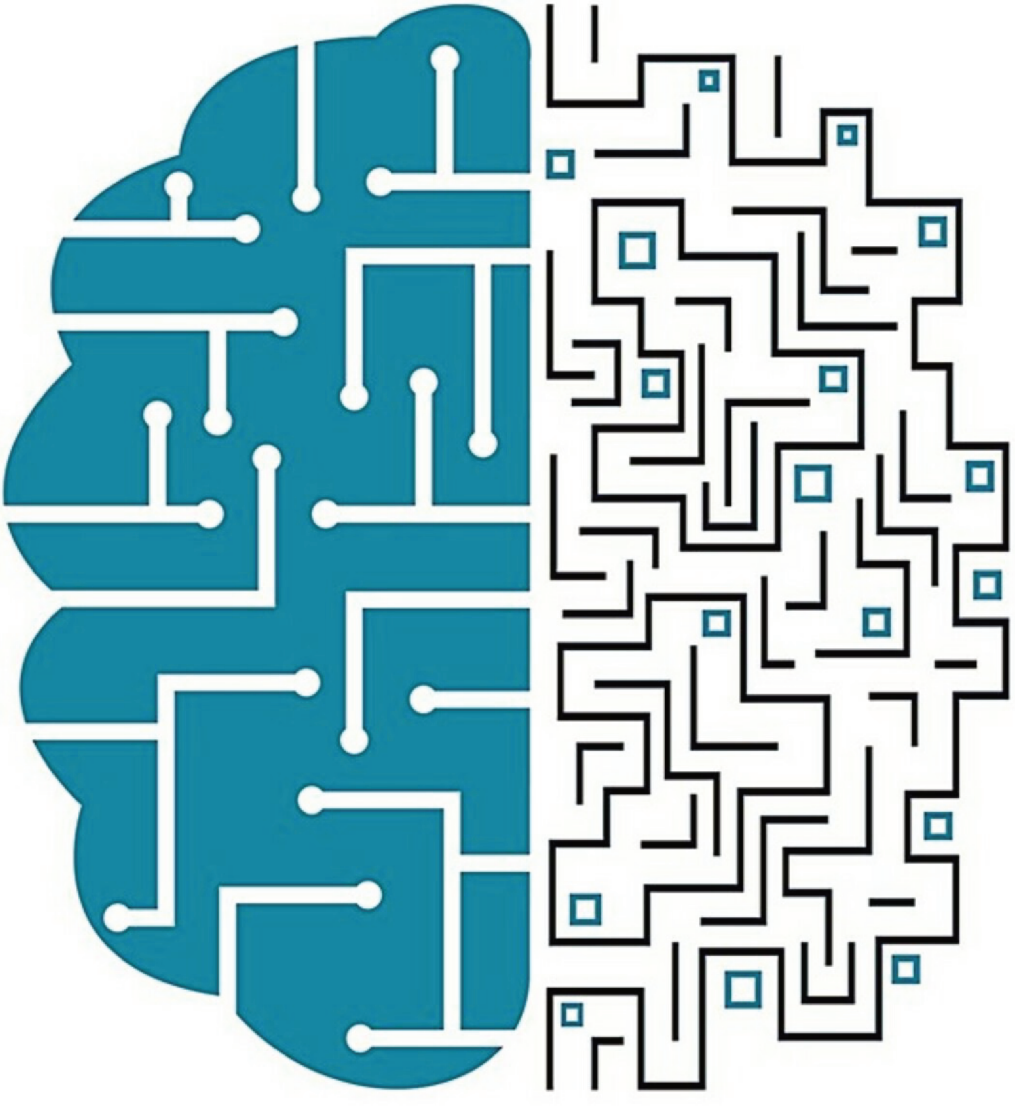
Feature Engineering & Selection for Explainable Models

A second course for data scientists

Md Azimul Haque



Feature Engineering & Selection for Explainable Models

A Second Course for Data Scientists

Md Azimul Haque

Contents

Foreword

I have been teaching data science as a part-time faculty and have taught hundreds of students. Many of my students have successfully transitioned to a data science career. I also have a set of open-source contributions as python libraries. I often get requests from my previous students when they start working on their first real machine learning project. I found the root cause of many challenges faced by my students who recently transitioned into data science and machine learning. I have tried to address these issues in my book and would like to dedicate this book to all my students for all the love and respect I have received.

There are many methods for feature engineering, feature selection and signal processing which were not available previously in python. I have coded and open-sourced these methods as python libraries over the years. Often, I get approached by researchers who use my python libraries, for understanding how to use certain algorithms and best practices for these algorithms. I have tried to explain these with the aid of underlying theory, and visualizations in this book.

Before We Start

This book aims to be a second course for beginner machine learning engineers who use python. It aims to answer the question "What to do now?" challenge faced by machine learning engineers and researchers when they are stuck below than desired model performance and are looking for ways to improve model performance. The ideal audience for the book is those who did a course on data science/machine learning, researchers who are exploring traditional machine learning algorithms for their thesis, people who are working in their first job as a data scientist, people with 1-3 years of experience in data science, and leaders who transitioned to lead data science team but do not have the background and want to speak with the technical team with confidence. Experienced, seasoned, and veteran data scientists and researchers who want to try new feature engineering, feature selection, and signal processing techniques, as well as to look at existing techniques with a fresh perspective. Supplemental code is provided in the accompanying GitHub page for the book to understand how all the analysis was done. This book is neither mathematical notation heavy nor code heavy. We have instead tried to explain the theoretical nuances, and in some cases aided the explanation with plots and analysis results. This is to give a feel of hand-held projects from beginning to completion.

The objective of the book is to make the readers self-sufficient in the fine art of developing highly accurate machine learning models that make it to production. We will cover different stages of the machine learning project from development to the explanation of model predictions. Learning the fundamental building blocks of building machine learning models will make us better educated machine learning engineers, researchers, capable consultants, and confident leaders. This book will help beginners shorten their learning curve in their journey to become capable machine learning engineers. For experienced consultants, we cover some methods and techniques which are less often heard and used. Techniques that were not available in python previously. For leaders in data science, it will help you get perspective on how to advocate your models by using model explainability.

We assume that the readers are aware of what is regression, and what is classification. Different machine-learning techniques generally used. For example, linear regression, random forest, logistic regression, etc. Different cost functions, such as root mean square error (RMSE), F1 score, Precision, and Recall. The reader has tried their hands at developing models and needs help in improving model performance from the existing level to a higher level.

Although concepts covered in the book are applicable across programming languages, this book is written for python users. It is for the same purpose that several algorithms mentioned in this book were developed from scratch and open-sourced as 4 separate python libraries by the author. It is assumed that the reader has a basic understanding of libraries such as Sklearn for machine learning, Pandas, and Numpy for data manipulation, as well as matplotlib and seaborn for data visualization.

This book is organized in the manner of how a project should be executed in a practical sense from feature engineering to model explanation. Some methods beyond feature engineering and feature selection that can help improve model performance, as well as denoising techniques for signal processing data are explained in Chapter 11.

Although we discuss multiple approaches for improving model performance, there is however no silver bullet in machine learning. It can be possible that even after performing feature engineering, feature extraction, feature selection, ensembling, etc. we might not get desirable model performance. In fact, there are machine learning projects which fail due to data inadequacy, and data quality issues, both of which are beyond the scope of the book. We will use 4 example datasets, out of which 2 datasets are failure examples and 2 are of machine learning success to explain the phenomenon.

Deep learning removes the need for feature engineering and subsequent processes. However, deep learning is suitable for unstructured data such as images, text, audio, and video. For structured tabular data, knowing how to do feature engineering, how to enrich constructed features by creating higher-level features from original features, and knowing which features to keep in your model, can decide the outcome of your project. Hence, for the problems for which you plan to use traditional machine learning, this book tries to make a sincere attempt to educate about the different steps involved in training a machine learning model, in the realm of feature engineering and feature selection.

This book aims to discuss model development after data cleaning has been performed and we have a clean dataset. This book does not teach you how to clean data, i.e. how to perform outlier treatment, and how to deal with missing data. There are many valuable resources available on this topic that are outside the scope of this book

Section I: Introduction

Chapter 1: Introduction

Quite often machine learning projects are shelved even when labeled data is available. This happens either because of less than desired predictive performance of the model. Or difficulty in being able to explain model prediction to non-statistician decision-makers. Building a machine learning model for real-world problems is way more difficult than developing a model for toy datasets. Real-world machine learning problems deal with inadequate features and messy, unclean data. Beyond data cleaning, we also need to do feature engineering, feature extraction, and feature reduction. Finding the combination of features that gives the best predictive performance is important for the success of the project. It is equally important to explain to non-statistician business owners how the model works. Why it predicts certain values, given certain input values? Doing so will ensure that the machine learning project is successful and the model is finally deployed and used. The model is of value to the business.

The scope of the book is limited to tabular data. Toward the end of the book, we will briefly cover natural language processing from the perspective of traditional machine learning. An additional chapter on signal processing is provided for readers whose work overlaps signal processing and machine learning. We will briefly discuss ensemble learning and how we can do feature selection for ensemble models to reduce the complexity and computational power needed while deploying such an ensemble model. We will explain the theory behind each technique and for most of the methods discussed, we will end with a worked python example.

By the time you have completed reading the book, you will take each machine-learning project as a combinatorial problem. You will initiate your projects to identify the combination of feature engineering and feature selection to achieve the best possible model performance. You will also be able to explain your model predictions, as to why it predicts certain values when it sees certain feature values. This book will equip you with all the necessary tools and methods to increase the likelihood of success of your machine learning projects.

1.1 Terminology

Over the next section in this chapter and subsequent chapters, we will use a few terms frequently. Let's understand the meaning of these terms.

1.1.1 Dataset, Variable, and Observation

If we are considering a CSV file that has relevant data for a machine learning project, we will refer to it as a dataset. Each column has data values and is of value for the project. We will refer to it as a 'variable'. A variable could be considered a dependent or target or outcome variable if it is the central point of focus of the project. The variables that we want to use for modeling the behavior of the target variable could be referred to as features or independent variables. Each row in the dataset will be referred to as an observation or record.

1.1.2 Feature Engineering

Feature engineering or feature construction is the process of creating new features in the data set using domain knowledge, as well as creating higher order features from original features.

A bank employee responsible for the upkeep of minimum cash reserve in a bank branch might be aware more customers visit branches for cash withdrawals just a few days before holidays, than the number of customers who visit during other days. Similarly, fewer customers might come to the branch for cash withdrawals, on the next day of the festival. This information can be used for creating 2 separate binary 1|0 indicator features to represent days before and after the festivals. This is an example of feature engineering using domain knowledge.

In addition to domain knowledge, we can also use data-driven approaches for feature construction, such as exploratory data analysis (EDA). For example, during EDA if we observe data anomalies, we can probe this with subject matter experts and if it indeed turns out to be a valid pattern in the data, and is of use for the machine learning project, we can create features that represent the pattern in raw data.

We can also create new features from existing features using transformations of the original features. For example, instead of taking original values, we can take the square of the original value to increase the magnitude. If body weight recorded for an adult is 71.9, 72, and 69.9, etc., we can instead take 5169.61, 5184, and 4886.01, which are squares of the original values. By performing a square, the difference in weight is more noticeable than the original values.

1.1.3 Feature Extraction

Feature extraction is the process of finding alternative representations for the original features created during feature engineering. It converts features into a lower dimension. This makes the structure of the data clearer for the model to learn.

For example, if there are 500 features, a technique such as principal component analysis (PCA) can convert these features into a lesser number of principal components, say 20. It might be easier and faster for the model to learn from fewer features that have maximum information.

1.1.4 Feature Selection

Feature selection deals with choosing a subset of features from a list of given features. This helps identify features that are rich in useful information for the model, ultimately resulting in a less complex model of high predictive power.

For example, if we have 5000 features, by performing feature selection we can identify a very small number of features that gives the highest performance. By having a small set of features, it also becomes easier to explain model prediction to laymen.

1.1.5 Cost Function

'Cost function', 'model metric', 'model performance', and 'predictive performance' are used interchangeably as synonyms in this book. For a regression model, example cost functions are root mean square error (RMSE), mean absolute error (MAE), etc. For a classification model, example cost functions could be F1 score, precision, and recall.

1.2 Process of Training a Machine Learning Model

There are 2 steps for training a machine learning model for structured data. Feature engineering is the bare minimum needed for developing a model using a dataset. Feature selection is performed on a need basis if the features created through feature engineering can't help us develop a model of desirable predictive power.

After these 2 steps are performed, we can also perform hyperparameter tuning and ensemble learning. Although hyperparameter tuning and ensemble learning are not the book's focus, hyperparameter tuning and ensemble learning are briefly discussed in sections 11.1 and 11.2 in chapter 11.

1.3 Preventing Overfitting

A dataset can be divided into a training dataset, and external test data in an 80:20, 70:30, or even 60:40 ratio, depending on the availability of data. Then a further split for validation data is created from the remaining training data. The remaining training dataset can be divided further into train and development test data in the same ratio. We then train our model in iterations after performing feature engineering and feature selection. At each iteration, we train our model, and we test its performance on the development test data and validation data. Once we get a model of acceptable performance, we test it on external test data to ensure it generalizes well on unseen data.

Bias-variance tradeoff tells us that by adding additional features, overfitting can happen. Overfitting can also happen by doing feature selection. If the model has learned our data in a way that gives the best predictive power for the dependent variable, it doesn't guarantee that it will generalize well on unseen data. What it only guarantees is that it can perform well on the development test data and validation data for validating our model.

One way to overcome this challenge is to test your model on multiple development test data to understand how well it can generalize. This can be performed by cross-validation.

Generally, we perform 5-fold cross-validation. We obtain the training dataset after separating external test data and validation data. The remaining training data is divided into 5 equal parts, and 4 parts are used for training. The remaining 1 part is for the development test. We also use validation data for measuring model performance. We repeat this 5 times and average the model metric across 5 samples for the development test data and validation data. Doing this reduces the likelihood of overfitting. In addition to cross-validation, we will test the model on external test data to ensure that indeed the cross-validation score is reliable.

If the average of the model metric of 5 cross-validations is very different from the model metric of external test data, we will need to do further probing in the dataset to check if the data distribution is very different in 5-fold cross-validation and validation data Vs external test data. The object of the probe should be to ensure that data distribution is close to the real-world values in all 3 datasets. i.e. 1) training,

2) development test data in all cross-validation samples, 3) validation data, as well as in 4) external test data. If any anomaly is identified, it should be subject to further investigation.

Using 5-fold cross-validation is subject to the availability of data. Although used as a universal standard, we should use 5-fold only when we have sufficient data. In the absence of volume, we can increase cross-validation from 5-fold to 10-fold, to get a more robust averaged model metric. 'More' and 'less' are subjective when it comes to datasets. For example, for the training of the Twitter sentiment classifier, 8000 tweets can be considered small, if we compare it against billions of tweets present in the Twitter database. If we still have to use such a small number of tweets dataset for modeling, it will be wise to use 10-fold cross-validation. However, while modeling the average lifespan of endangered species, 8000 observations can be considered voluminous data and we can limit cross-validation to 5-fold.

If the data has distinct strata, we should ensure that train and test samples in cross-validation, as well as external hold-out samples, should all be representative of all the strata. We should proceed with stratified k-fold cross-validation in such cases. For example, if we are developing a model for predicting the likelihood of occurrence of a specific disease amongst people living in the US, given all the body vitals and characteristics such as blood pressure, sugar level, heartbeat rate, sodium level, cholesterol, daily minutes of exercise, body weight, height, etc. Such a dataset should be collected from all the states. Overall health and well-being might have less impact on a person based on which state they belong to. Hence, we cannot use the state as a feature to train our model. However, we should ensure that in the training and test samples across all cross-validations, each state should have an equal percentage of observations. This will ensure that model will generalize well to real-world circumstances.

While doing train vs test split and even for external test data, we should give due regard to the unit of data. Most often one single observation is a unit of data. We can split such a dataset into cross-validation samples. However, when a data observation belongs to a distinguishable group of similar observations, as a single unit, we will consider it as a unit of data.

For example, if we are evaluating the effectiveness of new medicine in regulating the blood pressure of individual patients, we will record the dosage amount and resultant blood pressure for multiple days. In this case, data recorded for all days for a specific patient will be a unit of data. While splitting data into the training set, external test data, and development test data, we should ensure that a patient's data is present entirely either in training, external test data, validation data, or development test data. In no case, a patient's data should be present in more than one group from training, external test data, or development test data. If a patient's data is present in both training and test data simultaneously, then the results could

be biased, as it will lead to over-optimistic results for the same patient, as the model is already familiar with similar values for the patient in training data. At an aggregate level, it can give us overoptimistic model performance. Hence, we should identify units of data and should ensure that data from a single unit is not present in training, test, and/or external test data simultaneously.

If cross-validation is done right, feature engineering and feature selection can help us identify a smaller set of features that give the best performance for predicting the target variable.

1.4 Code Conventions

Most of the codes are provided in the accompanying GitHub repository [1]. We will use code sparingly in the book and instead focus on understanding the key concepts. Wherever we have used code, comments for the code will be presented in the below format

```
# We will add 1+1
```

Code will be in the below format.

```
print("1+1 is: "+str(1+1))
```

1.5 Datasets Used

We will use 4 separate datasets throughout this book for regression and classification problems. We will benchmark model performance and compare the performance of different techniques. These datasets are explained in the first four datasets from sections 1.5.1 to 1.5.3. We will also use 2 datasets for signal processing in chapter 11. These datasets are explained in sections 1.5.4 and 1.5.5.

1.5.1 Hotel Booking Demand Datasets

This dataset [2] has demand data for 2 hotels. Hotel H1 is a resort hotel that attracts customers who will like to stay at the hotel for recreation purposes. Hotel H2 is a city hotel, that people visit for business purposes.

We will use data from hotel H1 for developing classification models to predict the likelihood of cancellation for reservations. The objective of the classification model is to minimize losses from cancellations and increase profitability. Many customers cancel their reservations. This could mean a loss of revenue. If we know in advance which reservation is going to be canceled, we can try to resell the room, even though it is not canceled at the moment. However, it can lead to a situation when both guests arrive at the hotel and demand the room they have reserved. To avoid this situation, our modeling objective is to build a model which can predict cancellation with a high degree of accuracy. In other words, 'precision' will be our cost function. As a result, we might have a model which has a low recall, but high precision. We are fine with such a model, as long as it can predict cancellations with a low margin of error. Imagine a hypothetical model that has 95 precision and 30 recall. It simply means, for all predictions it makes for cancellations, 95 times out of 100 could actually be a cancellation. Although the precision is not the ideal 100, 95 precision is a good score for a statistician. A low recall of 30 means, out of all cancellations that happened, the model could only identify 30 percent of such cancellations. For a statistician or machine learning engineer, recall value of 30 is a less than ideal situation. However, for business owners, this is an acceptable situation. For a hotel revenue manager, recall value of 30 means that the hotel will be able to reduce their cancellation losses by 30 percent. This is still better than a situation where the hotel cannot save anything without the model. If not all, in many cases businesses desire a workable model. A perfect model which meets all criteria of theoretical statistics and theoretical machine learning is not always desired by real world businesses.

Data for hotel H2 will be used for developing a regression model to predict the total occupancy for rooms for future check-in dates. This dataset was not in the format where it could have been used for the regression model. The author has used his domain knowledge in the hotel industry to perform preprocessing and data cleaning to bring it into a usable format. Hotels have different types of customers. Two major groups are 'transient' and 'group'. Transient customers often seek short hotel stays, such as people who travel to different cities for business purposes, or other customers who want short-term stays. We will like to develop a model to predict total transient bookings at the check-in date level, at each booking window between 0 to 100 days before the date of check-in for the city hotel H2.

Booking day is the day when customers book the room for the check-in date in advance. For example, if you want to visit Hawaii on Christmas of 2022 and you are reserving the room on the 1st of October, 2022, then October 1st, 2022 will be the booking date and December 25th, 2022, the day of Christmas will be the check-in date. The number of days between these two dates is called lead time.

Sometimes customers book rooms, but they do not show up. We will remove these customers from our modeling problem, as these are very negligible. ADR is the average daily rate, for any check-in date it represents total revenue from bookings divided by total rooms sold. We will remove records for ADR which seem outliers, as we cannot get any explanation from the original authors about the nature of outliers.

We will use ADR between the bottom 5th percentile and the top 99.99 percentile. There were transactions for which no value was specified for the number of adults and children as guests. These will also be removed.

Some customers buy rooms for continuous stay. If a customer is reserving on the 1st of January for staying at the H2 hotel on the 7th and 8th of January, the lead time for the 7th of January will be 6 days and for the 8th of January will be 7 days.

We assume that one room can accommodate 2 adults and 2 kids in a hotel. If a transaction has 2 adults and 2 kids, it will be considered as 1 room. If a transaction has 1 adult and 2 kids, 2 adults, and 1 kid, or 1 adult and 1 kid, it will be considered as 1 room.

If there are more than 2 adults, the number of rooms will be calculated as the number of adults divided by 2. If the result is a decimal, it will be rounded to a higher number. For example, if there are 3 adults, 3 divided by 2 is 1.5. We will round 1.5 to a higher ceiling of 2. Hence concluding that 3 adults can stay in 2 rooms. Similarly, children are also considered for calculating the number of rooms.

1.5.2 Car Sales

This is sourced from Kaggle [3] and has information about the attributes of used cars and its price in India. It will be used for developing regression models. We will use the 'Car details v3.csv' file from version 3 of the dataset. The original file has details of 8128 cars. We have removed certain records which had inadequate data or missing data. These are mileage features for values of 0.0kmpl or left blank. Engine, torque, or seats are left blank. max_power as 0, 'bhp' or blank. km_driven as 1. After removing these observations, 7905 observations remain. Mileage, engine, and max_power were converted to numeric features after removing string suffixes.

1.5.3 Coupon Recommendation

In-vehicle coupon recommendation Data Set [4] studies the behavior of individuals and whether they will accept or not accept coupons. This dataset was generated through a survey. This survey describes different driving scenarios and asks the person if they will accept the coupon.

1.5.4 Raman Spectroscopy of Skimmed Milk Samples

This dataset [5] has a Matrix of quantitative whole spectrum analysis of 45 spectra on 21451 on skimmed milk samples. We will use this for discussing Raman spectra in chapter 11.

1.5.5 Beaver Body Temperatures

This dataset has body temperature of 2 beavers [6], measured every 10 minutes by telemetry. This has data for less than a day for each of the two beavers. We will discuss this in chapter 11 for filtering method.

1.6 References

[1] https://github.com/statguyuser/feature_engineering_and_selection_for_explanable_models

[2] Nuno Antonio, Ana de Almeida, Luis Nunes, Hotel booking demand datasets, Data in Brief, Volume 22, 2019, Pages 41-49, ISSN 2352-3409, https://doi.org/10.1016/j.dib.2018.11.126.

[3] https://www.kaggle.com/datasets/nehalbirla/vehicle-dataset-from-cardekho?select=Car+details+v3.csv

[4] Wang, Tong, Cynthia Rudin, Finale Doshi-Velez, Yimin Liu, Erica Klampfl, and Perry MacNeille. 'A bayesian framework for learning rule sets for interpretable classification.' The Journal of Machine Learning Research 18, no. 1 (2017): 2357-2393.

[5] Kristian Hovde Liland, Bjørn-Helge Mevik, Elling-Olav Rukke, Trygve Almøy, Morten Skaugen and Tomas Isaksson (2009) Quantitative whole spectrum analysis with MALDI-TOF MS, Part I: Measurement optimisation. Chemometrics and Intelligent Laboratory Systems, 96(2), 210–218.

[6] P. S. Reynolds (1994) Time-series analyses of beaver body temperatures. Chapter 11 of Lange, N., Ryan, L., Billard, L., Brillinger, D., Conquest, L. and Greenhouse, J. eds (1994) Case Studies in Biometry. New York: John Wiley and Sons.

Section II: Feature Engineering

Chapter 2: Domain Specific Feature Engineering

2.1 Introduction

Building a machine learning model is akin to raising children. Children grow into fully functional adults when given parenting and proper care. Similarly, when feature engineering, feature extraction, and feature selection are adequately done, it can give us a model with high predictive power and less complexity.

2.2 Domain-Specific Feature Engineering

Feature engineering is the most basic building block in training machine learning models. It can be likened to teaching the basic alphabet to children. If a child knows different alphabets, we can then teach him/her words. Along similar lines, we will need to explicitly create features to represent domain knowledge. Just because a pattern exists in data, does not mean that the machine learning model will be able to learn it. It may or may not. It is always better to explicitly create features. Someone's ability to create domain-specific features depends on the degree of awareness of the way domain, and the way the business operates on a day-to-day basis.

