is not a ideal candidate for such an airfare prediction task. Besides, it should be noticed that Random Forest, as a traditional method, still achieves the best performance in MAE and MAPE among all the methods, which means the traditional machine learning methods is still playing an important role in our task.

4.4.2 Ablation studies

In order to investigate how much each input feature can affect the final prediction effectively and efficiently, we do ablation studies by removing each input feature respectively. We give the RMSE and MAE results of Random Forest and Convolutional Neural Networks.

From the data in the table below, we can see that after removing some features, the regression performance becomes better, such as "Route" and "Duration". However, for other features, the performance after deleting them is not as good as retaining all features. In this paper, we consider the case of retaining all features.

4.4.3 Running time comparisons

We also give comparisons of running time for different methods. Since the traditional methods and deep learning methods are run on CPU and GPU separately, we also give the results separately.

Table 4.5 shows the running times of different machine learning methods. From the data in the table, we can see that the Decision Tree Regression Model runs the fastest, and the Random Forest Regression Model takes the longest time.

Table 4.6 shows the running time of different deep learning methods. From the data in the table, we can see that the Fully Connected Network runs the fastest and has the least number of parameters; the Bayesian Convolutional Neural Network takes the longest time and has the largest number of parameters.

Table 4.4: Ablation studies on input features. We give the RMSE and MAE results of Random Forest and Convolutional Neural Network.

Removed features	Random Forest		CNN	
	RMSE (↓)	MAE (↓)	RMSE (↓)	MAE (↓)
Airline	1808.48	817.58	1865.21	908.81
Source	1563.90	647.16	1504.47	801.58
Destination	1567.20	670.60	1512.64	737.11
Route	1567.20	670.60	1465.86	734.33
Additional info	1934.81	1106.53	2076.99	1290.78
Duration	1330.44	609.59	1427.72	744.00
Total stops	1565.36	665.47	1722.59	817.62
Journey day	2350.63	1095.24	2270.98	1164.65
Journey month	1859.17	1011.80	1854.76	1083.05
Departure hour	1542.54	654.78	1509.71	774.45
Departure minutes	1588.30	676.78	1512.44	772.68
Arrival hour	1579.16	649.57	1479.75	823.71
Departure minutes	1558.20	651.00	1576.35	770.87
All features included	1566.37	645.78	1486.25	756.20

Table 4.5: The running time comparison of different machine learning methods.

Methods	Time Per Sample [ms]
Lasso Regression	0.10
Ridge Regression	0.10
Support Vector Regression	0.47
K-Nearest Neighbors	0.28
XGBoost	0.94
Decision Tree	0.04
Random Forest	4.56

Table 4.6: The comparisons of running time, MACs, and number of parameters of different deep learning methods.

Methods	Time Per Sample [ms]	#Parameters
Transformer	1.22	26.3
Fully Connected Network	0.45	5.28
Bayesian FCN	0.83	5.29
Convolutional Neural Network	1.10	21.0M
Bayesian CNN	11.03	23.1M

Chapter 5

Conclusions

In this thesis, we did a systematic comparison of traditional machine learning methods (e.g., Ridge Regression, Lasso Regression, K-Nearest Neighbor, XG-Boost, Decision Tree, and Random Forest) and deep learning methods (e.g., Fully Connected Networks, Convolutional Neural Networks) on the problem of airfare prediction. We proposed a Bayesian neural network for airfare prediction, which is the first method that utilizes Bayesian Inference for the airfare prediction task. We evaluate the performance of different methods on an open dataset of 10,683 domestic routes in India from March 2019 to June 2019. The experimental results show that deep learning-based methods achieve better results than traditional methods in RMSE and R^2, while Bayesian neural networks can further improve the performance.

This thesis can be further extended for future work. First, the adopted public dataset only contains limited data, which hindered the performance of neural networks. It will be more worthwhile to collect a larger and wider dataset to explore the potential of deep neural networks. Second, it will be interesting to utilize the time-series information to make a better prediction. Third, it is promising to design a special network to better capture useful features and information from the given data.

Bibliography

[1] Tianyi Wang, Samira Pouyanfar, Haiman Tian, Yudong Tao, Miguel Alonso, Steven Luis, and Shu-Ching Chen. «A framework for airfare price prediction: a machine learning approach». In: *2019 IEEE 20th international conference on information reuse and integration for data science (IRI)*. IEEE. 2019, pp. 200–207 (cit. on pp. 4, 6).

[2] Megan S Ryerson and Hyun Kim. «Integrating airline operational practices into passenger airline hub definition». In: *Journal of Transport Geography* 31 (2013), pp. 84–93 (cit. on p. 4).

[3] Hojong Baik, Antonio A Trani, Nicolas Hinze, Howard Swingle, Senanu Ashiabor, and Anand Seshadri. «Forecasting model for air taxi, commercial airline, and automobile demand in the United States». In: *Transportation Research Record* 2052.1 (2008), pp. 9–20 (cit. on p. 4).

[4] William Groves and Maria Gini. «Optimal airline ticket purchasing using automated user-guided feature selection». In: *Twenty-Third International Joint Conference on Artificial Intelligence*. 2013 (cit. on p. 4).

[5] Zhenbang Wang. «RWA: A Regression-based Scheme for Flight Price Prediction». In: (2020) (cit. on p. 4).

[6] Kristopher S Gerardi and Adam Hale Shapiro. «Does competition reduce price dispersion? New evidence from the airline industry». In: *Journal of Political Economy* 117.1 (2009), pp. 1–37 (cit. on p. 5).