2.2.1 Ask Probing Questions

While initiating work on a machine learning problem, it is beneficial to interact with the end users. Two tasks should be undertaken at this stage, and if necessary, clarification should be sought from the intended users. Answers to these two questions could help us understand the domain better. Some of these answers can then guide us in feature engineering.

1) Try to understand how this task is done currently. If the users do not already have a machine learning model, do they do it through a rule-based system or manual calculations? Explore if some of these rules and calculations can be used for creating features for the machine learning model.

2) What are the things that could impact the dependent variable positively or negatively?

Let's take a look at a practical example of the hotel industry. Hotels operate on a check-in date basis. Guests reserve their stay in a hotel before staying. The day when they arrive at the hotel is called the "check-in" date. A revenue manager in a hotel might be interested to know in advance, how many rooms

will be occupied by guests in total for a specific check-in date. This will help him/her to price the rooms in a way to maximize profit.

As a machine learning engineer, our goal will be to develop a model that predicts hotel occupancy for a specific future day. Machine learning engineers who don't know how the hotel industry operates will believe it is a univariate time series forecast and will develop a forecasting model. In this case, if we ask the revenue manager our 2 questions, it will look like

1) How are you forecasting occupancy currently, as you do not have a model? Even if it's gut feeling, what factors do you consider for arriving at a conclusion?

2) What are the factors that could result in increased or decreased occupancy for a specific check-in date?

Some possible answers to the first question could be that they forecast occupancy by looking at demand for the particular check-in date so far, and demand for the room on the same check-in date in the previous year. This could suggest that there is seasonality in hotel demand and there is an effect from demand so far for the check-in date.

For the second question, hotel occupancy for a specific date decrease, if competing hotels reduce their price for the same check-in date. If there are more negative reviews for the hotel, in comparison to competing hotels, then a smaller number of guests will stay in the hotel. Price affects hotel occupancy inversely. If price of room is very high, few guests will come to stay, whereas if price is very low, there might not be any vacant rooms in the hotel. Online reputation directly affects the hotel reservations. Demand can also be affected if holidays or events are happening close to the check-in date.

2.2.2 Literature Review

Quite often machine learning engineers are required to solve business problems in new domains where they lack subject matter expertise. In some of these cases, subject matter experts (SMEs) are not accessible or access to SMEs is limited. Reasons could be any, such as workload of their own, SMEs being C-level executives and having limited time, etc. In such situations, it can be helpful to read research papers and books on the subject matter. In particular, try to understand the type of features used and the way data was organized for modeling for similar machine learning problems.

Let's take the example of hotel industry booking prediction for check-in date. There is an argument put forth and proven [1] by data that the social media reputation of a hotel has an impact on hotel room reservations, as against competitor hotels. A simple search on google scholar can help us find useful research papers that can help us improve feature engineering for the project.

2.3 References

[1] Chris K. Anderson, The Impact of Social Media on Lodging Performance, 2012, https://hdl.handle.net/1813/71194

Feature Engineering & Selection for Explainable Models

A Second Course for Data Scientists

Md Azimul Haque

Contents

Foreword

I have been teaching data science as a part-time faculty and have taught hundreds of students. Many of my students have successfully transitioned to a data science career. I also have a set of open-source contributions as python libraries. I often get requests from my previous students when they start working on their first real machine learning project. I found the root cause of many challenges faced by my students who recently transitioned into data science and machine learning. I have tried to address these issues in my book and would like to dedicate this book to all my students for all the love and respect I have received.

There are many methods for feature engineering, feature selection and signal processing which were not available previously in python. I have coded and open-sourced these methods as python libraries over the years. Often, I get approached by researchers who use my python libraries, for understanding how to use certain algorithms and best practices for these algorithms. I have tried to explain these with the aid of underlying theory, and visualizations in this book.

Before We Start

This book aims to be a second course for beginner machine learning engineers who use python. It aims to answer the question "What to do now?" challenge faced by machine learning engineers and researchers when they are stuck below than desired model performance and are looking for ways to improve model performance. The ideal audience for the book is those who did a course on data science/machine learning, researchers who are exploring traditional machine learning algorithms for their thesis, people who are working in their first job as a data scientist, people with 1-3 years of experience in data science, and leaders who transitioned to lead data science team but do not have the background and want to speak with the technical team with confidence. Experienced, seasoned, and veteran data scientists and researchers who want to try new feature engineering, feature selection, and signal processing techniques, as well as to look at existing techniques with a fresh perspective. Supplemental code is provided in the accompanying GitHub page for the book to understand how all the analysis was done. This book is neither mathematical notation heavy nor code heavy. We have instead tried to explain the theoretical nuances, and in some cases aided the explanation with plots and analysis results. This is to give a feel of hand-held projects from beginning to completion.

The objective of the book is to make the readers self-sufficient in the fine art of developing highly accurate machine learning models that make it to production. We will cover different stages of the machine learning project from development to the explanation of model predictions. Learning the fundamental building blocks of building machine learning models will make us better educated machine learning engineers, researchers, capable consultants, and confident leaders. This book will help beginners shorten their learning curve in their journey to become capable machine learning engineers. For experienced consultants, we cover some methods and techniques which are less often heard and used. Techniques that were not available in python previously. For leaders in data science, it will help you get perspective on how to advocate your models by using model explainability.

We assume that the readers are aware of what is regression, and what is classification. Different machine-learning techniques generally used. For example, linear regression, random forest, logistic regression, etc. Different cost functions, such as root mean square error (RMSE), F1 score, Precision, and Recall. The reader has tried their hands at developing models and needs help in improving model performance from the existing level to a higher level.

Although concepts covered in the book are applicable across programming languages, this book is written for python users. It is for the same purpose that several algorithms mentioned in this book were developed from scratch and open-sourced as 4 separate python libraries by the author. It is assumed that the reader has a basic understanding of libraries such as Sklearn for machine learning, Pandas, and Numpy for data manipulation, as well as matplotlib and seaborn for data visualization.

This book is organized in the manner of how a project should be executed in a practical sense from feature engineering to model explanation. Some methods beyond feature engineering and feature selection that can help improve model performance, as well as denoising techniques for signal processing data are explained in Chapter 11.

Although we discuss multiple approaches for improving model performance, there is however no silver bullet in machine learning. It can be possible that even after performing feature engineering, feature extraction, feature selection, ensembling, etc. we might not get desirable model performance. In fact, there are machine learning projects which fail due to data inadequacy, and data quality issues, both of which are beyond the scope of the book. We will use 4 example datasets, out of which 2 datasets are failure examples and 2 are of machine learning success to explain the phenomenon.

Deep learning removes the need for feature engineering and subsequent processes. However, deep learning is suitable for unstructured data such as images, text, audio, and video. For structured tabular data, knowing how to do feature engineering, how to enrich constructed features by creating higher-level features from original features, and knowing which features to keep in your model, can decide the outcome of your project. Hence, for the problems for which you plan to use traditional machine learning, this book tries to make a sincere attempt to educate about the different steps involved in training a machine learning model, in the realm of feature engineering and feature selection.

This book aims to discuss model development after data cleaning has been performed and we have a clean dataset. This book does not teach you how to clean data, i.e. how to perform outlier treatment, and how to deal with missing data. There are many valuable resources available on this topic that are outside the scope of this book

Section I: Introduction

Chapter 1: Introduction

Quite often machine learning projects are shelved even when labeled data is available. This happens either because of less than desired predictive performance of the model. Or difficulty in being able to explain model prediction to non-statistician decision-makers. Building a machine learning model for real-world problems is way more difficult than developing a model for toy datasets. Real-world machine learning problems deal with inadequate features and messy, unclean data. Beyond data cleaning, we also need to do feature engineering, feature extraction, and feature reduction. Finding the combination of features that gives the best predictive performance is important for the success of the project. It is equally important to explain to non-statistician business owners how the model works. Why it predicts certain values, given certain input values? Doing so will ensure that the machine learning project is successful and the model is finally deployed and used. The model is of value to the business.

The scope of the book is limited to tabular data. Toward the end of the book, we will briefly cover natural language processing from the perspective of traditional machine learning. An additional chapter on signal processing is provided for readers whose work overlaps signal processing and machine learning. We will briefly discuss ensemble learning and how we can do feature selection for ensemble models to reduce the complexity and computational power needed while deploying such an ensemble model. We will explain the theory behind each technique and for most of the methods discussed, we will end with a worked python example.

By the time you have completed reading the book, you will take each machine-learning project as a combinatorial problem. You will initiate your projects to identify the combination of feature engineering and feature selection to achieve the best possible model performance. You will also be able to explain your model predictions, as to why it predicts certain values when it sees certain feature values. This book will equip you with all the necessary tools and methods to increase the likelihood of success of your machine learning projects.

1.1 Terminology

Over the next section in this chapter and subsequent chapters, we will use a few terms frequently. Let's understand the meaning of these terms.

1.1.1 Dataset, Variable, and Observation

If we are considering a CSV file that has relevant data for a machine learning project, we will refer to it as a dataset. Each column has data values and is of value for the project. We will refer to it as a 'variable'. A variable could be considered a dependent or target or outcome variable if it is the central point of focus of the project. The variables that we want to use for modeling the behavior of the target variable could be referred to as features or independent variables. Each row in the dataset will be referred to as an observation or record.

1.1.2 Feature Engineering

Feature engineering or feature construction is the process of creating new features in the data set using domain knowledge, as well as creating higher order features from original features.

A bank employee responsible for the upkeep of minimum cash reserve in a bank branch might be aware more customers visit branches for cash withdrawals just a few days before holidays, than the number of customers who visit during other days. Similarly, fewer customers might come to the branch for cash withdrawals, on the next day of the festival. This information can be used for creating 2 separate binary 1|0 indicator features to represent days before and after the festivals. This is an example of feature engineering using domain knowledge.

In addition to domain knowledge, we can also use data-driven approaches for feature construction, such as exploratory data analysis (EDA). For example, during EDA if we observe data anomalies, we can probe this with subject matter experts and if it indeed turns out to be a valid pattern in the data, and is of use for the machine learning project, we can create features that represent the pattern in raw data.

We can also create new features from existing features using transformations of the original features. For example, instead of taking original values, we can take the square of the original value to increase the magnitude. If body weight recorded for an adult is 71.9, 72, and 69.9, etc., we can instead take 5169.61, 5184, and 4886.01, which are squares of the original values. By performing a square, the difference in weight is more noticeable than the original values.

1.1.3 Feature Extraction

Feature extraction is the process of finding alternative representations for the original features created during feature engineering. It converts features into a lower dimension. This makes the structure of the data clearer for the model to learn.

For example, if there are 500 features, a technique such as principal component analysis (PCA) can convert these features into a lesser number of principal components, say 20. It might be easier and faster for the model to learn from fewer features that have maximum information.

1.1.4 Feature Selection

Feature selection deals with choosing a subset of features from a list of given features. This helps identify features that are rich in useful information for the model, ultimately resulting in a less complex model of high predictive power.

For example, if we have 5000 features, by performing feature selection we can identify a very small number of features that gives the highest performance. By having a small set of features, it also becomes easier to explain model prediction to laymen.

1.1.5 Cost Function

'Cost function', 'model metric', 'model performance', and 'predictive performance' are used interchangeably as synonyms in this book. For a regression model, example cost functions are root mean square error (RMSE), mean absolute error (MAE), etc. For a classification model, example cost functions could be F1 score, precision, and recall.

1.2 Process of Training a Machine Learning Model

There are 2 steps for training a machine learning model for structured data. Feature engineering is the bare minimum needed for developing a model using a dataset. Feature selection is performed on a need basis if the features created through feature engineering can't help us develop a model of desirable predictive power.

After these 2 steps are performed, we can also perform hyperparameter tuning and ensemble learning. Although hyperparameter tuning and ensemble learning are not the book's focus, hyperparameter tuning and ensemble learning are briefly discussed in sections 11.1 and 11.2 in chapter 11.

1.3 Preventing Overfitting

A dataset can be divided into a training dataset, and external test data in an 80:20, 70:30, or even 60:40 ratio, depending on the availability of data. Then a further split for validation data is created from the remaining training data. The remaining training dataset can be divided further into train and development test data in the same ratio. We then train our model in iterations after performing feature engineering and feature selection. At each iteration, we train our model, and we test its performance on the development test data and validation data. Once we get a model of acceptable performance, we test it on external test data to ensure it generalizes well on unseen data.

Bias-variance tradeoff tells us that by adding additional features, overfitting can happen. Overfitting can also happen by doing feature selection. If the model has learned our data in a way that gives the best predictive power for the dependent variable, it doesn't guarantee that it will generalize well on unseen data. What it only guarantees is that it can perform well on the development test data and validation data for validating our model.

One way to overcome this challenge is to test your model on multiple development test data to understand how well it can generalize. This can be performed by cross-validation.

Generally, we perform 5-fold cross-validation. We obtain the training dataset after separating external test data and validation data. The remaining training data is divided into 5 equal parts, and 4 parts are used for training. The remaining 1 part is for the development test. We also use validation data for measuring model performance. We repeat this 5 times and average the model metric across 5 samples for the development test data and validation data. Doing this reduces the likelihood of overfitting. In addition to cross-validation, we will test the model on external test data to ensure that indeed the cross-validation score is reliable.

If the average of the model metric of 5 cross-validations is very different from the model metric of external test data, we will need to do further probing in the dataset to check if the data distribution is very different in 5-fold cross-validation and validation data Vs external test data. The object of the probe should be to ensure that data distribution is close to the real-world values in all 3 datasets. i.e. 1) training,

2) development test data in all cross-validation samples, 3) validation data, as well as in 4) external test data. If any anomaly is identified, it should be subject to further investigation.

Using 5-fold cross-validation is subject to the availability of data. Although used as a universal standard, we should use 5-fold only when we have sufficient data. In the absence of volume, we can increase cross-validation from 5-fold to 10-fold, to get a more robust averaged model metric. 'More' and 'less' are subjective when it comes to datasets. For example, for the training of the Twitter sentiment classifier, 8000 tweets can be considered small, if we compare it against billions of tweets present in the Twitter database. If we still have to use such a small number of tweets dataset for modeling, it will be wise to use 10-fold cross-validation. However, while modeling the average lifespan of endangered species, 8000 observations can be considered voluminous data and we can limit cross-validation to 5-fold.

If the data has distinct strata, we should ensure that train and test samples in cross-validation, as well as external hold-out samples, should all be representative of all the strata. We should proceed with stratified k-fold cross-validation in such cases. For example, if we are developing a model for predicting the likelihood of occurrence of a specific disease amongst people living in the US, given all the body vitals and characteristics such as blood pressure, sugar level, heartbeat rate, sodium level, cholesterol, daily minutes of exercise, body weight, height, etc. Such a dataset should be collected from all the states. Overall health and well-being might have less impact on a person based on which state they belong to. Hence, we cannot use the state as a feature to train our model. However, we should ensure that in the training and test samples across all cross-validations, each state should have an equal percentage of observations. This will ensure that model will generalize well to real-world circumstances.

While doing train vs test split and even for external test data, we should give due regard to the unit of data. Most often one single observation is a unit of data. We can split such a dataset into cross-validation samples. However, when a data observation belongs to a distinguishable group of similar observations, as a single unit, we will consider it as a unit of data.

For example, if we are evaluating the effectiveness of new medicine in regulating the blood pressure of individual patients, we will record the dosage amount and resultant blood pressure for multiple days. In this case, data recorded for all days for a specific patient will be a unit of data. While splitting data into the training set, external test data, and development test data, we should ensure that a patient's data is present entirely either in training, external test data, validation data, or development test data. In no case, a patient's data should be present in more than one group from training, external test data, or development test data. If a patient's data is present in both training and test data simultaneously, then the results could

be biased, as it will lead to over-optimistic results for the same patient, as the model is already familiar with similar values for the patient in training data. At an aggregate level, it can give us overoptimistic model performance. Hence, we should identify units of data and should ensure that data from a single unit is not present in training, test, and/or external test data simultaneously.

If cross-validation is done right, feature engineering and feature selection can help us identify a smaller set of features that give the best performance for predicting the target variable.

1.4 Code Conventions

Most of the codes are provided in the accompanying GitHub repository [1]. We will use code sparingly in the book and instead focus on understanding the key concepts. Wherever we have used code, comments for the code will be presented in the below format

```
# We will add 1+1
```

Code will be in the below format.

```
print("1+1 is: "+str(1+1))
```

1.5 Datasets Used

We will use 4 separate datasets throughout this book for regression and classification problems. We will benchmark model performance and compare the performance of different techniques. These datasets are explained in the first four datasets from sections 1.5.1 to 1.5.3. We will also use 2 datasets for signal processing in chapter 11. These datasets are explained in sections 1.5.4 and 1.5.5.

1.5.1 Hotel Booking Demand Datasets

This dataset [2] has demand data for 2 hotels. Hotel H1 is a resort hotel that attracts customers who will like to stay at the hotel for recreation purposes. Hotel H2 is a city hotel, that people visit for business purposes.

We will use data from hotel H1 for developing classification models to predict the likelihood of cancellation for reservations. The objective of the classification model is to minimize losses from cancellations and increase profitability. Many customers cancel their reservations. This could mean a loss of revenue. If we know in advance which reservation is going to be canceled, we can try to resell the room, even though it is not canceled at the moment. However, it can lead to a situation when both guests arrive at the hotel and demand the room they have reserved. To avoid this situation, our modeling objective is to build a model which can predict cancellation with a high degree of accuracy. In other words, 'precision' will be our cost function. As a result, we might have a model which has a low recall, but high precision. We are fine with such a model, as long as it can predict cancellations with a low margin of error. Imagine a hypothetical model that has 95 precision and 30 recall. It simply means, for all predictions it makes for cancellations, 95 times out of 100 could actually be a cancellation. Although the precision is not the ideal 100, 95 precision is a good score for a statistician. A low recall of 30 means, out of all cancellations that happened, the model could only identify 30 percent of such cancellations. For a statistician or machine learning engineer, recall value of 30 is a less than ideal situation. However, for business owners, this is an acceptable situation. For a hotel revenue manager, recall value of 30 means that the hotel will be able to reduce their cancellation losses by 30 percent. This is still better than a situation where the hotel cannot save anything without the model. If not all, in many cases businesses desire a workable model. A perfect model which meets all criteria of theoretical statistics and theoretical machine learning is not always desired by real world businesses.

Data for hotel H2 will be used for developing a regression model to predict the total occupancy for rooms for future check-in dates. This dataset was not in the format where it could have been used for the regression model. The author has used his domain knowledge in the hotel industry to perform preprocessing and data cleaning to bring it into a usable format. Hotels have different types of customers. Two major groups are 'transient' and 'group'. Transient customers often seek short hotel stays, such as people who travel to different cities for business purposes, or other customers who want short-term stays. We will like to develop a model to predict total transient bookings at the check-in date level, at each booking window between 0 to 100 days before the date of check-in for the city hotel H2.

Booking day is the day when customers book the room for the check-in date in advance. For example, if you want to visit Hawaii on Christmas of 2022 and you are reserving the room on the 1st of October, 2022, then October 1st, 2022 will be the booking date and December 25th, 2022, the day of Christmas will be the check-in date. The number of days between these two dates is called lead time.

Sometimes customers book rooms, but they do not show up. We will remove these customers from our modeling problem, as these are very negligible. ADR is the average daily rate, for any check-in date it represents total revenue from bookings divided by total rooms sold. We will remove records for ADR which seem outliers, as we cannot get any explanation from the original authors about the nature of outliers.

We will use ADR between the bottom 5th percentile and the top 99.99 percentile. There were transactions for which no value was specified for the number of adults and children as guests. These will also be removed.

Some customers buy rooms for continuous stay. If a customer is reserving on the 1st of January for staying at the H2 hotel on the 7th and 8th of January, the lead time for the 7th of January will be 6 days and for the 8th of January will be 7 days.

We assume that one room can accommodate 2 adults and 2 kids in a hotel. If a transaction has 2 adults and 2 kids, it will be considered as 1 room. If a transaction has 1 adult and 2 kids, 2 adults, and 1 kid, or 1 adult and 1 kid, it will be considered as 1 room.

If there are more than 2 adults, the number of rooms will be calculated as the number of adults divided by 2. If the result is a decimal, it will be rounded to a higher number. For example, if there are 3 adults, 3 divided by 2 is 1.5. We will round 1.5 to a higher ceiling of 2. Hence concluding that 3 adults can stay in 2 rooms. Similarly, children are also considered for calculating the number of rooms.

1.5.2 Car Sales

This is sourced from Kaggle [3] and has information about the attributes of used cars and its price in India. It will be used for developing regression models. We will use the 'Car details v3.csv' file from version 3 of the dataset. The original file has details of 8128 cars. We have removed certain records which had inadequate data or missing data. These are mileage features for values of 0.0kmpl or left blank. Engine, torque, or seats are left blank. max_power as 0, 'bhp' or blank. km_driven as 1. After removing these observations, 7905 observations remain. Mileage, engine, and max_power were converted to numeric features after removing string suffixes.

1.5.3 Coupon Recommendation

In-vehicle coupon recommendation Data Set [4] studies the behavior of individuals and whether they will accept or not accept coupons. This dataset was generated through a survey. This survey describes different driving scenarios and asks the person if they will accept the coupon.

1.5.4 Raman Spectroscopy of Skimmed Milk Samples

This dataset [5] has a Matrix of quantitative whole spectrum analysis of 45 spectra on 21451 on skimmed milk samples. We will use this for discussing Raman spectra in chapter 11.

1.5.5 Beaver Body Temperatures

This dataset has body temperature of 2 beavers [6], measured every 10 minutes by telemetry. This has data for less than a day for each of the two beavers. We will discuss this in chapter 11 for filtering method.

1.6 References

[1] https://github.com/statguyuser/feature_engineering_and_selection_for_explanable_models

[2] Nuno Antonio, Ana de Almeida, Luis Nunes, Hotel booking demand datasets, Data in Brief, Volume 22, 2019, Pages 41-49, ISSN 2352-3409, https://doi.org/10.1016/j.dib.2018.11.126.

[3] https://www.kaggle.com/datasets/nehalbirla/vehicle-dataset-from-cardekho?select=Car+details+v3.csv

[4] Wang, Tong, Cynthia Rudin, Finale Doshi-Velez, Yimin Liu, Erica Klampfl, and Perry MacNeille. 'A bayesian framework for learning rule sets for interpretable classification.' The Journal of Machine Learning Research 18, no. 1 (2017): 2357-2393.

[5] Kristian Hovde Liland, Bjørn-Helge Mevik, Elling-Olav Rukke, Trygve Almøy, Morten Skaugen and Tomas Isaksson (2009) Quantitative whole spectrum analysis with MALDI-TOF MS, Part I: Measurement optimisation. Chemometrics and Intelligent Laboratory Systems, 96(2), 210–218.

[6] P. S. Reynolds (1994) Time-series analyses of beaver body temperatures. Chapter 11 of Lange, N., Ryan, L., Billard, L., Brillinger, D., Conquest, L. and Greenhouse, J. eds (1994) Case Studies in Biometry. New York: John Wiley and Sons.

Section II: Feature Engineering

Chapter 2: Domain Specific Feature Engineering

2.1 Introduction

Building a machine learning model is akin to raising children. Children grow into fully functional adults when given parenting and proper care. Similarly, when feature engineering, feature extraction, and feature selection are adequately done, it can give us a model with high predictive power and less complexity.

2.2 Domain-Specific Feature Engineering

Feature engineering is the most basic building block in training machine learning models. It can be likened to teaching the basic alphabet to children. If a child knows different alphabets, we can then teach him/her words. Along similar lines, we will need to explicitly create features to represent domain knowledge. Just because a pattern exists in data, does not mean that the machine learning model will be able to learn it. It may or may not. It is always better to explicitly create features. Someone's ability to create domain-specific features depends on the degree of awareness of the way domain, and the way the business operates on a day-to-day basis.

2.2.1 Ask Probing Questions

While initiating work on a machine learning problem, it is beneficial to interact with the end users. Two tasks should be undertaken at this stage, and if necessary, clarification should be sought from the intended users. Answers to these two questions could help us understand the domain better. Some of these answers can then guide us in feature engineering.

1) Try to understand how this task is done currently. If the users do not already have a machine learning model, do they do it through a rule-based system or manual calculations? Explore if some of these rules and calculations can be used for creating features for the machine learning model.

2) What are the things that could impact the dependent variable positively or negatively?

Let's take a look at a practical example of the hotel industry. Hotels operate on a check-in date basis. Guests reserve their stay in a hotel before staying. The day when they arrive at the hotel is called the "check-in" date. A revenue manager in a hotel might be interested to know in advance, how many rooms

will be occupied by guests in total for a specific check-in date. This will help him/her to price the rooms in a way to maximize profit.