[7] Tanisha Patel et al. «FLIGHT FARE PREDICTION». In: (2021) (cit. on p. 5).

[8] Manolis Papadakis. «Predicting Airfare Prices». In: (2014) (cit. on p. 6).

[9] Jun Lu. «Machine learning modeling for time series problem: Predicting flight ticket prices». In: *arXiv preprint arXiv:1705.07205* (2017) (cit. on p. 6).

[10] Oren Etzioni, Rattapoom Tuchinda, Craig A Knoblock, and Alexander Yates. «To buy or not to buy: mining airfare data to minimize ticket purchase price». In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2003, pp. 119–128 (cit. on p. 6).

[11] Qiqi Ren. «When to Book: Predicting Flight Pricing». In: *Standford university* () (cit. on p. 6).

[12] Selim BUYRUKOĞLU and Yıldıran YILMAZ. «An Approach for Airfare Prices Analysis with Penalized Regression Methods». In: *Veri Bilimi* 4.2 (), pp. 57–61 (cit. on p. 6).

[13] Ruixuan Ren, Yunzhe Yang, and Shenli Yuan. «Prediction of airline ticket price». In: *University of Stanford* (2014) (cit. on p. 6).

[14] Tim Janssen, T Dijkstra, Saiden Abbas, and AC van Riel. «A linear quantile mixed regression model for prediction of airline ticket prices». In: *Radboud University* (2014) (cit. on p. 6).

[15] Till Wohlfarth, Stéphan Clémençon, François Roueff, and Xavier Casellato. «A data-mining approach to travel price forecasting». In: *2011 10th International Conference on Machine Learning and Applications and Workshops*. Vol. 1. IEEE. 2011, pp. 84–89 (cit. on p. 7).

[16] Han-Chen Huang. «A hybrid neural network prediction model of air ticket sales». In: *Telkomnika Indonesian Journal of Electrical Engineering* 11.11 (2013), pp. 6413–6419 (cit. on p. 7).

[17] Konstantinos Tziridis, Th Kalampokas, George A Papakostas, and Kostas I Diamantaras. «Airfare prices prediction using machine learning techniques». In: *2017 25th European Signal Processing Conference (EUSIPCO)*. IEEE. 2017, pp. 1036–1039 (cit. on p. 7).

[18] Everton Santana, Saulo Mastelini, et al. «Deep regressor stacking for air ticket prices prediction». In: *Anais do XIII simpósio brasileiro de sistemas de informação*. SBC. 2017, pp. 25–31 (cit. on p. 7).

[19] Stacey Mumbower and Laurie A Garrow. «Data Set—Online Pricing Data for Multiple US Carriers». In: *Manufacturing & Service Operations Management* 16.2 (2014), pp. 198–203 (cit. on p. 7).

[20] Mian Dai, Qihong Liu, and Konstantinos Serfes. «Is the effect of competition on price dispersion nonmonotonic? Evidence from the US airline industry». In: *Review of Economics and Statistics* 96.1 (2014), pp. 161–170 (cit. on p. 8).

[21] Krishna Rama-Murthy. «Modeling of United States Airline Fares–Using the Official Airline Guide (OAG) and Airline Origin and Destination Survey (DB1B)». PhD thesis. Virginia Tech, 2006 (cit. on p. 8).

[22] Evelyn Fix and Joseph Lawson Hodges. «Discriminatory analysis. Nonparametric discrimination: Consistency properties». In: *International Statistical Review/Revue Internationale de Statistique* 57.3 (1989), pp. 238–247 (cit. on p. 15).

[23] Naomi S Altman. «An introduction to kernel and nearest-neighbor nonparametric regression». In: *The American Statistician* 46.3 (1992), pp. 175–185 (cit. on p. 15).

[24] S Madeh Piryonesi and Tamer E El-Diraby. «Role of data analytics in infrastructure asset management: Overcoming data size and quality problems». In: *Journal of Transportation Engineering, Part B: Pavements* 146.2 (2020), p. 04020022 (cit. on p. 16).

[25] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction.* Vol. 2. Springer, 2009 (cit. on p. 16).

[26] Xin Yao. «Evolving artificial neural networks». In: *Proceedings of the IEEE* 87.9 (1999), pp. 1423–1447 (cit. on p. 40).

[27] Shifei Ding, Chunyang Su, and Junzhao Yu. «An optimizing BP neural network algorithm based on genetic algorithm». In: *Artificial intelligence review* 36.2 (2011), pp. 153–162 (cit. on p. 40).

[28] Zhi Xiao, Shi-Jie Ye, Bo Zhong, and Cai-Xin Sun. «BP neural network with rough set for short term load forecasting». In: *Expert Systems with Applications* 36.1 (2009), pp. 273–279 (cit. on p. 40).

[29] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. «Layer normalization». In: *arXiv preprint arXiv:1607.06450* (2016) (cit. on p. 46).

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Deep residual learning for image recognition». In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016, pp. 770–778 (cit. on p. 46).

[31] Robert Gilmore Pontius, Olufunmilayo Thontteh, and Hao Chen. «Components of information for multiple resolution comparison between maps that share a real variable». In: *Environmental and ecological statistics* 15.2 (2008), pp. 111–142 (cit. on p. 56).

[32] Cort J Willmott and Kenji Matsuura. «On the use of dimensioned measures of error to evaluate the performance of spatial interpolators». In: *International Journal of Geographical Information Science* 20.1 (2006), pp. 89–102 (cit. on p. 56).