As a machine learning engineer, our goal will be to develop a model that predicts hotel occupancy for a specific future day. Machine learning engineers who don't know how the hotel industry operates will believe it is a univariate time series forecast and will develop a forecasting model. In this case, if we ask the revenue manager our 2 questions, it will look like

1) How are you forecasting occupancy currently, as you do not have a model? Even if it's gut feeling, what factors do you consider for arriving at a conclusion?

2) What are the factors that could result in increased or decreased occupancy for a specific check-in date?

Some possible answers to the first question could be that they forecast occupancy by looking at demand for the particular check-in date so far, and demand for the room on the same check-in date in the previous year. This could suggest that there is seasonality in hotel demand and there is an effect from demand so far for the check-in date.

For the second question, hotel occupancy for a specific date decrease, if competing hotels reduce their price for the same check-in date. If there are more negative reviews for the hotel, in comparison to competing hotels, then a smaller number of guests will stay in the hotel. Price affects hotel occupancy inversely. If price of room is very high, few guests will come to stay, whereas if price is very low, there might not be any vacant rooms in the hotel. Online reputation directly affects the hotel reservations. Demand can also be affected if holidays or events are happening close to the check-in date.

2.2.2 Literature Review

Quite often machine learning engineers are required to solve business problems in new domains where they lack subject matter expertise. In some of these cases, subject matter experts (SMEs) are not accessible or access to SMEs is limited. Reasons could be any, such as workload of their own, SMEs being C-level executives and having limited time, etc. In such situations, it can be helpful to read research papers and books on the subject matter. In particular, try to understand the type of features used and the way data was organized for modeling for similar machine learning problems.

Let's take the example of hotel industry booking prediction for check-in date. There is an argument put forth and proven [1] by data that the social media reputation of a hotel has an impact on hotel room reservations, as against competitor hotels. A simple search on google scholar can help us find useful research papers that can help us improve feature engineering for the project.

2.3 References

[1] Chris K. Anderson, The Impact of Social Media on Lodging Performance, 2012, https://hdl.handle.net/1813/71194

Chapter 3: EDA Feature Engineering

3.1 Introduction

Exploratory data analysis (EDA) can help us uncover hidden patterns in the data. We can in turn represent the discovered patterns into features in the dataset. In order to do this, we should approach any type of EDA with 3 questions. What, So what, and now what.

1) What do the patterns in this visualization say?

2) So, what does this pattern say about my problem statement and how it can affect my problem statement?

3) Now what should I do to inculcate the patterns discovered during EDA? Should I include this information as a new feature or should I perform data cleaning?

Let's take a look at our datasets.

3.2 Car Sales

Dependent variable for this dataset is numerical and the modeling problem is regression. Let's explore numerical features through a pair plot in figure 3.2.1 and ask the 3 questions.

Question 1: What do the patterns in this visualization say?

It appears that there is a strong relationship between the selling price of the car and the year since when the car has been used. This is also observed for how many kilometers the car has been used for. Interestingly, the car price has a not so strong impact on the number of seats.

Max_power and engine have a strong relationship with the selling price. Mileage also has some noticeable patterns.

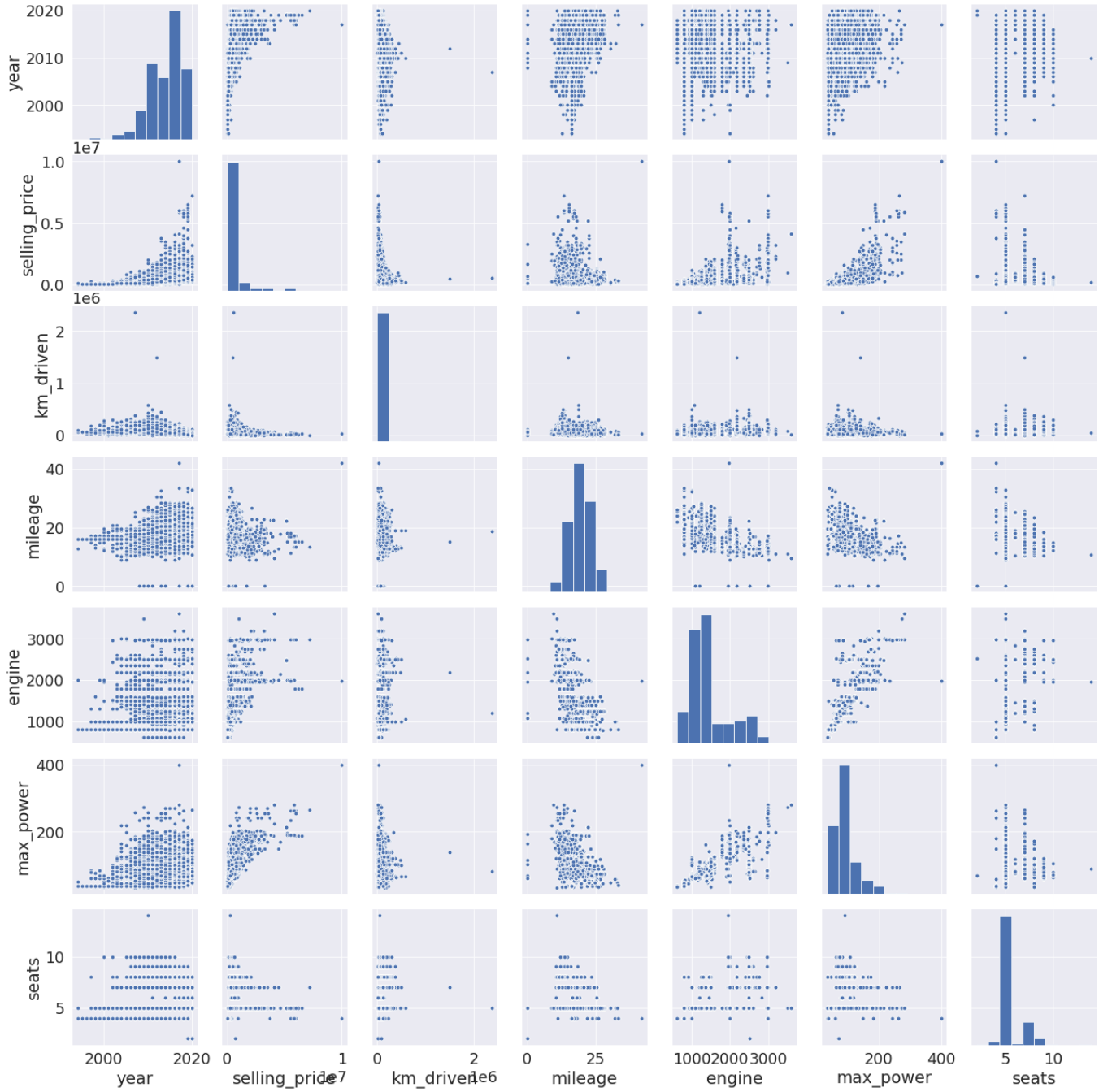


Figure 3.2.1: pair plot of the dependent variable with numerical features

Question 2: So, what does this pattern say about my problem statement, and how it can affect my problem statement?

Let's confirm what we saw in the first plot by quantifying the extent of the relationship by checking the correlation heat map in figure 3.2.2.

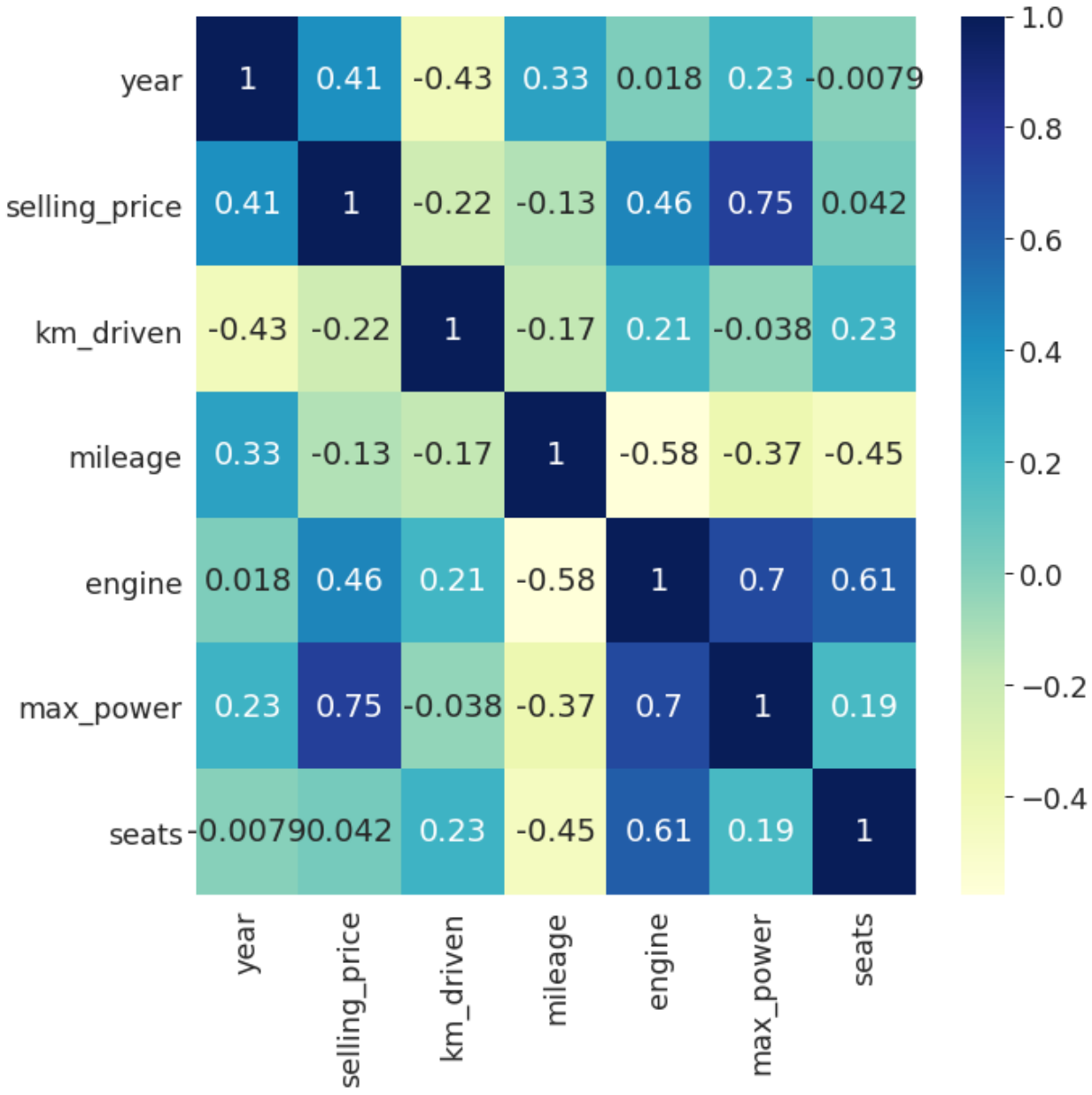


Figure 3.2.2: Correlation heatmap of numerical features with the selling price.

We can see that max_power has the strongest correlation with the selling price, followed by the engine. Mileage on the other hand has a mild negative correlation with the selling price of used cars.

The selling price has a positive correlation with the year. If the year is higher, then it can be sold for a higher price. In other words, if the car is new, it can be sold for a higher price. On the other hand, car price has a negative relationship with the number of kilometers driven. If the car has been driven for long distances, it has been through wear and tear. It is likely to fetch less price. Finally, the number of seats has a weak positive correlation with the selling price. People would like to have a car with a higher number of seats, but not so much.

Question 3: Now what should I do to inculcate the patterns discovered during EDA? Should I include this information as a new feature or should I perform data cleaning?

For the features which have a high correlation, we can check if the original features give better performance or if we can get better performance by using higher order features of these features. This is even more applicable for seats. We will need to check if there is any higher order feature for seats that can help us get better performance.

If we go by common knowledge about cars, the sports cars are sold at the highest price, although they have the lowest number of seats. This is followed by SUV cars, which have a higher number of seats but fetch higher prices. However, cars that have 4-5 seats are used by middle class people and have a relatively lower selling price. Hence, there is a non-linear relationship between the number of seats and car price. This could also be applicable for used cars. Higher order feature engineering might be able to uncover this non-linear relationship. We can also consider the number of seats as ordinal and try higher order ordinal features to see if it works better. Now, let's also explore categorical features.

Let's start with the '*owner*' feature in figure 3.2.3. This feature has values representing how many people have previously owned the car. "*First Owner*" suggests that the car is owned by a first-time buyer, whereas "Second Owner" means the car has been owned by 2 owners, including the current owner. In our dataset, the first owner and second owner are 65.9% and 25.5% respectively. They constitute the majority group.

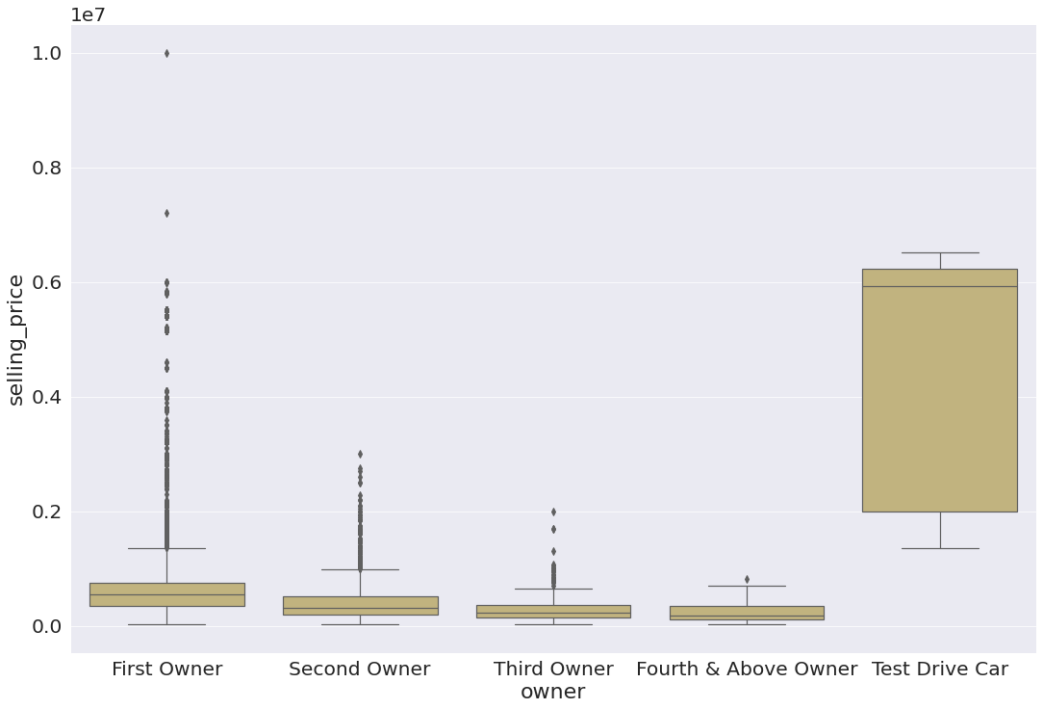


Figure 3.2.3: Boxplot of selling_price for each type of '*owner*'

Question 1: What do the patterns in this visualization say?

There is a huge degree of variation in prices at which used cars are sold for the first owner and second owners, as evident from the number of outliers in the boxplot for these 2 categories. This is also true for the third owner. Test drive cars and cars which have been owned 4 times have relatively stable prices, as they do not have any outliers.

Cars that have been owned a fourth time or above, fetch the lowest prices as evidenced by the average selling price. This can be seen in figure 3.2.4.

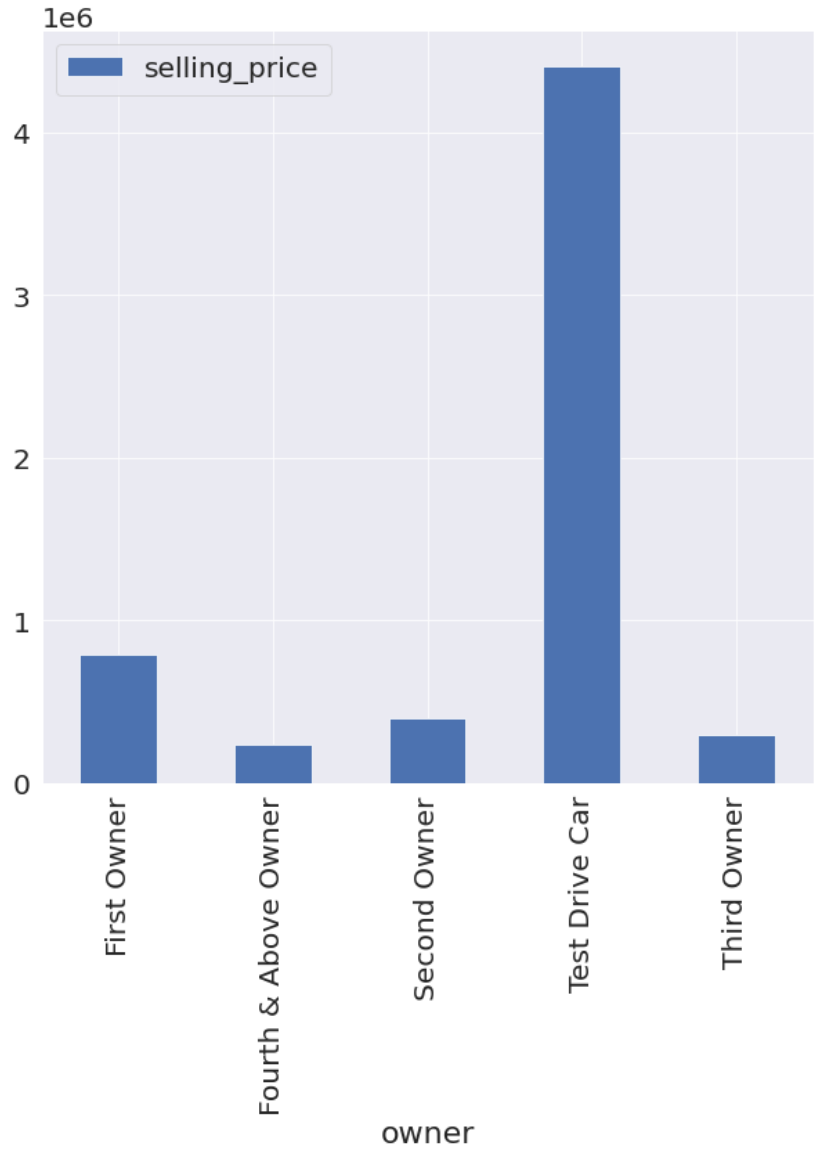


Figure 3.2.4: Average selling price by type of owner.

Question 2: So, what does this pattern say about my problem statement and how it can affect my problem statement?

It seems that first, second, and third-owner cars have lower average selling prices than test-driving cars. However, many outliers have very high prices. We need to account for it.

Question 3: Now what should I do to inculcate the patterns discovered during EDA? Should I include this information as a new feature or should I perform data cleaning?

As this is common knowledge, we know that sports cars and SUVs have higher prices than other cars. Let's verify this by checking the car brand names from the '*brand*' column for cars that have above 90 percentiles selling_price, through a word cloud. Figure 3.2.5 has the word cloud for cars with higher percentile selling price.

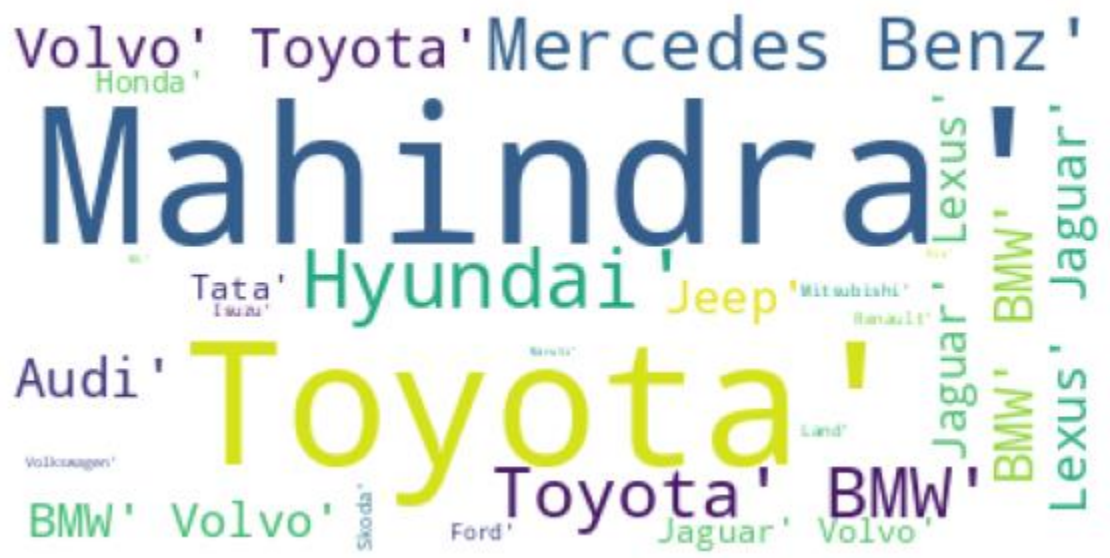


Figure 3.2.5: Brand names with above 90 percentiles selling_price.

Most of the cars for which selling_price is outliers, are considered as premium and luxury cars in the Indian market. If we could distinguish these cars from others while modeling, it can give us comparatively better performance. We can represent this information as a feature in 2 different ways. Firstly, we can create a binary 1|0 feature that represents these car brands as an additional feature. Secondly, we can create a higher order feature, which will have the average selling price for each car model or brand.

3.3 Coupon Recommendation

Most of the features in this dataset are categorical or ordinal. The dependent variable in this dataset is binary 1|0, as the nature of the problem is classification. We created another column to represent the dependent variable '*couponstatus*' with descriptive values '*accepted*' and '*not accepted*' for ease of understanding. We explored the relationship between different classes of the dependent variable and different categories of features. Crosstab function in pandas library are used for inspecting the association of different classes with different categories.

During our investigation, we found that different categories within the same feature inclined against different classes to a noticeable extent. These features are '*weather*', '*coupon*', '*maritalStatus*', '*occupation*', and '*CoffeeHouse*'. Let's observe each of these features one by one and seek answers to 3 'What' questions.

Question 1: What do the patterns in this visualization say?

couponstatus weather	accepted	not accepted
Rainy	46.21	53.79
Snowy	47.63	52.37
Sunny	59.50	40.50

Table 3.3.1: Crosstab of '*couponstatus*' with '*weather*'

For the '*weather*' feature, more people accept the coupon if it is sunny weather. Let's now explore the relationship between the type of coupon and its acceptance in table 3.3.2 below.

couponstatus coupon	accepted	not accepted
Bar	41.19	58.81
Carry out & Take away	73.77	26.23
Coffee House	49.63	50.37
Restaurant(20-50)	44.60	55.40
Restaurant(<20)	70.90	29.10

Table 3.3.2: crosstab of '*couponstatus*' with the type of coupon offered in the '*coupon*' feature

More people accept coupons if it is a coupon for 'Carry out & Take away' or if it is a restaurant coupon of value less than $20. Ironically, fewer people accepted a higher-value restaurant coupon which is between $20-$50. Let's now explore the relationship between marital status and the acceptance rate of coupons in table 3.3.3 below.

couponstatus maritalStatus	accepted	not accepted
Divorced	52.58	47.42
Married partner	54.30	45.70
Single	61.03	38.97
Unmarried partner	55.57	44.43
Widowed	47.22	52.78

Table 3.3.3: crosstab of '*couponstatus*' with the marital status

Everyone, except widowed individuals accepted more coupons than they rejected. Single people have the highest likelihood of accepting coupons against other groups. Let's explore the relationship between different occupations and the likelihood of accepting coupons in table 3.3.4 below.

couponstatus occupation	accepted	not accepted
Architecture & Engineering	63.43	36.57
Arts Design Entertainment Sports & Media	53.37	46.63
Building & Grounds Cleaning & Maintenance	59.09	40.91
Business & Financial	56.98	43.02
Community & Social Services	49.77	50.23
Computer & Mathematical	56.36	43.64
Construction & Extraction	68.83	31.17
Education&Training&Library	51.35	48.65
Farming Fishing & Forestry	53.49	46.51
Food Preparation & Serving Related	56.52	43.48
Healthcare Practitioners & Technical	71.62	28.38
Healthcare Support	69.83	30.17
Installation Maintenance & Repair	53.38	46.62
Legal	47.03	52.97
Life Physical Social Science	57.65	42.35
Management	59.33	40.67
Office & Administrative Support	60.29	39.71
Personal Care & Service	55.56	44.44
Production Occupations	70.45	29.55
Protective Service	64.57	35.43
Retired	45.67	54.33
Sales & Related	56.34	43.66
Student	60.86	39.14
Transportation & Material Moving	59.63	40.37
Unemployed	54.58	45.42

Table 3.3.4: crosstab of '*couponstatus*' with the type of occupation of person

People who work in architecture and construction related professions, healthcare-related professions, as well as production occupations accepted coupons more often. However, people who are retired, work in the legal profession, or work for the community and social services more often did not accept coupons.

Let us look at the combined effect of all these features through a correspondence analysis plot and try to answer the second question.

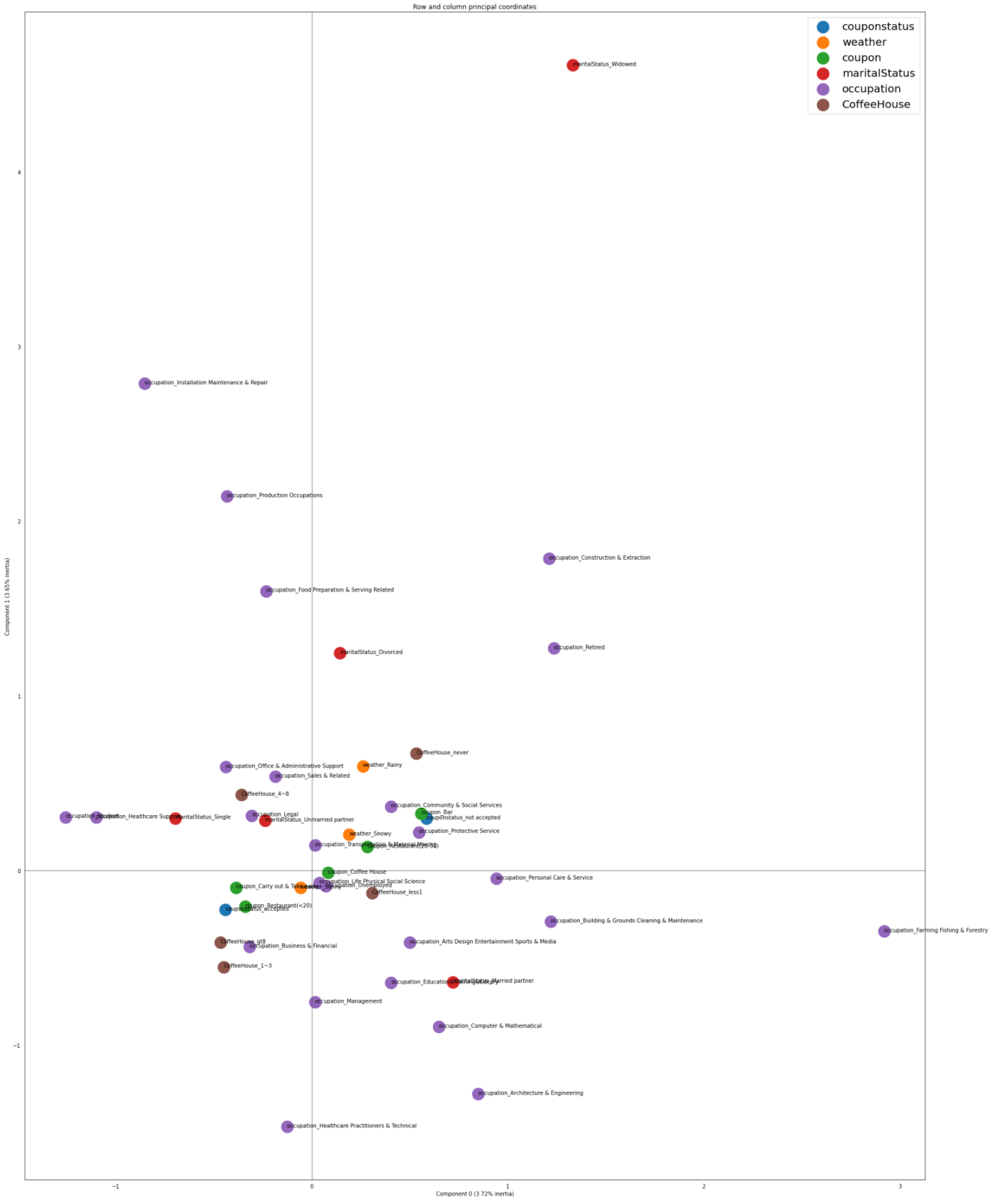


Figure 3.3.1: Correspondence analysis of features with coupon acceptance.

Question 2: So, what does this pattern say about my problem statement and how it can affect my problem statement?

We used multiple features together in figure 3.3.1 for correspondence analysis. We will zoom into different quadrants to understand the plot better. The bottom left and top right quadrants have the coupon 'accepted' and 'not accepted' values respectively. Let us look at figures 3.3.2 and 3.3.3 respectively to understand each of these quadrants.

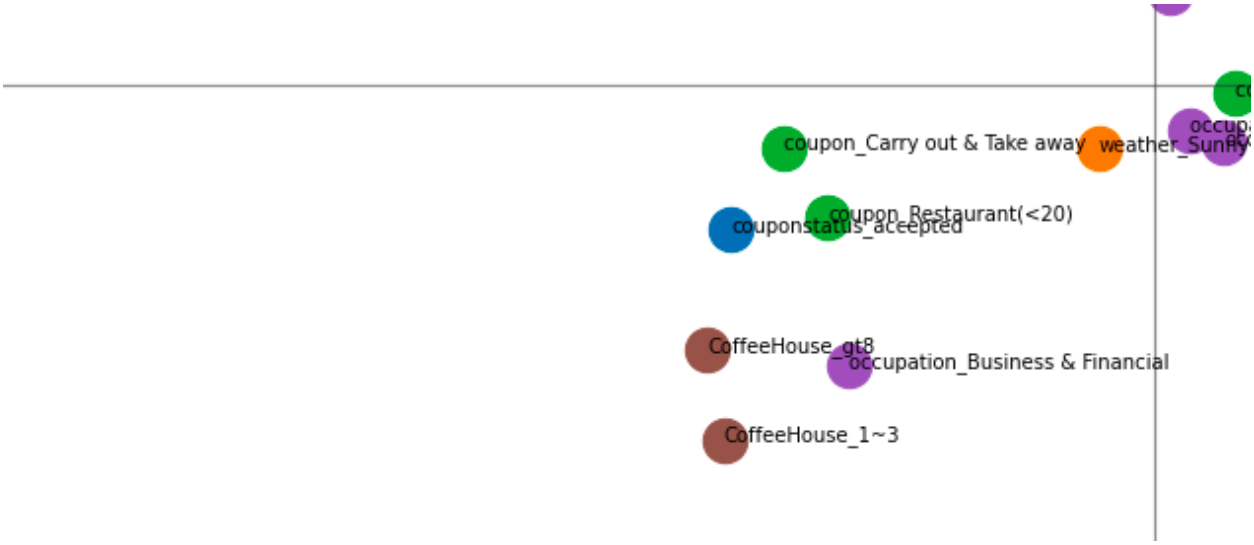


Figure 3.3.2: Bottom left quadrant of correspondence analysis with *couponstatus_accepted*.

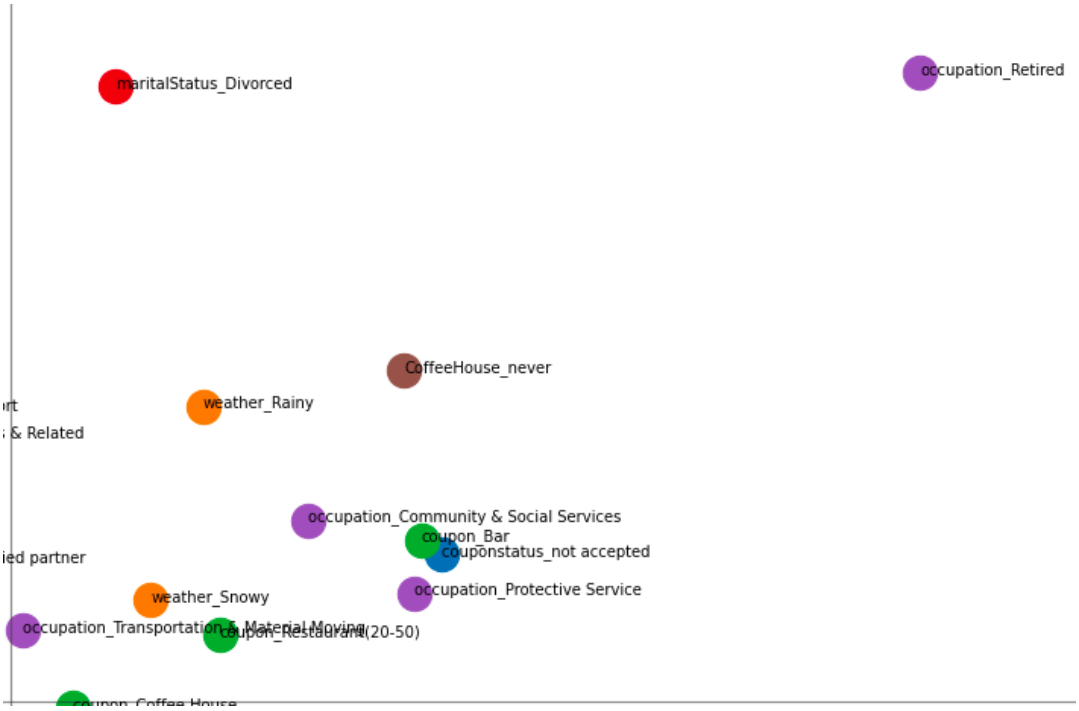


Figure 3.3.3: Top right quadrant of correspondence analysis with *couponstatus_not accepted*.

We can use correspondence analysis, by looking at figure 3.3.2, from the proximity of people who accepted coupons with attributes to make judgments about patterns in the data. People who accepted coupons were mostly those who were offered to carry out & takeaway coupons. Restaurant coupons of value less than $20 were also accepted. When the weather was sunny, more people accepted coupons. Those who visit coffee houses either 1-3 times or more than 8 times a month also accepted more coupons than rejected. People who identified their profession as business & financial accepted more coupons than rejected.

Similarly, from figure 3.3.3, by looking at the proximity of people who did not accept coupons, we can see that people who work at community & social services or protective services, people to whom either bar coupons were given or restaurant coupons of $20-$50 value and people who never went to the coffee house were most likely to not accept the coupon. People also did not accept coupons when the weather was rainy or snowy. All of these patterns match with descriptive statistics, except for people who work for protective services, for whom only 35.43% of people did not accept and the rest of the majority accepted coupons.

If we can somehow include the patterns we discovered through EDA as features in our model, we might be able to get better performance. For this, let's find the answer to question 3.

Question 3: Now what should I do to inculcate the patterns discovered during EDA? Should I include this information as a new feature or should I perform data cleaning?

We created 2 features based on combined information from correspondence analysis and descriptive statistics. '*accepted_coupon*' feature has a binary coded 1|0 value, based on information from correspondence analysis.

The second feature '*rejected_coupon*' has a 1|0 value, based on the proximity of attributes for people who did not accept coupons in the correspondence analysis plot, except for people who work for protective services, as it contradicted descriptive statistics.

For these 2 created features, we performed a chi-square test to validate the relationship between each created feature and the dependent variable. We found that the p-value of the test in both cases was below 0.05, which proved the validity of the relationship and the usefulness of the created features.

3.4 Conclusion

Before starting the actual machine learning model training, we should spend time on understanding the data and features. We can use EDA for identifying interesting patterns in the data. Some of these patterns can be turned into useful features for the model. In some cases, the original dataset has limited number of features and not much can be done using these features. EDA assisted feature engineering can be very useful in these situations for creating a dataset with information rich features.

A secondary usefulness of EDA is that we discover many anomalies in the dataset. Upon verifying with the domain knowledge experts, our understanding of the domain can be improved.

One caveat of EDA assisted feature engineering is that we need to be careful while making deductions from EDA and using the insights in feature engineering. If the person who derived the insights is a subject matter expert, then we have a better chance of success. However, if the machine learning engineer does not have expertise in the domain, then before finalizing conclusions from EDA, SME help should be sought to check and confirm the derived insights. Even so, we should use more than one visualization before concluding with the insights to create a new feature.

Chapter 4: Higher Order Feature Engineering

Features as they exist in their natural form might not always be useful for a machine learning model to learn. In some cases, we need to change the functional form, such as taking the log of the feature to help the model learn. In some cases, we need to make transformations so that model can process and use the feature. For example, label encoding for categorical features for random forest model, and dummy encoding for linear models. In some other cases, we need to enrich features by including information from dependent variables through methods such as mean encoding.

Certain feature engineering can be done on the entire dataset. Whereas, some other type of feature engineering is performed on cross-validated training data and then applied to test data. This is done to avoid overfitting and data leakage. There are 3 types of features in general. These are categorical, ordinal, and numeric. We will learn about higher order feature engineering for these 3 feature types in this chapter.

4.1 Engineering Categorical Features

A categorical feature or nominal feature has multiple unique categories, where there is no order of importance for one category against the other. An example can be different departments in a company, such as finance, human resource, sales, and marketing. Categorical features cannot be introduced to machine learning models as they are. Instead, they should be preprocessed and converted into a format that can be understood by the model. These techniques are called as ‘encodings’. There are many different types of encodings available. We will now look at different types of encodings for categorical features.

4.1.1 Dummy Encoding or One-Hot Encoding

Linear algorithms cannot learn from categorical features as it is. Instead, these features need to be converted to a one-hot encoded format for the model to be able to learn. This is especially useful in the event of auto-correlation, where we can create seasonality related dummy variables to treat auto-correlation.

Let's take the example of the feature 'weather' from the coupon recommendation dataset. It has 3 categories, namely 'Sunny', 'Rainy', and 'Snowy'. We can use the Pandas' function `get_dummies` to create dummy encoded variables. Below is what the output will look like.

weather_Rainy	weather_Snowy	weather_Sunny
0	0	1
0	0	1
0	0	1
0	0	1
0	0	1

While creating dummy encoding, we should be careful about the 'dummy variable trap'. This is a scenario when variables are highly correlated to each other. For linear models, it can be problematic and could lead to multicollinearity. To mitigate this, we can drop one of the columns. In Pandas, we can do this by changing the parameter in `pd.get_dummies` from `drop_first=False`, to `drop_first=True`.

4.1.2 Label Encoding

Tree-based algorithms can learn from categorical features without having to create dummy features. Tree models instead require categorical features to be represented as labels. In this method, a unique number is assigned to each category. The numbers assigned to each category are to distinguish from other categories. These numbers do not represent rank in any order of importance or usefulness.

We can use the `LabelEncoder` function from the Sklearn library to convert categorical features into label-encoded features. Below is how the ‘time’ feature will look, before and after label encoding.

time	timeLabelEncoded
2PM	2
10AM	0
10AM	0
10AM	0
2PM	2
2PM	2
2PM	2
2PM	2
6PM	3
6PM	3

4.1.3 Count, and Percentage Encoding

In count encoding, we replace the respective categories with their count of occurrence. In percent encoding, we replace categories with percentages. It can be used for both linear and tree-based algorithms. If a specific category is present more often than others, in such a situation, count replacement for the category could become an outlier. In this situation, we can perform log transformation, as it will smoothen the effect of the outlier. Also, count or percentage encoding should be done based on the count of categories in training data only and not on the entire data set. As otherwise it will lead to data leakage and overfitting.

Below is the encoding for the ‘passanger’ feature in the coupon recommendation data set with count and percentage.

passanger	passanger_countEncoded	passanger_percentEncoded
Alone	6969	57.70
Friend(s)	3148	26.06
Friend(s)	3148	26.06
Friend(s)	3148	26.06
Friend(s)	3148	26.06

4.1.4 Encoding by Rank of Counts

One of the problems with count encoding is that if a specific category is the present majority of the time, it can make the counts look skewed. One way to fix this problem is by performing log transformation. We can mitigate this challenge also by ranking categories based on the count for the categories and replacing categories with the rank. We first take the count of each category in the feature. Categories are sorted in the order of their counts in ascending order. The category which has the lowest count is given value 1. The count for other categories in ascending order is incremented by 1. These encodings should be developed only from training data.

One additional advantage of the rank of counts encoding is that this is useful for both linear and non-linear models.

For the 'passanger' feature in the coupon recommendation data set, below is what the rank of counts will look like.

passanger	passanger_countRankEncoded
Alone	4
Friend(s)	3
Friend(s)	3
Friend(s)	3
Friend(s)	3

4.1.5 Target Encoding

We can take summary statistics for categories against the dependent variable and use it to replace the categories. The most commonly used summary statistics are the mean value of the dependent variable. We can calculate the mean value of the dependent variable for each category and replace these mean values with actual categories. Mean encodings can be applied for both regression and classification problems.

For regression problems, we can also use quantiles, such as the 25th percentile, median, and 75th percentile instead of the mean. We can also use the standard deviation for each category, to replace each category. If data in the dependent variable has outliers, we can instead convert the dependent variable to a

log scale and then calculate desired summary statistics for each category. Below is an example of mean encoding for the coupon recommendation dataset for the ‘occupation’ feature.

occupation	occupation_MeanEncoded
Architecture & Engineering	0.634286
Architecture & Engineering	0.634286
Architecture & Engineering	0.634286
Architecture & Engineering	0.634286
Architecture & Engineering	0.634286

This encoding is useful when we have too many categories in the categorical feature. We should however calculate these encodings only on the training data. After calculating the encodings, we can apply these to test data and validation data. This will prevent overfitting.

For categorical features with high cardinality, for the last 3 encodings discussed, it can be possible that a category is present only in the test data and validation data, whereas it is absent in training data. In such a case, encoding for the categorical feature will not be representative of all data. We can drop the encoding feature of high cardinality features in such a case. Although not an ideal solution, we can also include encoding from training data of another cross-validation sample for the missed categories, as a workaround. It will ensure that we have encoding value for the missing category, and at the same time, there is minimal data leakage, as we are only taking encoding for the missing category and from training data of another cross-validation sample. Including encoding from the entire dataset should always be avoided, as it will lead to data leakage and overfitting. In rare occurrences when the specific category for the feature is only present in test data and not in training data of any cross-validation sample, we may obtain encoding for the specific category from test data. If there are too many such categories in a feature that are not present in training data, we should avoid using count, percent, rank percent, and target encoding for such features.

4.2 Engineering Ordinal Features

Ordinal feature, just like the categorical feature has multiple categories. The difference between the both type of feature is that in the case of ordinal feature, categories follow a specific order. For example, an ordinal feature 'age group' can have the age of individuals recorded as categories such as 'kid', 'teenager', 'adult', and 'elderly'. These categories are reflective of the age of individuals from lower to higher value

incrementally. In this case, the 'age group' can be considered an ordinal feature. Let’s discuss the different types of encoding possible for ordinal features.

Let's consider the ordinal feature '*income*' in the coupon recommendation data set. It has 9 levels, starting from 0 to more than 100000. Each category is a range with lower and upper ranges. The difference between the upper and lower ranges of each category is 12499. The difference between the lower range of the next category and the upper range of the previous category is 1.

Just like categorical features, ordinal features should be presented as encodings.

4.2.1 Rank Encoding

This is the most simplistic feature encoding for ordinal features. We rank categories in such a way that the category of least value is given a value of 1 and for other categories, we incrementally increase the label by 1. In the example given for '*income*', the category "Less than $12500" will be given value as 1, for '$12500 - $24999' it will be 2, and so on.

Below is a snapshot of the rank-encoded representation of the income feature.

income	income_Ranking
6250074999 –	6
6250074999 –	6
6250074999 –	6
6250074999 –	6
6250074999 –	6

4.2.2 Polynomial Encoding

It searches for 3 different types of trends in the feature, based on which it creates contrast encodings. It fits a regression line using mean, quadratic parabola, and a cubic term to produce linear, quadratic, and cubic encodings. If the ordinal feature has N number of categories, polynomial encodings produce N-1 polynomial encodings. Below is a snapshot of polynomial encoded features for ‘income’.

income	Poly_incm_0	Poly_incm_1	Poly_incm_2	Poly_incm_3	Poly_incm_4	Poly_incm_5	Poly_incm_6	Poly_incm_7
6250074999 –	-0.516398	0.531816	-0.444949	0.312893	-0.1849	0.089893	-0.034139	0.008815
6250074999 –	-0.516398	0.531816	-0.444949	0.312893	-0.1849	0.089893	-0.034139	0.008815
6250074999 –	-0.516398	0.531816	-0.444949	0.312893	-0.1849	0.089893	-0.034139	0.008815
6250074999 –	-0.516398	0.531816	-0.444949	0.312893	-0.1849	0.089893	-0.034139	0.008815
6250074999 –	-0.516398	0.531816	-0.444949	0.312893	-0.1849	0.089893	-0.034139	0.008815

4.2.3 Backward Difference Encoding

In this method, the mean of the dependent variable for one level of the categorical variable is compared to the mean of the dependent variable for the prior adjacent level. It produces N-1 encoded features for N categories in an ordinal feature. Below is a snapshot of backward difference encoding for ‘income’.

income	BackDif_incm_0	BackDif_incm_1	BackDif_incm_2	BackDif_incm_3	BackDif_incm_4	BackDif_incm_5	BackDif_incm_6	BackDif_incm_7
6250074999 –	-0.888889	-0.777778	-0.666667	-0.555556	-0.444444	-0.333333	-0.222222	-0.111111
6250074999 –	-0.888889	-0.777778	-0.666667	-0.555556	-0.444444	-0.333333	-0.222222	-0.111111
6250074999 –	-0.888889	-0.777778	-0.666667	-0.555556	-0.444444	-0.333333	-0.222222	-0.111111
6250074999 –	-0.888889	-0.777778	-0.666667	-0.555556	-0.444444	-0.333333	-0.222222	-0.111111
6250074999 –	-0.888889	-0.777778	-0.666667	-0.555556	-0.444444	-0.333333	-0.222222	-0.111111

4.3 Engineering Numerical Features

Values of a continuous or numeric feature exist within a range of lower and higher points. It can have unlimited values between the lowest and highest points. Some examples are prices, distance, weight, height, etc. Most of the higher order feature engineering for numerical features are transformations and are useful for linear models. Linear models expect the relationship between the feature and dependent variable as linear and residuals to have homoscedasticity. If this is not met, transformations can help.

For classification problems, we can select all higher order numerical features, for whom F-test is significant. In the case of numerical features, for regression problems, we can rank the transformed features based on correlation with the dependent variable. We can select the higher order feature which has the highest correlation with the dependent variable. In some cases, we can also discuss why a certain

higher order feature has a higher correlation than others by looking at the data distribution in the original feature. This principle is applicable for all types of higher order features, except for ‘Binning’, as it produces a categorical feature.

4.3.1 Binning

It is the process of creating a categorical feature from a numerical feature. We can for example use quartiles 0-25 percentile, 25-50 percentile, 50-75 percentile, and 75th percentile-maximum for creating bins. If a value in the original numeric feature falls under a specific quartile, we can code the value amongst the 4 quartiles in the categorical feature. Similarly, we can also divide the data into percentile bins and code the categorical values. Binning helps in finding definite structure in numerical features, at the cost of removing nuances.

4.3.2 Square and Cube

Square and the cube of the original feature are polynomials. It is helpful to use polynomial features if the feature follows an inverted-U pattern. An inverted-U pattern exists when the dependent variable increases concerning an increase in the independent variable at lower values. However, at higher values of the independent variable, the dependent variable increases at decreasing rate. An example can be wage, as against age. When age increases, wage also increases. After a certain age, the wage doesn't increase and instead decreases. In other words, when the relationship between the feature and the dependent variable is not linear or when the relationship is curvilinear or quadratic, we can use polynomial features. Polynomial features are used mostly as square. Sometimes, a higher order polynomial such as a cube can also be used.

4.3.3 Regression Splines

If we are using a linear model and the evidence suggests that the relationship is nonlinear, it's better to replace the linear model with a polynomial model. However, if the number of polynomial features keeps increasing, it can lead to overfitting. In this type of situation, we can instead use regression splines. It divides data into multiple regions, known as bins, and fits linear or low-degree polynomial models for each bin. Points at which data are separated as bins are called knots. It usually uses a cubic polynomial function, within each region. Using a very high number of knots overfits the model. We should try a different number of knots to identify which one produces the best results.

4.3.4 Square Root and Cube Root

Square root and cube root transformation can help normalize a skewed distribution. It does so by compressing higher values and inflating lower values, resulting in lower values becoming more spread out. This is especially true when the feature has count data and follows a Poisson distribution. Square and cube root transformation could be much closer to gaussian. It can convert a non-linear relationship between 2 variables into a linear relationship. However, we should be careful when applying square root and cube root transformation on features that have negative values. Square root and cube root of negative values are returned as missing values.

For the feature 'CumulativeNumberOfRooms' in the hotel booking demand dataset, the correlation of the original feature with total rooms was 0.80. After square root transformation, it marginally improved to 0.83. The effect of the transformation can be understood better with the help of figure 4.3.4 below.

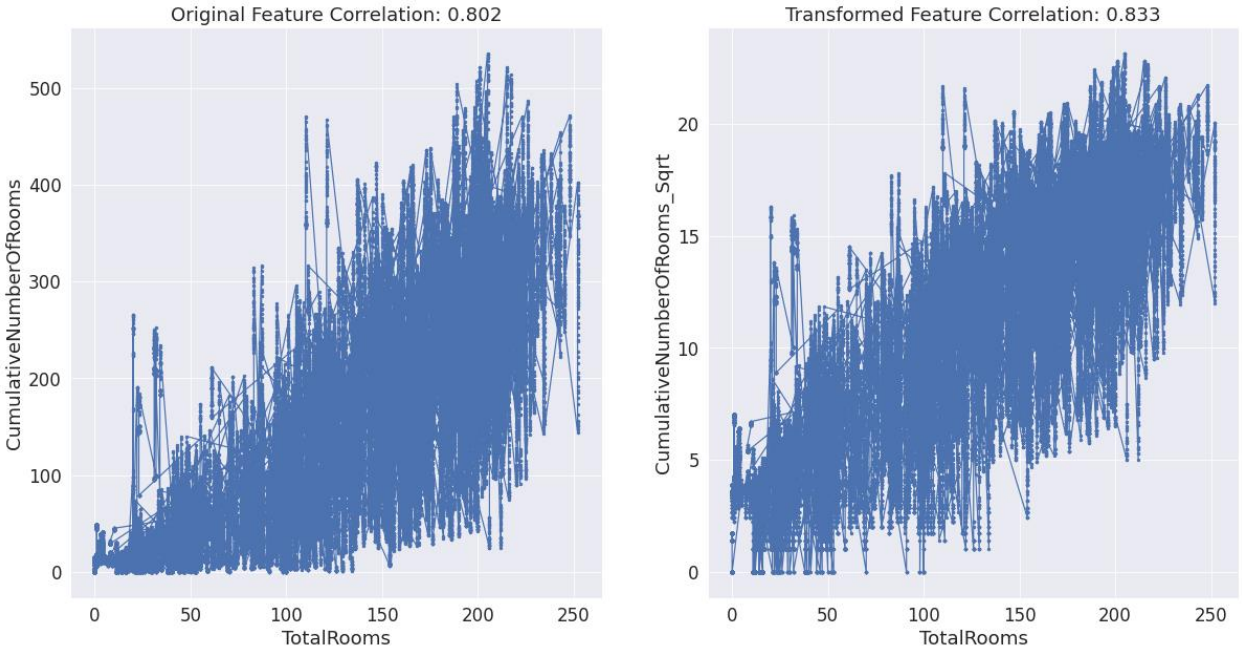


Figure 4.3.4: Scatter plot of TotalRooms with CumulativeNumberOfRooms and the square root of CumulativeNumberOfRooms.

We can infer from the straight line like structure in the second plot that after square root transformation, the feature has a nearly linear relationship with the dependent variable.

4.3.5 Log Transformation

If the feature or the dependent variable has an outlier, log transformation can help subdue the effect of such observations. Just like square root and cube root, log transformation can compress higher values. It does so more aggressively than square root and cube root. This in turn can help models which are sensitive to outliers. For example, linear regression can help achieve normality.

It can also help in converting a non-linear model into a linear model. For models which study the effect of percentage change in the feature on the percentage change in the dependent variable, performing log transformation before modeling can result in a linear model.

Let's consider the feature AdjustedLeadTime_CumulativeRevenue in the hotel room booking dataset. This is an interaction effect feature and a product of AdjustedLeadTime and CumulativeRevenue. Values in this feature are very high. Log transformation was able to subdue the higher values, and as a result correlation for the feature with the dependent variable 'TotalRooms' increased from 0.637 to 0.775.

We can also infer from plot 4.3.5 that the log-transformed feature has a nearly linear relationship with the dependent variable.

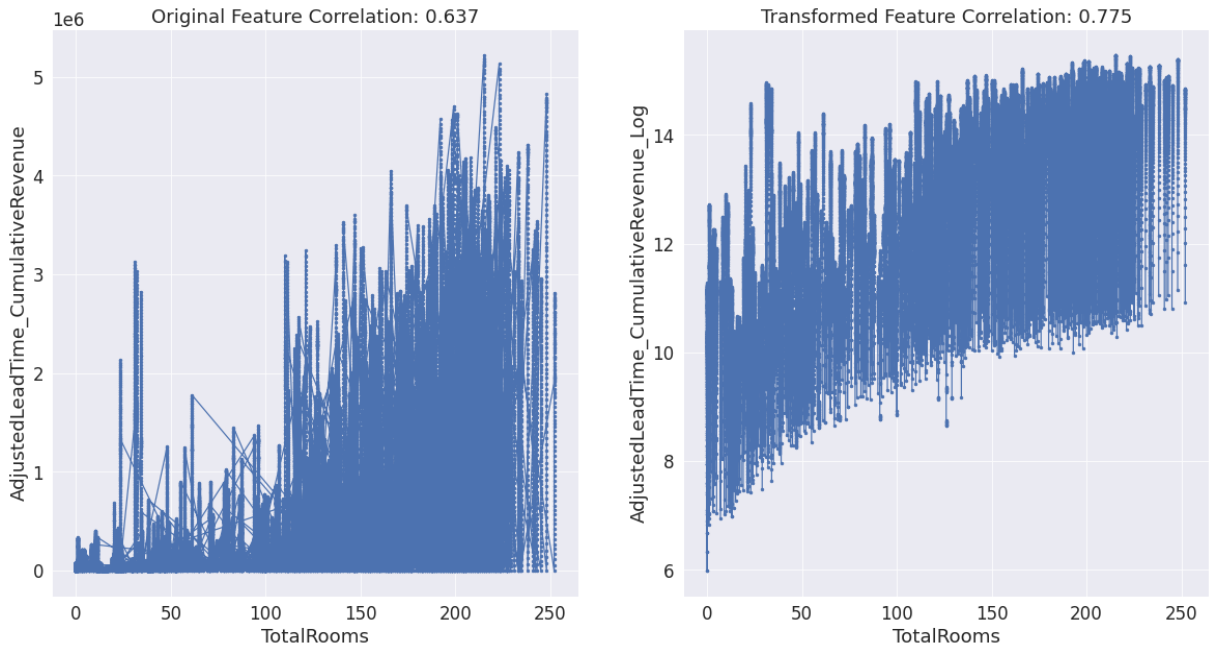


Figure 4.3.5: Scatter plot of TotalRooms with AdjustedLeadTime_CumulativeRevenue and log of AdjustedLeadTime_CumulativeRevenue.

4.3.6 Standardization and Normalization

Linear models use gradient descent for converging. For gradient descent, it helps if all the features are on the same scale. If the features are not in the same scale, we can take use 2 methods to convert features in different scale of measurement, into same scale of measurement. These two methods are called as scaling and standardization. Scaling is also known as normalization, and standardization, is otherwise known as Z-score.

Normalization transforms all the values in features in the range of 0, and 1. It does so while preserving the shape of data distribution.

Unlike normalization, standardization retains useful information about outliers. For a standardized feature, its mean value becomes 0 and the standard deviation becomes 1, and hence follows a normal distribution. standardization is more applicable in linear algorithms. It is sensitive to outliers and outliers should be treated first, before proceeding with this method.

4.3.7 Box-cox Transformation

Box-cox transformation is done to convert non-normal distribution into a normal distribution. It relies on a parameter λ. If the value of $\lambda=1$, it means no transformation. If $\lambda=0$, it will result in log transformation. If $\lambda=0.5$, it will lead to square root transformation, whereas $\lambda=-1$ will give inverse transformation. It is useful for linear models which require normal distribution. One caveat of this method is that it can only be applied to features that have strictly positive values.

4.3.8 Yeo-Johnson Transformation

Yeo-Johnson technique helps convert skewed data into Gaussian distribution. It can handle zero and negative values, unlike the box-cox transformation.

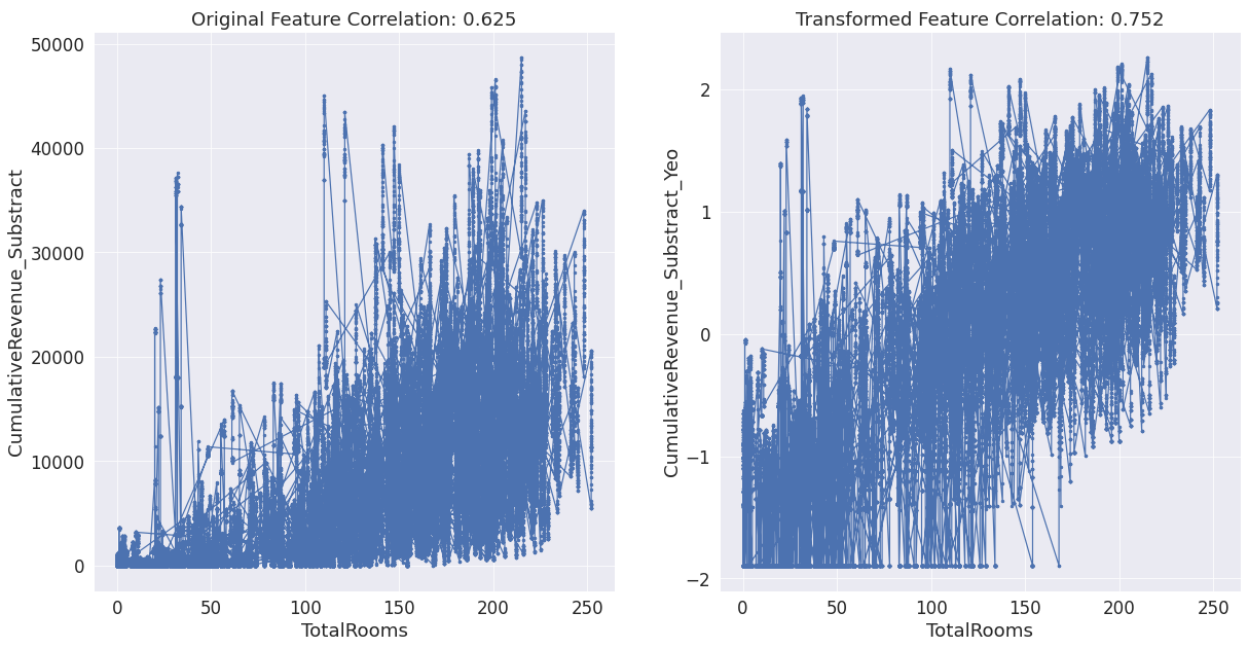


Figure 4.3.8: Scatter plot of TotalRooms with CumulativeRevenue_Substract and yeo-johnson transformation of CumulativeRevenue_Substract.

Let's consider the feature CumulativeRevenue_Substract in the hotel room booking dataset. Yeo-Johnson transformation was able to change the relationship between the feature and dependent variable. As a result of the transformation, the correlation for the feature with the dependent variable 'TotalRooms' increased from 0.625 to 0.752.

We can also infer from plot 4.3.8 that the transformed feature has a nearly linear relationship with the dependent variable.

4.4 Conclusion

For all the different types of encoding and transformation discussed, a big question arises as to which encoding to use for what feature. There are two ways to approach this. The first method is doing the correct transformation based on the nature of the data, aided by domain knowledge. We can choose a specific type of transformation for a feature, based on the nature of the data.

The second method follows the principle of doing the least harm. We should try all encoding and transformation possible for a feature based on its type. The only exceptions should be where it might cause harm. i.e., if the specific type of transformation is not suitable for the feature type or for the

modeling technique. For example, we cannot use higher order feature engineering suitable for numerical features and apply it on categorical features.

We should also consider the suitability of the higher order feature engineering technique for the feature, based on the modeling technique being used. For example, we cannot use label-encoded categorical features in linear models.

We can create multiple higher order features for an original feature. If the new features are beyond the computational capacity, we can select a few features from the list of higher order features. For this, we can use techniques such as correlation and hypothesis testing techniques, namely F-test and Chi-square test. In some cases, we might still end up with more than one type of higher order representation for an original feature. In such situations, we can keep these, as they can help improve the model performance.

Chapter 5: Interaction Effect Feature Engineering

The relationship between the independent variable and the dependent variable is called the main effect. For example, the impact of calorie intake on cholesterol levels could be referred to as the main effect. Some features might interact with each other and can affect each other's values. For example, body weight can impact calorie intake. People with higher body weights tend to consume more calories. Body weight and calorie intake, both can impact cholesterol levels in the body. Interaction effects indicate that a third variable influences the relationship between the dependent variable and the independent variable. The effect of one independent variable on the dependent variable is dependent on the value of another independent variable.

Interaction effects can explain additional variation in the dependent variable, more than what individual features could do alone. Interaction effect exists amongst some independent variables. It is important to identify these interaction effects and represent them as new features. In the next step, we need to test the validity of the interaction effect. If we keep adding interaction effects mechanically amongst all existing features in a dataset, without any backing of domain knowledge or established principles, it will lead to overfitting. In this chapter, we will discuss methods other than domain knowledge for identifying interaction effects.

5.1 Interaction Plot

The interaction plot is an extension of two-way ANOVA, which tests if two factors affect the dependent variable. If the p-value is below 0.05, we will believe that there is a significant interaction effect between the two factors. Before that, let's understand what is one-way and two-way ANOVA.

The one-way ANOVA tests variance in the group means within a sample while considering only one categorical feature. In the case of two-way ANOVA, it tests variance in the group means within a sample while considering levels of two categorical features. It tests the interaction between categorical features, for the continuous dependent variables.

There are two steps for concluding the relationship between two categorical features. In the first step, we perform two-way ANOVA and check the p-value of the interaction effect. If the interaction effect is significant, we perform the second step. In the second step, we plot the relationship with the help of an interaction plot. Interaction plot can help us understand the relationship visually. If the lines in the plot are

parallel, we can conclude that there is no interaction. On the other hand, if lines intersect each other, we can say that there is an interaction among the features. If lines neither intersect nor run parallel, we can say that there is some degree of interaction.

The final decision of accepting or rejecting the presence of interaction effect should be done after checking the p-value of ANOVA. To avoid overfitting, we will perform ANOVA on training data across all cross-validations. If the result is significant across all cross-validations, we will consider it statistically significant and the interaction effect valid.

For the hotel room booking dataset, all the categorical features are derived either from the check-in date, or quartile features from numerical features such as booking trend, and revenue. Hence It will not be meaningful to do ANOVA between the dependent variable and the categorical variables derived from the dependent variable. It will neither be meaningful to perform ANOVA with only categorical variables derived from the seasonality of the date feature.

Let's go through the car sales regression dataset. We will use a few categorical features to explain the interaction plot. We will start with the 'fuel' and 'sellertype' categorical features against the dependent variable 'sellingprice'. Figure 5.1 shows the interaction plot between the features and the dependent variable..

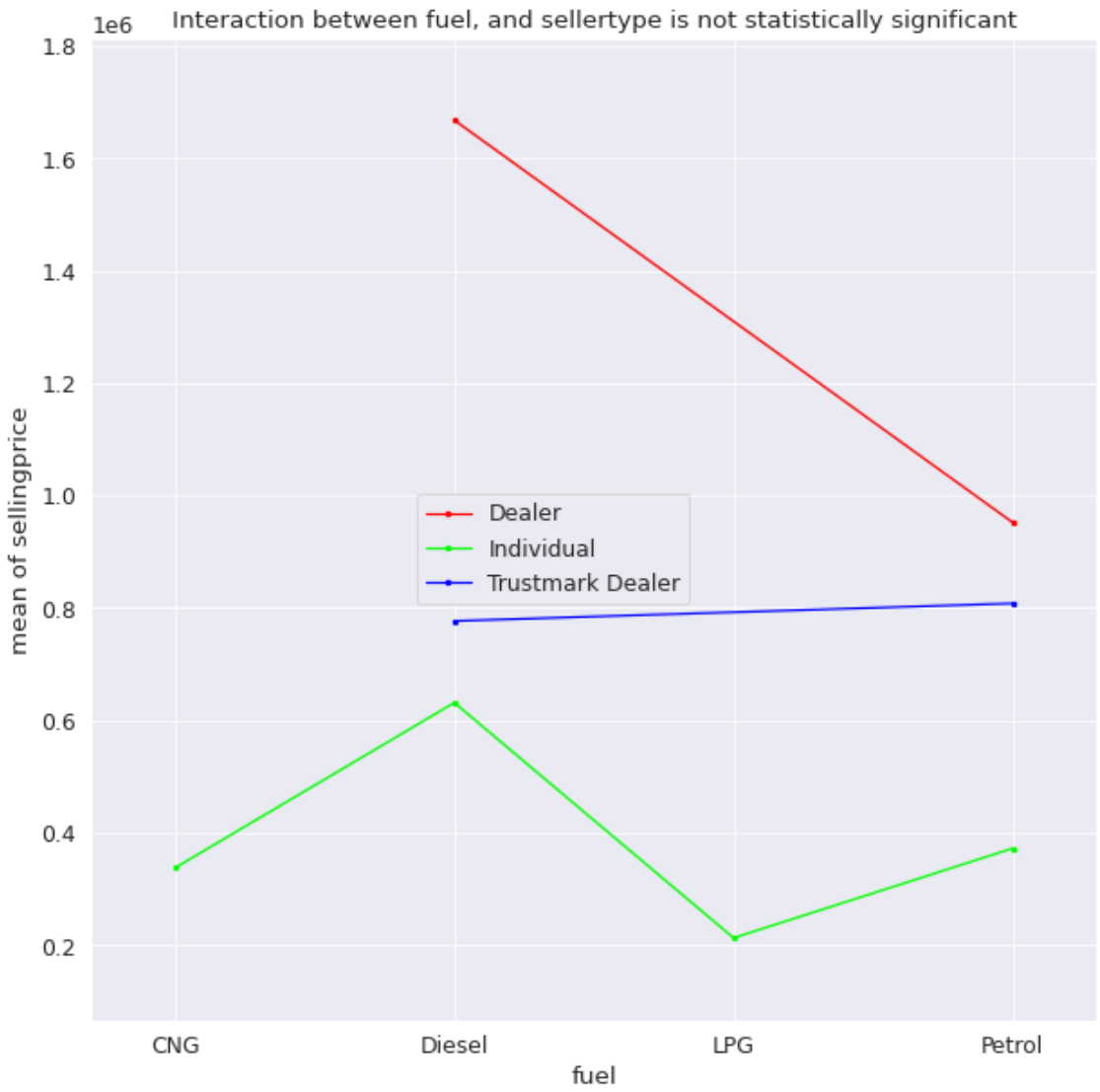


Figure 5.1: Interaction plot of fuel and sellertype against sellingprice

From the graph we can see that dealer car have the highest price, regardless of the fuel type. This is followed by petrol cars sold by Trustmark dealers. LPG, followed by CNG cars has the lowest prices when sold by individual sellers. Although we can infer these relationships from the graph, it was not validated by the p-value of ANOVA for training data across all cross-validation samples. Also, none of the lines in the plot cross each other. Hence, we conclude that the interaction effect is not present between 'fuel' and 'sellertype'.

Now let's look at the relationship between fuel and owner on car selling price in figure 5.2.

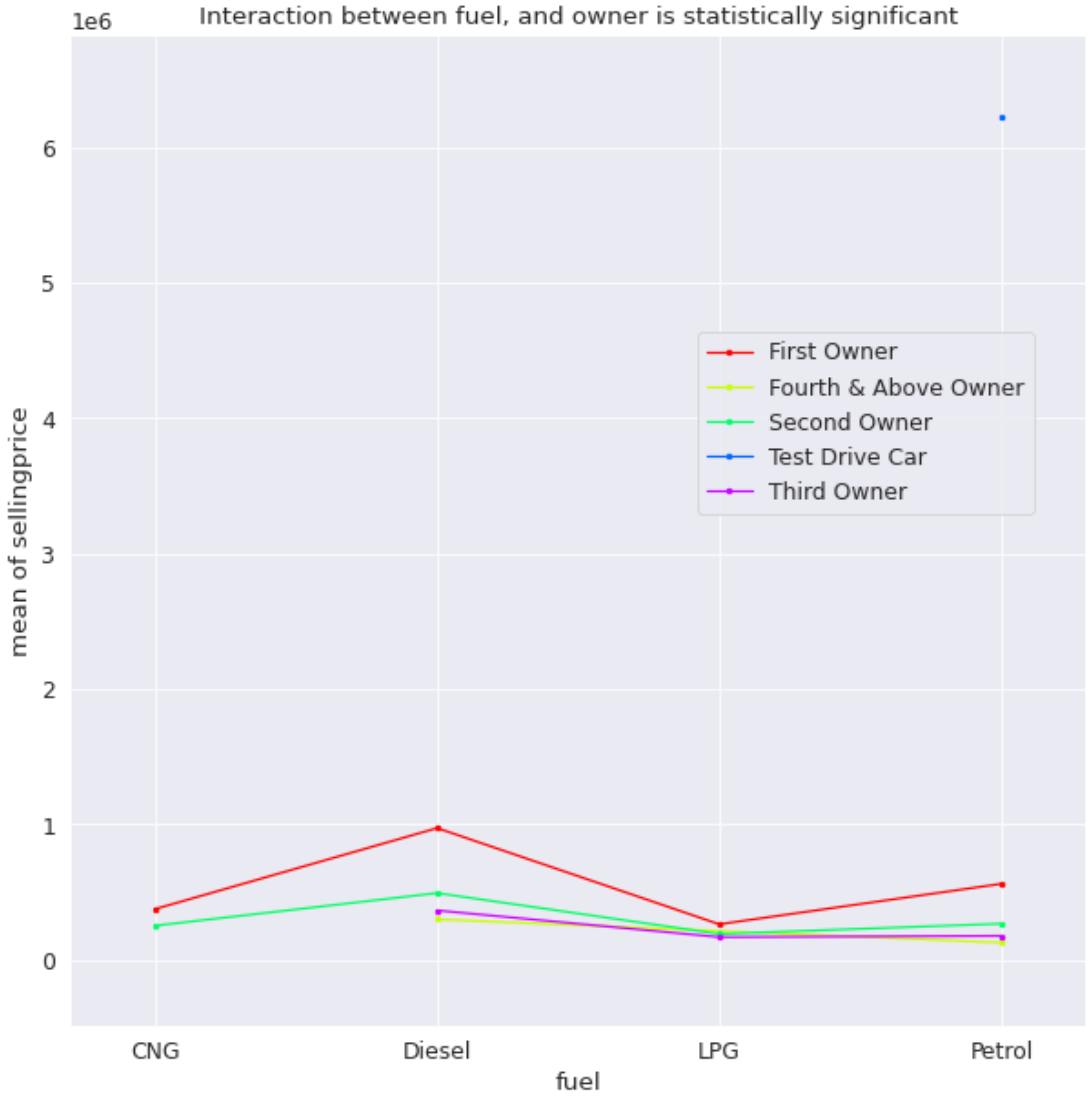


Figure 5.2: Interaction plot of fuel and owner against sellingprice

From figure 5.2 we can see that petrol test drive cars have the highest price. We can infer this from the position of the blue dot at the top right section of the plot. This is followed by diesel cars sold by first owners, which have the second highest price. Petrol cars sold by first owners fetch less price than diesel cars sold by first owners.

Second, third, and fourth owners intersect at LPG for all owner types, which means LPG cars are sold at similar prices, regardless of the number of times it has been sold in the past. The only exception is the first owner, for whom the selling price is relatively higher.

We also tested the relationship with ANOVA for training data across 5 cross-validation samples. In all the samples, the relationship came as significant. We can finally create an interaction effect feature between these two categorical features.

The best way to go about finding the interaction effect through an interaction plot is that first, we should perform ANOVA across different samples of training data in cross-validation. If it came significant in all the cross-validation samples, we can then use an interaction plot to get an intuitive explanation of the type of relationship that exists. We can finally use it for creating interaction effect features.

5.2 SHAP

Shapley additive explanation (SHAP)[1] method can help us detect feature interactions. SHAP is a model explanation technique. For a specific data point, it can explain the extent of importance a feature has on a model for the data point. It uses game theory to quantify the contribution of each feature. Positive and negative SHAP values represent their impact in increasing or decreasing the prediction outcome. SHAP can be extended to find interaction amongst features. It does so by ascertaining the impact of the main effect and interaction effect.

There are three steps for finding the interaction effect using SHAP. The first step is to create a model with available features. The second step is to calculate interaction values using the SHAP python package, for each record, using the model. The result from SHAP will be in the form of a matrix, which has interaction amongst all features, for each record. The third step is to aggregate interaction values across all records to get overall interaction across features. If the number of features is a handful, we can visually inspect to understand the overall feature interaction of features. If the number of features is numerous, we can instead do mathematical aggregation, such as mean value to finalize the extent of interaction.

There are a few caveats with using SHAP feature interaction.

1) The interactions identified by SHAP will be as good as the model. If the model is not of good performance, interaction effects detected by SHAP will not be reliable.

2) Calculating SHAP interaction values is computationally expensive.

3) There is no fixed cutoff value of the SHAP interaction value. For some interactions, it can be 0 or very close to 0. If a cutoff is indeed needed to be established, it will be the discretion of the analyst. We put a

cutoff of 0.1 for the sake of convenience. We will remove interactions for which SHAP mean score is less than 0.1.

4) The method shap_interaction_values is available for TreeExplainer in SHAP python library. It can only be used for tree-based bagging and boosting technique for obtaining interactions. It cannot be used for models which do not use a decision tree at its core, such as linear regression, or logistic regression. etc.

One advantage of using SHAP for interaction effects is that it can be used for both regression and classification. This is unlike the interaction plot method discussed earlier, which is useful only for regression.

Let's use this method for finding the interaction effect in the coupon recommendation dataset and car sales dataset.

5.2.1 Car Sales

For the car sales dataset, we ran the experiment across all cross-validation training samples. Each cross-validation has a set of interactions discovered by SHAP. To avoid overfitting, we will consider only those interactions which are present across all cross-validations. We shortlisted 374 interactions that were present in all 5 cross-validation samples and where the SHAP value was greater than or equal to 0.1. Also, SHAP values for the 374 features were obtained by averaging across 5 cross-validations. By looking at the interactions discovered by SHAP, we can infer that it was able to detect interaction only amongst higher-order features. Figure 5.2.1a is the plot for the bottom 5 interactions.

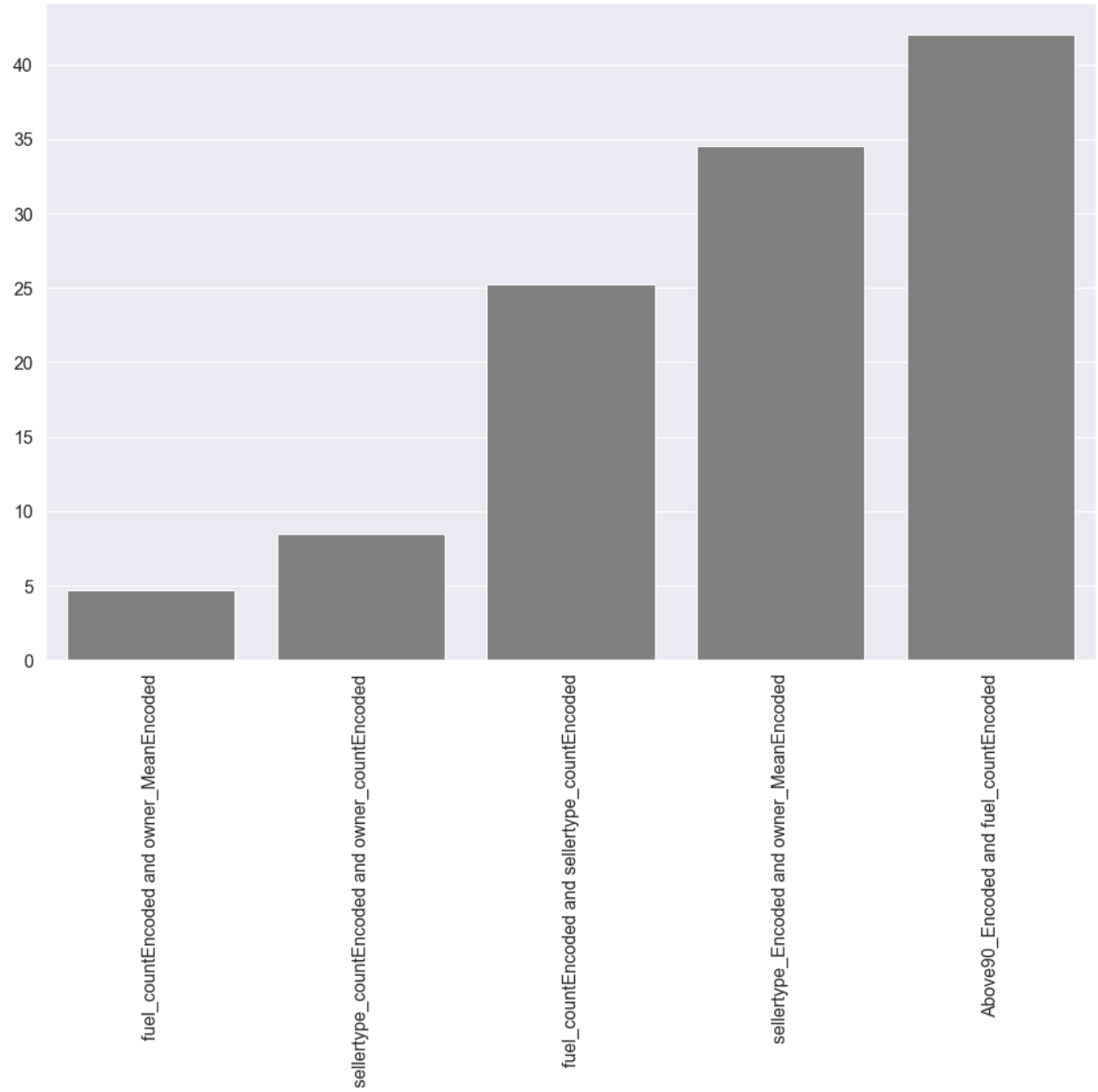


Figure 5.2.1a Bottom 5 interactions

Higher-order features for fuel and owner have the lowest degree of interaction effect. For fuel, count encoding of different categories is a numerical variable. It has a weak interaction with the mean encoding for dependent variable for the owner feature. The rest of the bottom 5 interaction effects are amongst higher order features and none of these are original features.

Now let's understand the top 5 interaction effects in figure 5.2.1b.

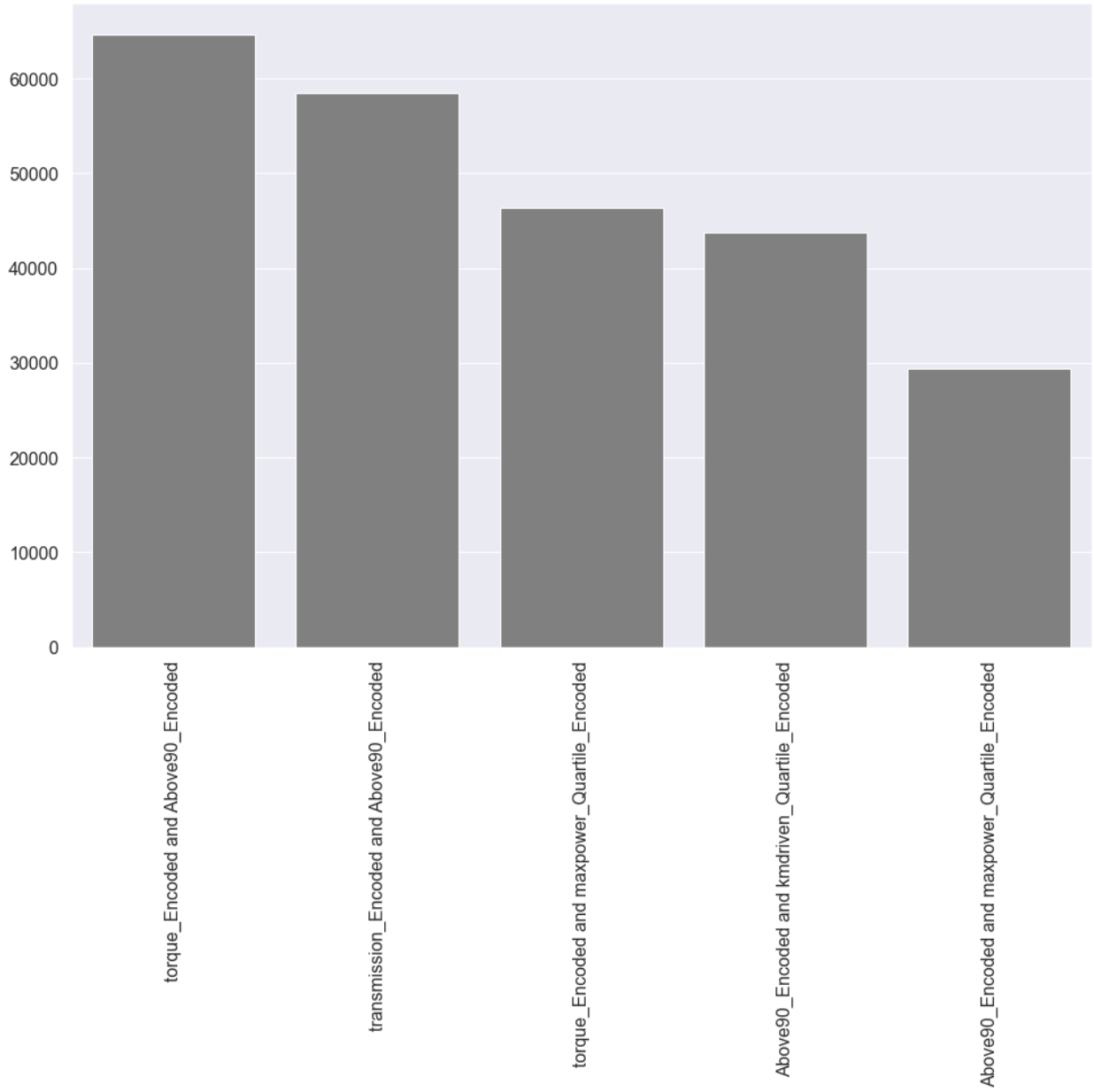


Figure 5.2.1b Top 5 interactions

Label encoding of 2 features namely torque and Above90 have the highest degree of interaction effect. Since both the features are categorical, we can do a simple concatenation between the two features to obtain an interaction effect feature. For the rest of the interactions in the top 5, all the features are label-encoded feature of a categorical feature. For such interaction between categorical encoded features and other higher-order features, we can simply perform concatenation between two features to obtain interaction effect feature.

5.2.2 Coupon Recommendation

Now let's look at the coupon recommendation dataset. The interaction was detected amongst higher order encoded features of categorical features. Although the magnitude of the interaction effect is small, there were 6 interaction effects detected in this dataset. These interaction effects were present across all cross-validation samples. These are shown in figure 5.2.2.

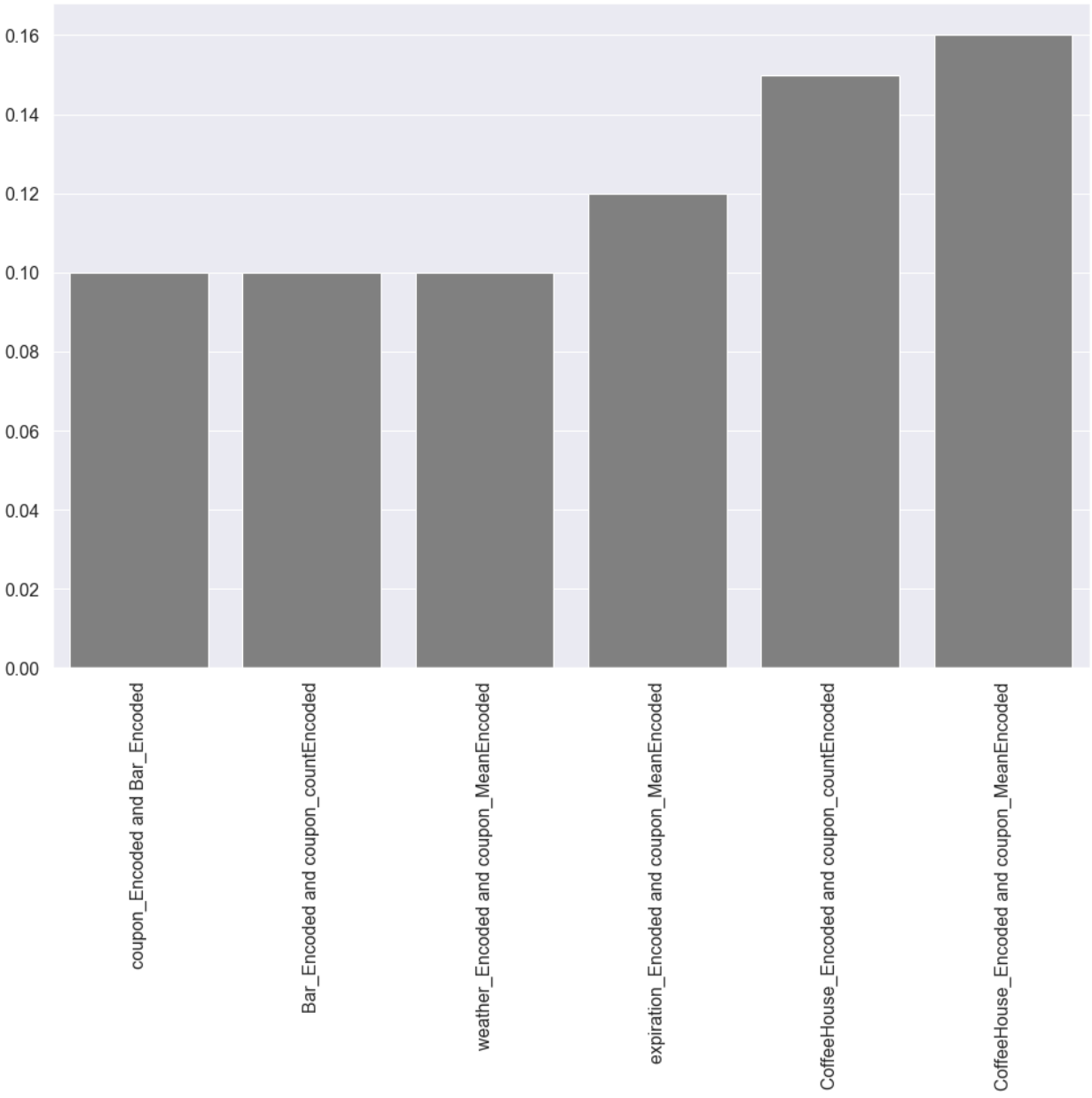


Figure 5.2.2 Interactions detected for coupon recommendation dataset

The smallest degree of interaction was detected between label encoding feature for bar, and label encoding for the type of coupon. Both features are numerical. For the rest of the detected interactions, at least one feature is label-encoded features of a categorical feature, while the other feature is a count or mean encoded feature. The only way to create interaction feature between these two types of features is by creating dummy variables of the label encoded feature and multiply it with the numerical feature.

The next step will be to create new categorical or numerical features from these features. For newly created categorical features, these can't be used directly. To be able to use these features, we will need to further create higher order features from these categorical interaction effect features.

For the rest of the datasets, the feature matrix is huge. It is computationally very expensive to detect interaction effects through SHAP. Also, most of the features are higher order features for either date of check-in, revenue, and booking patterns. Hence, there is a chance that very few interaction effects could be detected which are meaningful and could be explained to laymen.

5.3 Putting Everything Together

After learning different methods of feature engineering, let’s now put our knowledge into practice and measure the model performance. We will use both linear and tree-based nonlinear models for benchmarking purposes. In linear, we will use linear and logistic regression. For tree-based nonlinear, we will use Lightgbm, and Xgboost models.

In some cases, linear models will perform better than nonlinear models and vice versa. We will select the model which gives the best performance and is easy to explain to a non-technical audience. The results will be the first benchmark performance. We will try to find models which perform better than the benchmark performance using methods discussed in section III.

5.3.1 Hotel Total Room Booking

Let’s try to understand the hotel total room prediction data. It is a regression problem and the total occupancy for hotels for a specific check-in date is the dependent variable. We tried Lightgbm, Xgboost, and linear regression. Lightgbm regression gave the best performance. Figure 5.3.1 explains the performance of the Lightgbm tree model on cross-validation test, validation, and external test data.

The average RMSE for the cross-validation test and validation data together is 16.4. The average RMSE for the external test data is 12.9. This model predicts the total number of rooms that will be sold for a future check-in date, based on which the hotel property manager will ascertain market demand and set the price for the unsold rooms.

For any prediction made for total occupancy, RMSE is an indicator as to what extent predictions might have an error. RMSE of 16.4 means, predictions might be off by 16 extra rooms or 16 less rooms. For the external test data, we can see that model is performing better at 12.9 RMSE. There are 3 issues in this model. The first issue with the model is that RMSE is very different across different test sets. The second issue is that within the external test data, RMSE is different for each cross-validation. The third issue is high RMSE i.e., we will like to have a model which has the lowest amount of RMSE possible. If we can get a model with a lower RMSE, it will be easier to convince the stakeholders to use the model.

These 3 issues make the model unreliable to use. We will try to use feature selection to reduce the noise in the model. We will discuss different methods for feature selection across different chapters in section III.

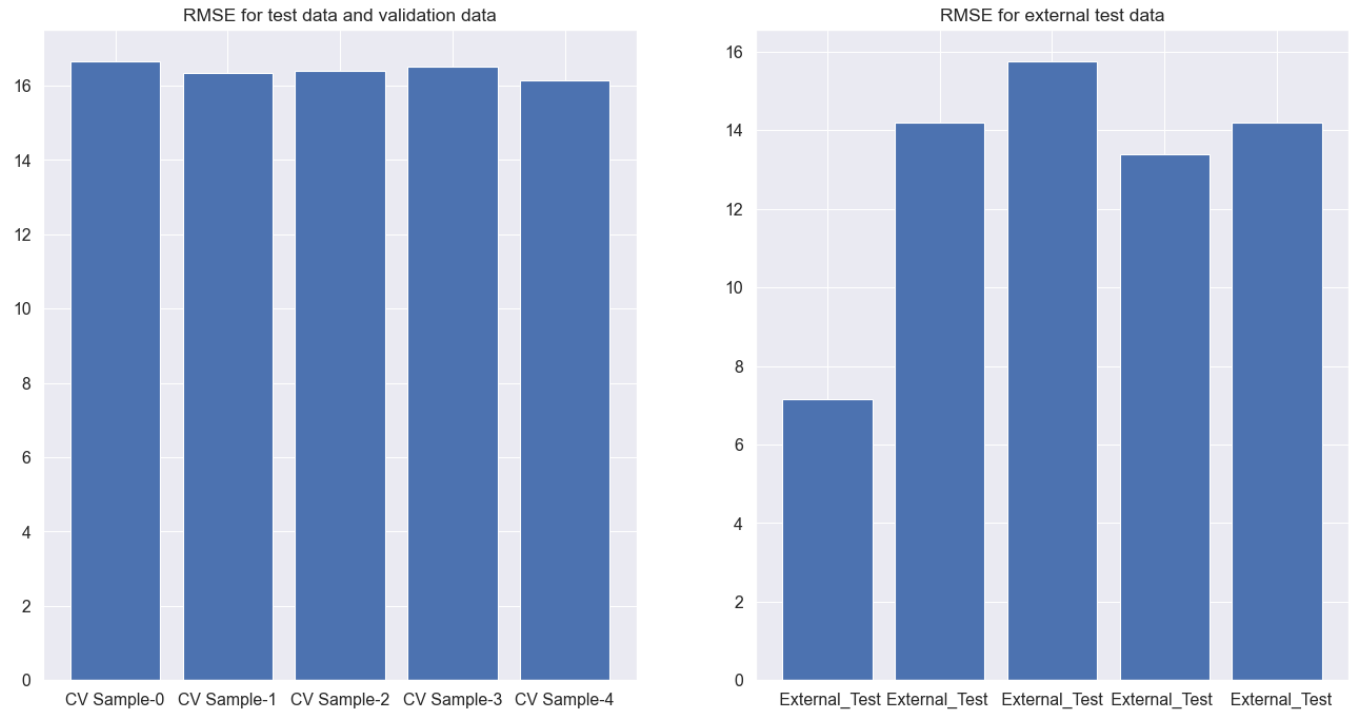


Figure 5.3.1 performance of Lightgbm tree model on cross-validation test, validation, and external test data for hotel total room booking prediction

5.3.2 Hotel Booking Cancellation

Let's try to understand the model performance for hotel booking cancellations data. It is a classification problem and 1 means canceled and 0 means not canceled. We tried Lightgbm, Xgboost, and logistic regression models for classification.

We are using the precision score for ascertaining model performance. Overbooking is a phenomenon wherein hotels sometimes sell more rooms than their available. As a result, a situation can arise when more than one guest can come to the hotel on the date of check-in to request their stay. On the other hand, many guests cancel their booking before the check-in date. This might lead to another problem wherein hotels have an unsold inventory of rooms.

If we can develop a model that can identify transactions most likely to be canceled, with a high degree of precision, hotels can minimize the loss occurring due to unsold inventory and increase profit by indulging in overbooking.

Xgboost had the highest precision, followed by Lightgbm. Xgboost had lower recall than Lightgbm. Figure 5.3.2 explains the performance of the Xgboost tree model on cross-validation test, validation, and external test data for hotel booking cancellation.

The Xgboost classifier has a few issues. Firstly, the precision values are not above 0.9 in both the test data. Secondly, the precision score varies across cross-validation samples. For example, in the first cross-validation training data, external test data has the lowest precision score and for the third cross-validation training data, precision is the highest. Thirdly, although precision scores are very near to each other, there is a difference of 0.04 in precision scores between external test data and the combination of cross-validation test data and validation data.

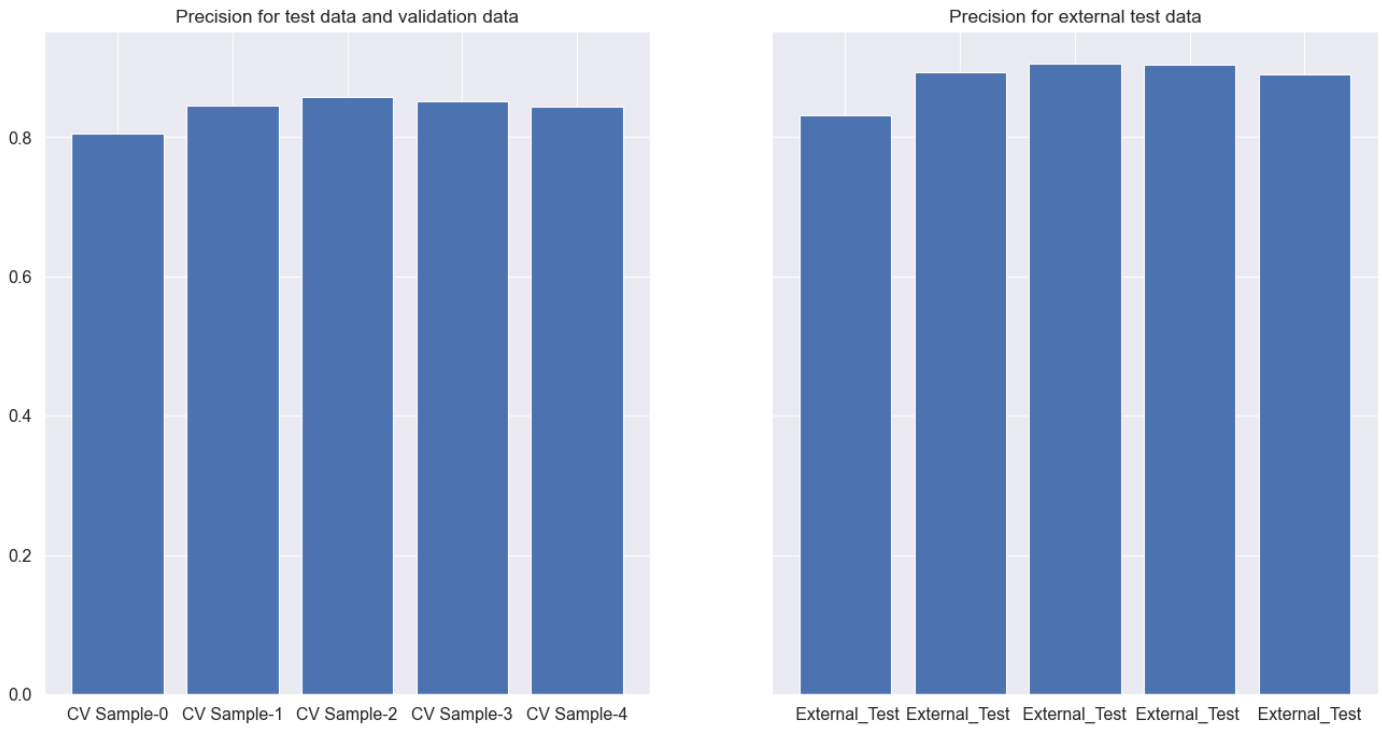


Figure 5.3.2 performance of Xgboost tree model on cross-validation test, validation, and external test data for hotel booking cancellation.

We will try different methods of feature selection discussed across different chapters in section III to help resolve the noise issue and try to identify features that are high in the signal.

5.3.3 Car Sales

Let's try to understand the model performance for car sales data. It is a regression problem and the selling price of the car is the dependent variable. We tried linear regression, Lightgbm, and Xgboost regressors to model the car price. The dependent variable has car prices for used cars in Indian rupees.

Lightgbm performed better than the rest of the modeling techniques. Figure 5.3.3 explains the performance of Lightgbm for the combination of cross-validation test and validation data, as well as for external test data. RMSE for the linear model is unreasonably high. For the Lightgbm model, for the combination of cross-validation test data, and validation data set, RMSE came as 406737.9, and for the external test data, RMSE came as 264851.9. This means while predicting the price of used cars, the model can make an error on an average to the extent of a little more than 400000 Indian rupees. In the US Dollar

to Indian rupees exchange rate of 1 USD = 80 Indian rupees, the model can make an error to the extent of $5000. This is a very high error margin for a model that is trying to predict the price of used cars.

There are two more problems. Firstly, RMSE is not consistent across both test datasets. A good model should be able to generalize well. In this situation, the model cannot generalize equally on the test, validation, and external test data. Secondly, RMSE varies hugely across different test data samples in cross-validation. This is even true for external test data. Here we can see that model with the provided set of features is unable to predict reliably on the external test data, at different cross-validation.

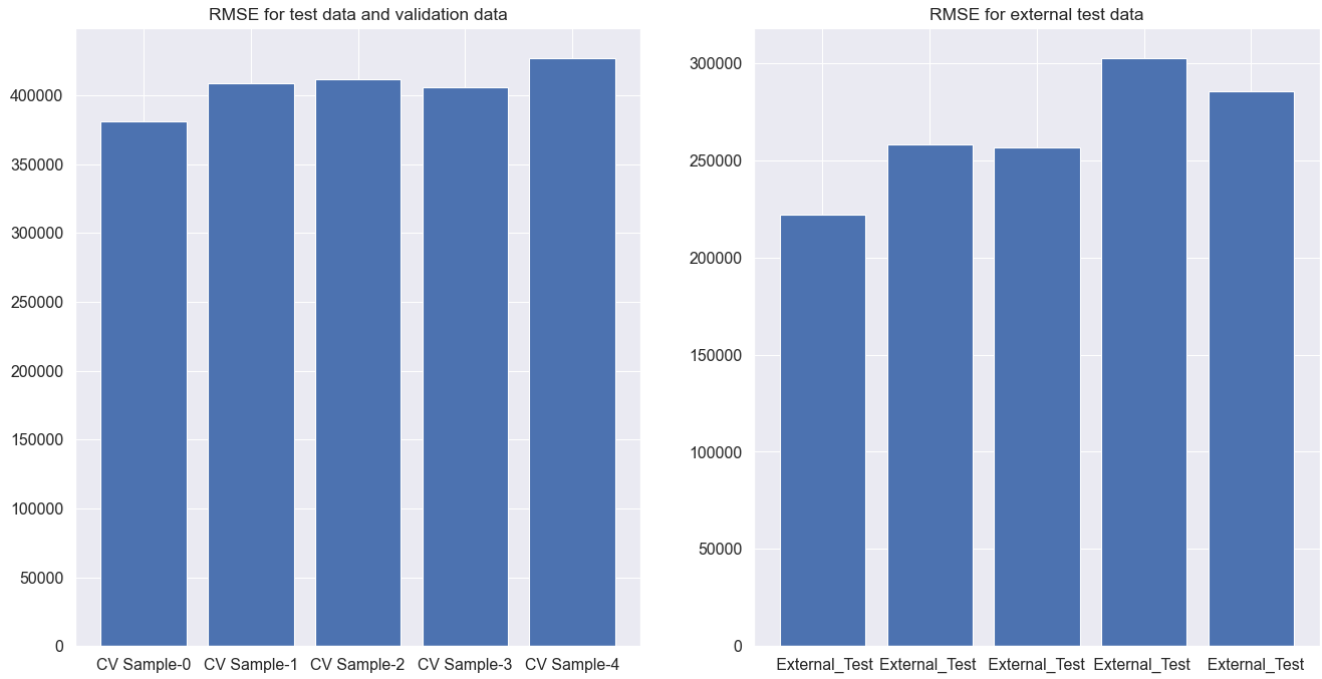


Figure 5.3.3 performance of Lightgbm tree model on cross-validation test, validation, and external test data for used car price prediction.

We will try noise reduction from features with the help of feature selection techniques in the next section. We will try to develop a model which can predict with smaller, and acceptable RMSE.

5.3.4 Coupon Recommendation

We used Lightgbm, Xgboost, and Logistic regression models and checked precision, and recall. Lightgbm performed the best and is presented in figure 5.3.4 for the coupon recommendation dataset.

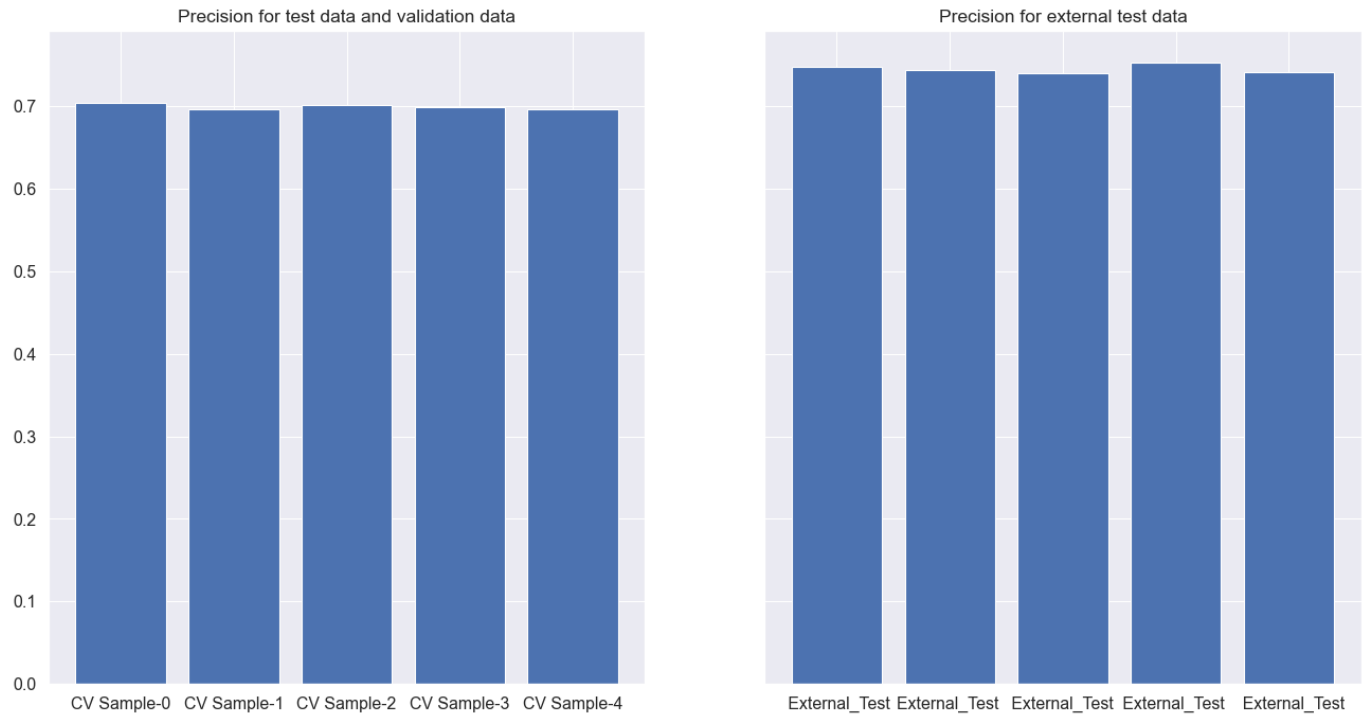


Figure 5.3.4 performance of Lightgbm tree model on cross-validation test, validation, and external test data for coupon recommendation.

Precision for both the test and validation is close to 70, whereas, for the external test data, it is 72.44. Recall for both the datasets are 74.5, and 77.9 respectively. Neither the precision nor recall are satisfactory for this model. Our goal for this dataset will be to remove noise from the data so that we can improve both the precision and recall.

We will try to use feature selection to find features that are high in signal and we will remove features that are high in noise in the next section of this book.

5.4 Conclusion

Interaction effects exists amongst features in a dataset. By accounting these effects, we can develop a model that reflects the true relationship amongst features. Both the methods SHAP, and interaction plot discussed in the chapter are useful in discovering interaction effect amongst features.

One caveat while looking for interaction effects is that we should also avoid looking for interaction between different higher-order features of the same feature. For example, body weight and the square root

of body weight are derived from the same feature. It will be useful to search for the interaction effect between these two features.

For all the 4 datasets, we included EDA features, higher order features and interaction effect features. In all the cases, there is a possibility of underfitting. Some features may contain more signal than others, whereas some features could contain more noise than signal. To reduce noise and find a limited set of useful features, we can remove the less useful features. There are many proven methods and techniques to perform feature selection. We will discuss these methods in the next section.

5.5 References

[1] Richard Li, Ashwin Shinde, An Liu, Scott Glaser, Yung Lyou, Bertram Yuh, Jeffrey Wong, Arya Amini, Machine Learning-Based Interpretation and Visualization of Nonlinear Interactions in Prostate Cancer Survival, American Society of Clinical Oncology, https://doi.org/10.1200/CCI.20.00002

Section III: Feature Selection

Chapter 6: Fundamentals of Feature Selection

6.1 Introduction

The objective behind training a machine learning model is to accurately predict the target variable using features. Some of these features facilitate the process of accurately predicting the target variable. These can be considered as 'signals'. Some features will be weak at predicting the target variable. These are called as 'noise'.

If a model is trained to predict hotel reservations for a specific hotel, we might consider the demand for similar hotels in the vicinity as an indicator of market demand. In the absence of demand data for competitors, we can consider the pricing of rooms by competitors as indicator of market demand. Similarly, we can also consider seasonality for the weekday as another useful feature. i.e., day of the week out of 7 days. Here we have two sets of features, competitor pricing, and 7 dummy variables indicating each day of the week.

Let us take the example of another feature. Employee strength is the number of employees who come to work at the hotel daily. It is not constant and fluctuates every day. Using this information might not be the best indicator of demand for the hotel room. Less number of employees in a hotel in a day might affect hotel operations for a specific day, and a greater number of employees might affect hotel running costs. However, it might not have any significant impact on the way people book hotel rooms on a travel website. This is an example of noise feature.

It must be noted that some features could be weak signals and might fall in between 'signal' and 'noise'. For example, the demand for hotel rooms is seasonal in tourist locations. During the peak tourist season, we can expect hotel bookings at 100%, whereas during the off-season, occupancy might be less. In this situation, demand is seasonal and varies between each month. In contrast, the day of the week starting Monday to Sunday does not give clear indication of occupancy levels. It varies for each specific month. Here, day of the week could be a noisy feature in comparison to the month of the year for the check-in date.

There can be an interaction between a strong signal feature and weak signal feature. It can occur that the combined presence of both the features in the model gives higher predictive power, than when they are present individually in the model. Hence, our objective in doing feature selection is to identify the

combination of features that gives the best predictive power. Feature selection is even more relevant in high-dimensional data, where we need to reduce noise and identify features that are rich in the signal. A smaller number of features can significantly reduce model training time as well as the speed of prediction during production deployment. It also makes the model less complex and easier to explain the model prediction to laymen.

We will be performing cross-validation during feature selection. The methods discussed in this chapter return a list of features for each cross-validation. For ease of understanding, we will look at how many features are common across all cross-validation from among the selected features.

6.2 Different Feature Selection Methods

There are three broad approaches to feature selection. Filter method, Wrapper method, and Embedded method. We will discuss the first two methods in this chapter. For certain modeling techniques, feature selection and model training happen in a parallel fashion. These are called embedded methods of feature selection. Also, for certain modeling techniques, feature selection should be approached after keeping in mind the nuances of the technique. For certain techniques, feature selection is altogether discouraged. We will discuss these fine nuances in detail in chapter 7.

Apart from filter, wrapper, and embedded methods, there is another, 4th group of feature selection techniques. These are known as metaheuristic algorithms. In chapter 8, we will discuss these techniques in detail.

6.3 Filter Method

This method uses the characteristics of the data. This method is independent of the machine learning technique. This is done before training the model. It requires less computational power and is usually fast. This method can miss out on removing features that are not useful for the modeling problem. This method is sometimes used as a preliminary feature selection method for datasets. After obtaining a reduced set of features from filter method, these features are introduced to other sophisticated feature selection methods. This approach is useful when the number of features is too high and computing power is limited. The reduced set of features can then be used by advanced methods to give superior results.

It mostly uses correlation coefficient and hypothesis testing techniques such as ANOVA, T-Test, and Chi-Square test to identify the relationship between the dependent variable and independent variables.

Features that do not have any proven relationship with the dependent variable are discarded from the features for modeling.

If the nature of the modeling problem is classification and the feature is categorical, we can use the chi-square test to validate if the distribution of different categories across different classes is statistically significant. If the p-value is above 0.05, we can remove the categorical feature, as the distribution of categories across different classes is not significant. This could imply a weak predictor.

For classification problems and numerical features, we can perform ANOVA. The dependent variable has a value of either 1 or 0 and can be considered as 2 categories. We can calculate averages for both classes for numerical features. With the help of ANOVA, we can test the hypothesis if the different mean value thus calculated for each class of dependent variable is statistically significant. If it is not significant, we can remove such numerical features. ANOVA can also be applied for the numeric dependent variable and categorical features, in other words, regression models and categorical features.

For regression problems and numerical features, we can use correlation to ascertain the strength of the feature. Although correlation is not causation, this technique can still be used for high-dimensional datasets. One caveat of this method is deciding the correlation threshold to use for accepting or rejecting a feature as useful. Although it is subjective, we can keep a higher threshold such as 0.5 and above. For linear regression with multiple features, using correlation as a feature selection method has its limitations. The biggest risk will be losing out on potential interaction effects, as correlation only talks about the relation between the feature and the dependent variable in isolation, without accounting for any other feature. The same could be said for the Chi-square test and logistic regression. In such a situation, regularization techniques such as Lasso would be the most ethically sound method for feature selection for linear regression.

6.4 Wrapper Method

This method uses a subset of features from the original list of features to train the model. It is a greedy method, as it evaluates all possible combinations of features to find the best possible combination. The inference is drawn from a model trained on the subset of data. Based on this, decisions are made to add or remove features from the subset. It evaluates many combinations of features against a specified model metric, such as the f1 score for classification and R square for regression. It returns a feature set that gives the best results. It is computationally more expensive than filter methods and takes more time to perform

feature selection. Even after performing this search, results might not always be desirable, as the wrapper method often leads to overfitting.

4 methods come under the wrapper method. These are forward selection, backward selection, Stepwise feature selection, and recursive feature selection.

6.4.1 Forward Selection

Forward selection starts with an empty set of features and features are added until the R square keeps increasing and F-statistics are significant. It starts with a model with a single feature. Models with single features with the highest r squares are selected. In the subsequent iterations, additional features are incrementally added to the model, until R square keeps increasing, and f-statistics is significant.

6.4.2 Backward Selection

This method starts with all features. Features for whom a change in R square is not reflected with statistically significant F-statistics are removed.

6.4.3 Stepwise Selection

Stepwise feature Selection is done by creating a regression model, followed by ranking features based on p values. Features with a p-value of more than 0.05 are dropped. The removed features can still enter the model at a later step. This process continues until it meets convergence criteria. However, it could lead to overfitting and increase in false positives.

6.4.4 Recursive Feature Elimination

The recursive feature elimination (RFE) is an iterative procedure and is an instance of backward feature elimination. The first model has all features and features are removed one by one based on some scoring function, such as importance of the coefficients to maximize some target metric. This is performed until the desired number of features remains. This is the most commonly used feature selection method.

Now let’s look at different datasets and the impact of each method on the model performance.

6.5 Putting Everything Together

We tried all the methods discussed in this chapter for the 4 datasets. In this section, we will look at the best results achieved for each dataset.

Certain feature selection methods are computationally expensive, whereas some others are computationally prohibitive. Backward and step feature selection methods in sequential feature selection are of computationally prohibitive type. We tried executing these methods in an intel i7, 64GB RAM machine. For linear and Xgboost models, many times its execution wasn't completed even after 48 hours. At this point, we stopped the Jupyter notebook cell and moved on to the next method. This criterion was applied to all methods where it took more than 48 hours, it was stopped.

6.5.1 Hotel Total Room Booking

We tried different models and feature selection methods. Of all the methods, Xgboost regression, when used with the filter method, gave the best performance. For cross-validation test, and validation data, the RMSE was observed to be 13.9. RMSE for the external test data was identified as 5.9. The detailed results for each cross-validation can be seen in figure 6.5.1.

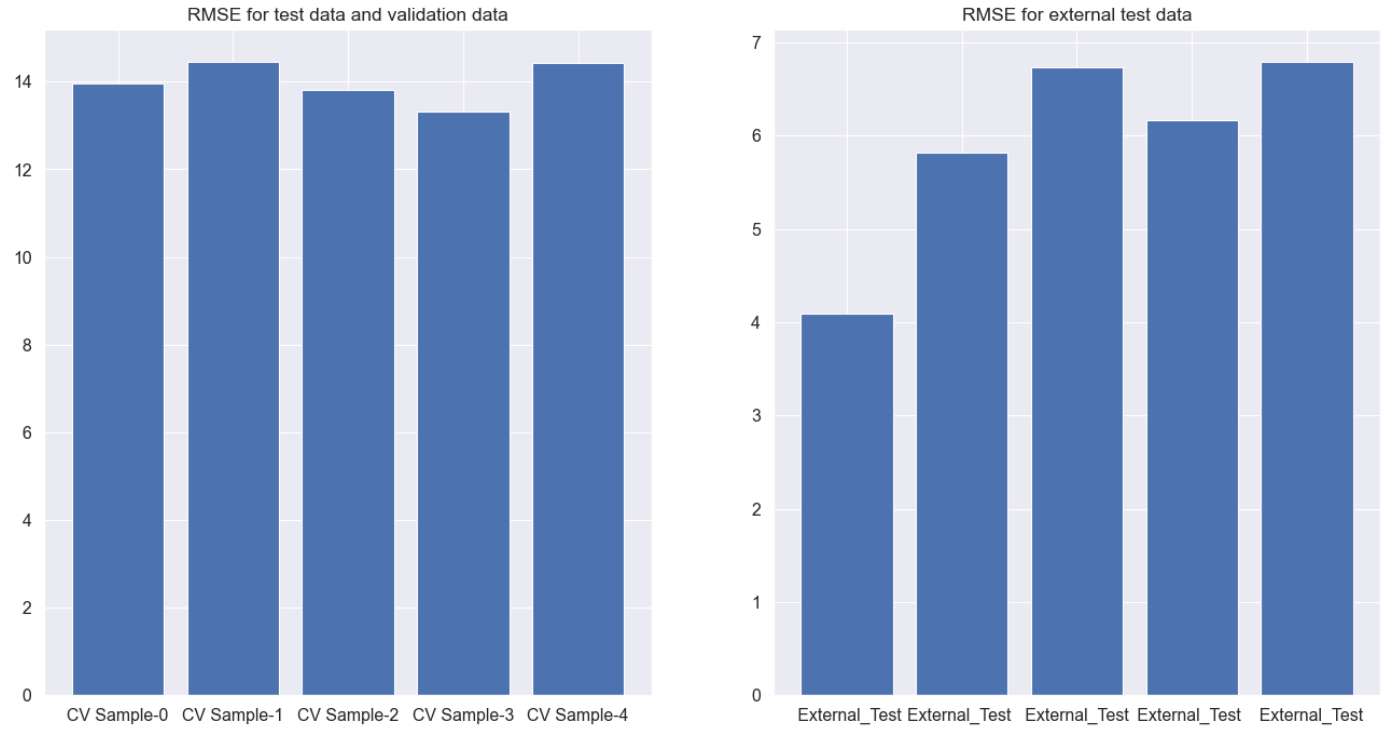


Figure 6.5.1 performance of Xgboost tree model with filter method for feature selection on cross-validation test, validation, and external test data for hotel total room booking prediction

The results came as an improvement upon the results reported in chapter 5. It still has a few areas for improvement. Firstly, the difference in results between the two test data sets is quite high. Also, among different cross-validations, the results are not consistent and fluctuate drastically. This can be seen by comparing the first vs third cross-validation results for external test data. Secondly, the results despite being better than what was reported in chapter 5, still fall short and RMSE are quite high. We will need to try other feature selection methods to see if the results are any different.

6.5.2 Hotel Booking Cancellation

We tried different models and feature selection methods. Of all the methods, the Xgboost classifier, when used with the filter method, gave the best performance. For the cross-validation test, and the validation data precision was recorded at 0.835, and 0.877 for external test data. The detailed results for each cross-validation can be seen in figure 6.5.2.

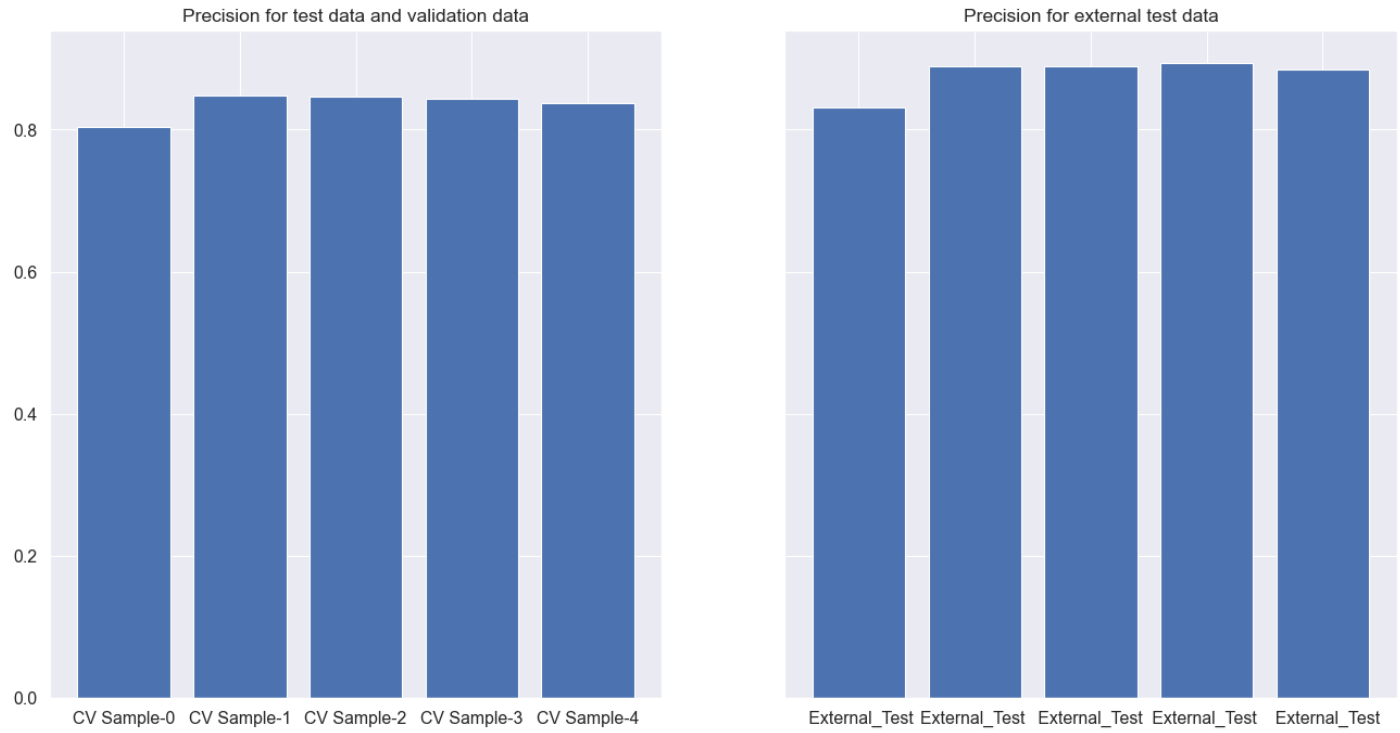


Figure 6.5.2 performance of Xgboost tree model with filter method for feature selection on cross-validation test, validation, and external test data for hotel booking cancellation prediction

The previous results achieved in chapter 5 were better than the results obtained through the filter method. However, recall improved to a slight extent, in comparison to previous results. Even then, we still need to improve on both the precision and recall for both datasets. Also, we need to bring the performance of both datasets to a similar level. Finally, the results in the 1st cross-validation is poor than the rest of the cross-validations. All these points suggest us to try other methods of feature selection to find a better solution.

6.5.3 Car Sales

We tried different models and feature selection methods. Lightgbm regression, when used with the filter method, gave the best performance of all the methods. For cross-validation test, and validation data RMSE was observed to be 398263, and 230356 for external test data. The detailed results for each cross-validation can be seen in figure 6.5.3.

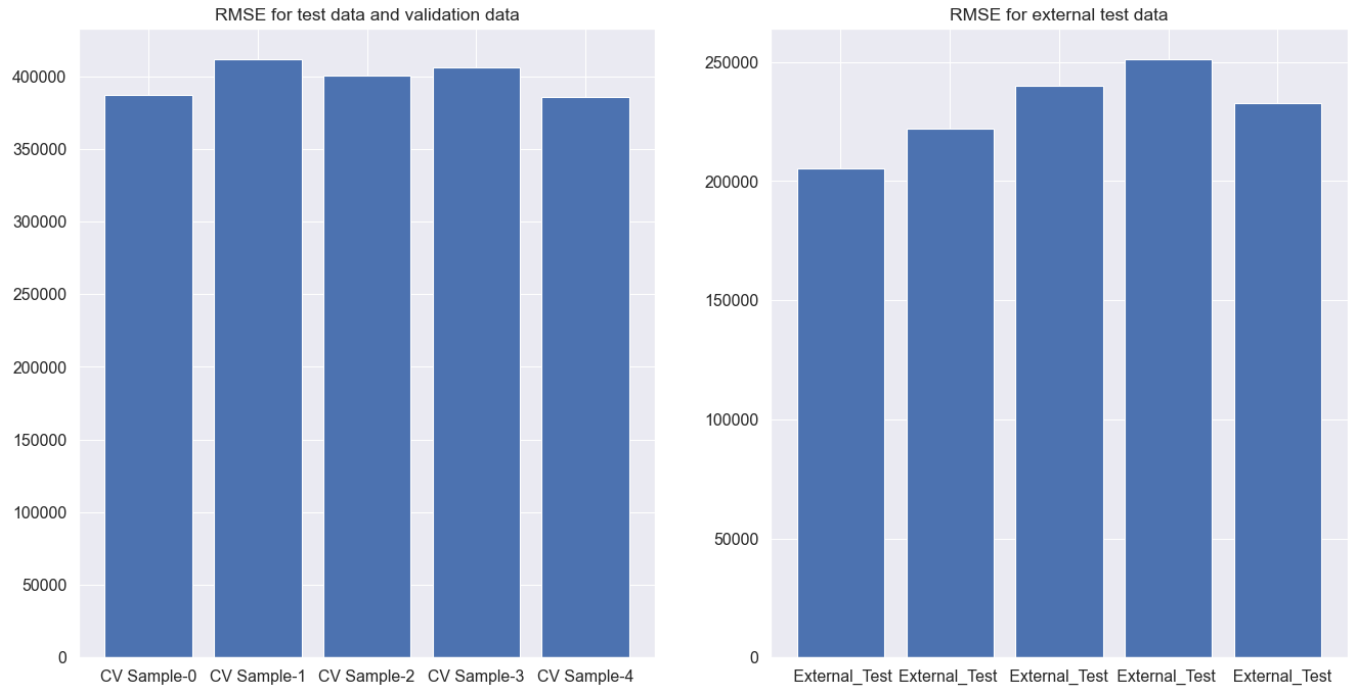


Figure 6.5.3 performance of Lightgbm tree model with filter method for feature selection on cross-validation test, validation, and external test data for used car price prediction.

The results obtained through the filter method are marginally better than the previous results presented in chapter 5. Even then, it suffers from the same issues we saw in chapter 5. Results across different test datasets are inconsistent. Results across different cross-validations are not consistent. If we had to use this model, we cannot say with confidence that it will generalize well on new unseen data, to the extent it does for the test, validation, and external test data. Also, the RMSE values for car prices are very high to be considered acceptable.

As the results presented for car prices are not acceptable enough to be used as a model, we will try other methods of feature selection in subsequent chapters.

6.5.4 Coupon Recommendation

We tried different models and feature selection methods. Of all the methods, the Xgboost classifier, when used with the filter method, gave the best performance. For the cross-validation test, validation data precision was recorded at 0.708, and 0.755 for external test data. Although recall worsened than previous results to a small extent. The detailed results for each cross-validation for precision can be seen in figure 6.5.4 below.

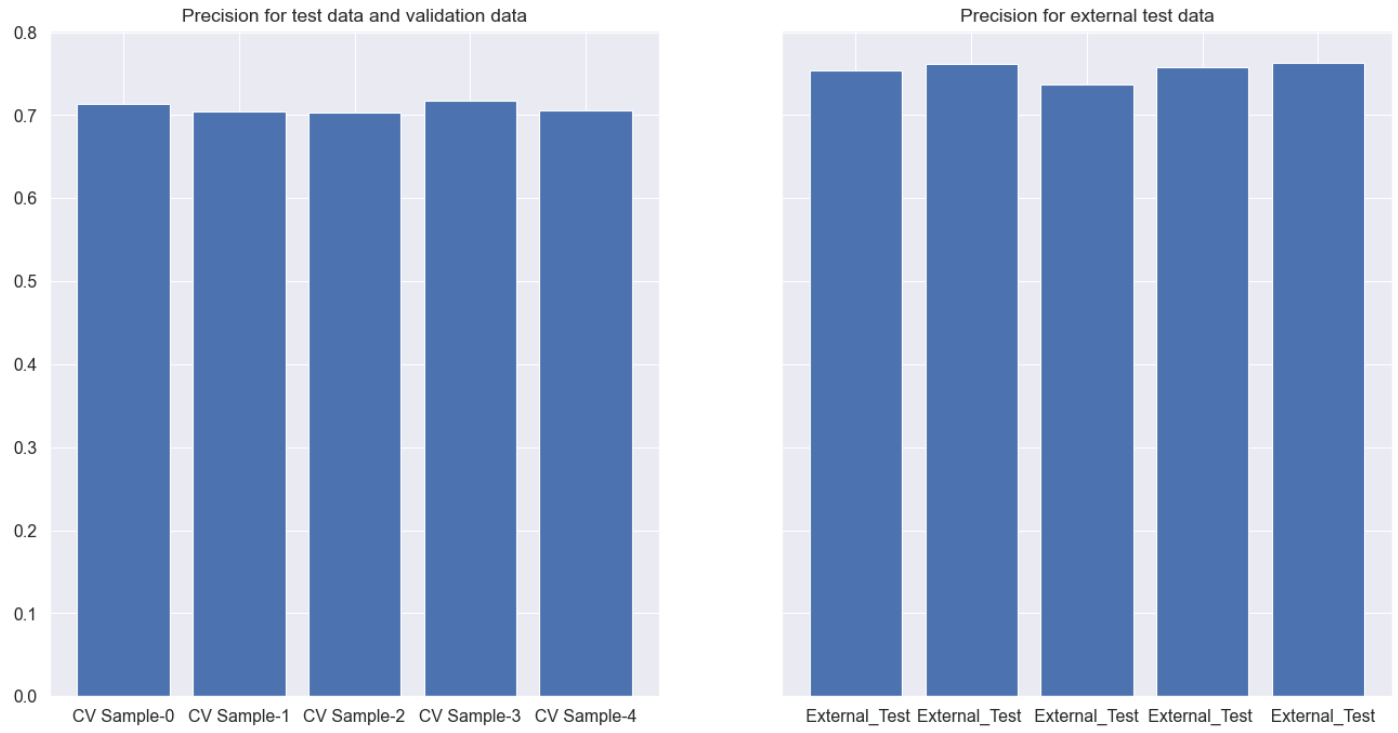


Figure 6.5.4 performance of Xgboost tree model with filter method for feature selection on cross-validation test, validation, and external test data for coupon recommendation dataset.

This model, like its predecessor model discussed in chapter 5 suffers from the same issues. The precision and recall are not good enough and the results in external test data are not consistent with cross-validations test and validation data. We will try other methods of feature selection in the next chapters to see if any improvement is possible.

6.6 Conclusion

Filter method can be very useful if we have huge number of features, say thousands, and we want to try the first model. We want to ascertain how well the model can perform. But we have limited computing power to train the model with all the features. Filter methods can help us reduce the size of feature matrix. This comes with a tradeoff between losing an important predictor. Something we should keep in mind while performing feature selection with filter method.

Wrapper methods is computationally expensive. The only exception to this is the forward selection method. It may or may not be useful in searching better performing models. This method should be tried

if we have the required computing power and time to explore a greater number of feature selection methods.

Chapter 7: Feature Selection Concerning Modeling Techniques

In this chapter, we will discuss three topics. First is embedded methods for feature selection such as lasso, and feature importance in tree-based models such as the random forest. The second topic is the interplay of linear feature selection on non-linear models and vice versa. The third are considerations we need to have while doing feature selection for some modeling and preprocessing techniques, such as linear regression, SVM, PCA, etc.

The embedded method performs feature selection during the process of training the model. These algorithms are part of the modeling technique itself. It considers dependencies amongst features, as well as the relationship between the input feature and the output.

We will perform cross-validation while using techniques mentioned in this chapter. We obtain the list of features for each cross-validation. We will look at how many features are common across all cross-validation from among the selected features.

7.1 Lasso, Ridge, and ElasticNet

Least absolute shrinkage and selection operator (Lasso), Ridge, and ElasticNet are regularization techniques that prevent overfitting. Overfitting happens when the model performs poorly on unseen data in comparison to train/test data used during training. To fix this, a regularization parameter can be introduced in the model to suppress the bias in learned coefficients. It does so by reducing the size of the coefficients. Lasso, and Ridge, also known as L1 and L2 are two different regularization methods to regularize the linear model. These techniques can also be used for feature selection.

Let's discuss Lasso. Imagine there are 2 independent variables x1 and x2 with weights 11.1 and 10.5 respectively. Lasso will suppress the coefficient of either of the one variable to 0. So new weights might look like 11.1 or some different value for x1, and 0 for x2. Vice versa can also be true. As coefficients for some features become 0 in Lasso, it automatically means these variables are of no use for the outcome variable. Hence as a byproduct of L1 regularization, we can do feature selection to remove some features from the model and make it more robust, less complex, and computationally faster.

There are a few limitations of the Lasso model. Firstly, Lasso is more appropriate when we have a handful variables. If we have a large number of features, for example, a huge number of dummy

variables, it will randomly select one out of the many features. As a result, features that have very little bearing, based on business understanding can get selected. At the same time, variables that have a relatively higher impact might get rejected. It will select a useful set of features, not necessarily the most important features. Secondly, when features are collinear, it's not the best choice to use Lasso feature selection on collinear variables. However, if we standardize the features and perform feature extraction methods such as PCA on standardized features, we can perform Lasso feature selection on PCA components. The third limitation is that Lasso doesn't have any means of suggesting which feature is statistically significant. To know which variables are significant, we might have to feed the variables selected by Lasso and into a linear regression model.

Ridge regression shrinks coefficients towards zero but doesn't make the coefficients zero. New weights will reduce in magnitude and can become negligible but not zero. For structured data with sparse features, e.g. hundreds of thousands of dummy variables, ridge or ElasticNet with ridge weight can be used for feature selection. Although it will practically not 'remove' features, it will give higher weights for many useful features.

7.2 Feature Importance of Tree Models

Tree-based models are of two types. The first type is bootstrapped aggregated models. It consists of individual decision trees. The final prediction is averaged prediction from all individual trees. Random forests are a bootstrapped aggregation model. Boosting models are a series of connected models. They correct the error of previous decision trees. An example of boosting model is Xgboost.

It is easy to interpret a single decision tree by looking at the tree diagram. We can identify most and least important features from the visualization and we can use most important features and discard the rest. Tree models also have a built-in feature selection method. We can select the best features to split on when building trees. But this isn't foolproof and often leads to overfitting.

For bagging and boosting models, feature importance can be calculated for each feature individually. Feature importance is measured by calculating the weighted impurity decrease at each node by the number of examples at the node, for each decision tree. Resultant weighted impurities from all individual trees are averaged for features. Feature level averaged weighted impurities are normalized so that they add up to 1. Feature importance thus obtained, can be used for feature selection.

There are a few caveats to using feature importance in tree-based models for feature selection.

1) Feature importance is always relative to the feature set used and does not tell us anything about the statistical dependence between the target and features.

2) Feature importance does not take into account co-dependence among features. As a result, it might not represent the correct extent of feature importance.

3) Feature importance cannot be interpreted as a direct dependence between the predictor and target.

4) If the model is weak and cannot generalize well on test or validation data, then feature importance becomes less reliable to use for feature selection.

Despite all the caveats, the less computational time needed for tree models could make it a convenient method for feature selection for tree models.

7.3 Boruta

Boruta helps in fixing the limitations of the feature importance method in tree-based models. It uses randomization on top of results obtained from variable importance obtained from random forests to determine the truly important and statistically valid results.

A copy of the original dataset is created, in which features are randomly shuffled. These shuffled features are called 'shadow features'. The original dataset and the dataset with shadow features are merged as a single dataset. With the help of the z score, the hypothesis is tested whether the feature has significant information. If found insignificant, the feature is dropped. The remaining features go through the same process iteratively several times, as specified by the analyst. This process stops if either all features are removed or all the features are selected by the Boruta algorithm. This process can also stop if the specified number of iterations are complete.

7.4 Using Tree-Based Feature Importance for Linear Model

It might seem useful to train a random forest model and select high-importance features from the random forest model in a linear model. However, this can be problematic for two reasons.

Firstly, feature importance in the random forest is not comparable to importance as measured by a linear model using the beta coefficient. A feature can have high importance in a random forest model, while the

same feature can have a relatively lower beta coefficient in a linear model and vice versa. Or worse, the feature could become statistically insignificant in a linear model while it has high importance in random forest.

Secondly, random forest is good at finding nonlinear relationships between features and the dependent variable. On the other hand, linear models have a linearly additive relationship between independent variables. i.e., one feature's relationship with the dependent variable is independent of any other feature. The same can't be true for the random forest as it explores nonlinear relations.

7.5 Using Linear Model Feature Importance for Tree Models

Using linear models such as linear, logistic model and its feature importance or lasso model and the resultant selected features in a non-linear model such as random forest can be problematic. Linear models remove any feature that isn’t linear. Thereby important nonlinear features can get eliminated and the resultant random forest model could be less than optimal.

Despite all the reasons for not mixing linear feature selection with non-linear models and vice versa, we can still try it. If the empirical evidence suggests to us that doing so will yield good results, then it could be worth trying. For example, if after using random forest important features, the linear regression model metric improves and becomes better than the benchmark linear regression model, we can give it a try.

7.6 Linear Regression

Most common method of feature selection for anyone who went through a linear regression tutorial will be to remove features whose p-value is above 0.05. This approach to feature selection in linear models is problematic. Ideally, we include or exclude features based on the assumption that features are good or bad predictors for the outcome variable. Not simply by observing the p-value.

The p-value can tell us about the correlation of a feature with the variance in the outcome variable, given all other features have explained the variance in the outcome variable. However, it does not prove that the feature and dependent variables are unrelated. Also, P-values are subjective. Some consider 0.05, while some may stick to more stringent levels of the p-value.

The P-value of a feature, say X1 in a regression model doesn't explain what will be the impact on other features' coefficients and its p-values if we drop X1 for being insignificant. It might happen that non-

significant features might become significant after removing an insignificant feature. Similarly, features that are already significant can also become non-significant after removing an insignificant feature. Another aspect of going by the method of removing features whose p-value is less than 0.05 will be that features that are difficult to explain to the layman can be kept in the model, just because the feature is significant.

Below are circumstances when we can still include features that are statistically not significant.

1) We have a categorical variable and it is represented as dummy encoding. Of all the dummy encoding features, some are statistically significant while some are not. In such a case, we should keep all the dummy features, including those categories whose p-value is above 0.05.

2) Interaction variable between two variables is significant, whereas one of the original features is insignificant. We cannot keep the interaction effect without keeping the main effects. Hence it is not appropriate to drop insignificant features.

3) Model has a feature that is insignificant and after removing the feature, the predictive power of the model reduces. We will be better off keeping such features in the model if our main goal is predictive power.

4) For features that are the main focus of the research, even if these are statistically insignificant, we can keep these. For example, predicting alcohol consumption quantity with the probability of a car accident while driving. If alcohol consumption is statistically insignificant, we cannot remove the feature, as this is the central focus of the research.

5) In certain areas of science, we will need convincing reasons to exclude certain features. For example, in biostatistics, removing age, gender, etc. from the model is counterintuitive.

It must be noted that it is not a compulsion to keep insignificant features in the model always. We can remove an insignificant feature in below situations.

1) If the beta coefficient of the feature is nearly zero or negligible, then it will have minimal impact on the error and r-square of the model. In such cases, we can remove the feature if it is insignificant.

2) If a feature has high multicollinearity, dropping a feature will have less impact on regression error and goodness of fit.

3) If there are other and better features present, which have a very strong bivariate correlation with the outcome variable as well as with other features, we can drop the insignificant feature.

4) If a feature is hard to explain and statistically insignificant, we can remove such a feature.

5) If the presence or absence of the insignificant feature has a negligible impact on the predictive power of the model, we can remove it.

Instead of using p-value, we can use the beta coefficient for selecting features in linear regression model. Beta co-efficient shows the level of importance of the feature in the model. Features that have small beta are of less importance. If all the features are in the same scale of measurement through some transformation such as z-score, and there is no collinearity, we can compare features by the beta coefficients and select important ones.

If there is multicollinearity amongst features, we will first need to fix multicollinearity. If multicollinearity is hard to fix, we should either do ridge regression or ElasticNet regression. We can then use the coefficients to keep only those features that are highly impactful for the dependent variable.

7.7 SVM

Support vector machines learn very well from high-dimensional data. When we have more features, SVM uses the extra information from additional features and tries to build a better model. This is true even when multicollinearity exists. Hence, feature selection doesn't help mostly in improving performance for SVM.

If overfitting is of concern, we can instead select a suitable value of SVM parameter C through experimentation. It intuitively acts like ridge regression. In that it doesn't remove variables. It only changes the level of impact a specific feature has. This is best suited for situations when the feature set has many binary features, which increases degrees of freedom.

7.8 PCA

Principal component analysis (PCA) can help us with dimensionality reduction. It is a feature extraction technique. It can help us reduce a huge set of highly correlated features into a smaller set of 'components'. These 'components' are uncorrelated from each other. It has a very specific application in removing multicollinearity from features in linear models. One caveat with PCA is that the principal components are not ordered based on their importance for the supervised or unsupervised machine learning problem at hand.

Hence, although it might appear intuitive to perform feature selection on these reduced principal components, it is not recommended. If the model performance improves as a result of the feature selection of PCA components, it will be pure coincidence.

7.9 Putting Everything Together

For the 4 datasets, we will process all the feature selection methods discussed in the book. In this section, we will present the best results obtained for the dataset, across modeling techniques and feature selection methods in this chapter. The best performance will be briefly compared against the previously obtained best performance in previous chapters, we will be performing cross-validation. The methods discussed in this chapter return the list of features for each cross-validation. For ease of understanding, we will look at how many features are common across all cross-validation from among the selected features.

For the tree feature importance, we have considered the top 90 percent of features. For linear regression, we have kept the top 95 percent of features based on the beta coefficient.

7.9.1 Hotel Total Room Booking

We tried different models and feature selection methods. For Lightgbm regression, when used with the feature importance for 90% of top features, gave the best performance of all the methods. For cross-validation test, and validation data RMSE was observed to be 17.9, and 12.3 for external test data. The detailed results for each cross-validation can be seen in figure 7.9.1.

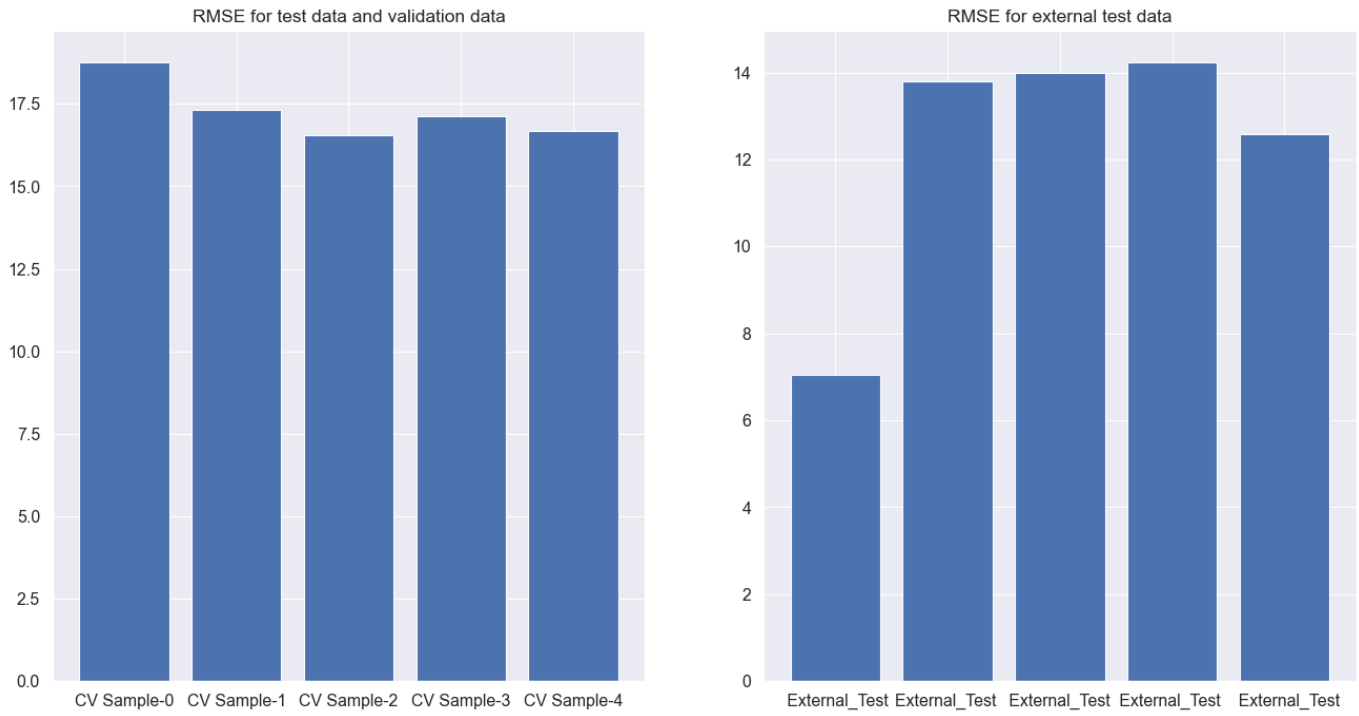


Figure 7.9.1 performance of Lightgbm tree model with filter method for feature selection on cross-validation test, validation, and external test data for hotel total room booking prediction

This is worse than the previous results presented in chapter 6. In addition to this, the results are inconsistent across different cross-validations and different test, and validation sets. Hence, we will discard this method.

7.9.2 Hotel Booking Cancellation

We tried different models and feature selection methods. Of all the methods, the Xgboost classifier, when used with Boruta, gave the best performance. For the cross-validation test, and the validation data precision was recorded at 0.835, and 0.881 for external test data. The detailed results for each cross-validation can be seen in figure 7.9.2.

The results obtained are almost similar to the results obtained in the previous chapter. Although the precision improved very marginally, the recall worsened than previous levels. There is no additional advantage in the results, as the model suffers from the same inadequacies that the model in chapter 6 suffered from. In such a scenario, it is up to the judgment of the analyst whether the solution should be

accepted or we should keep searching for other solutions. In our case, we will still like to try feature selection using metaheuristics techniques in chapter 8.

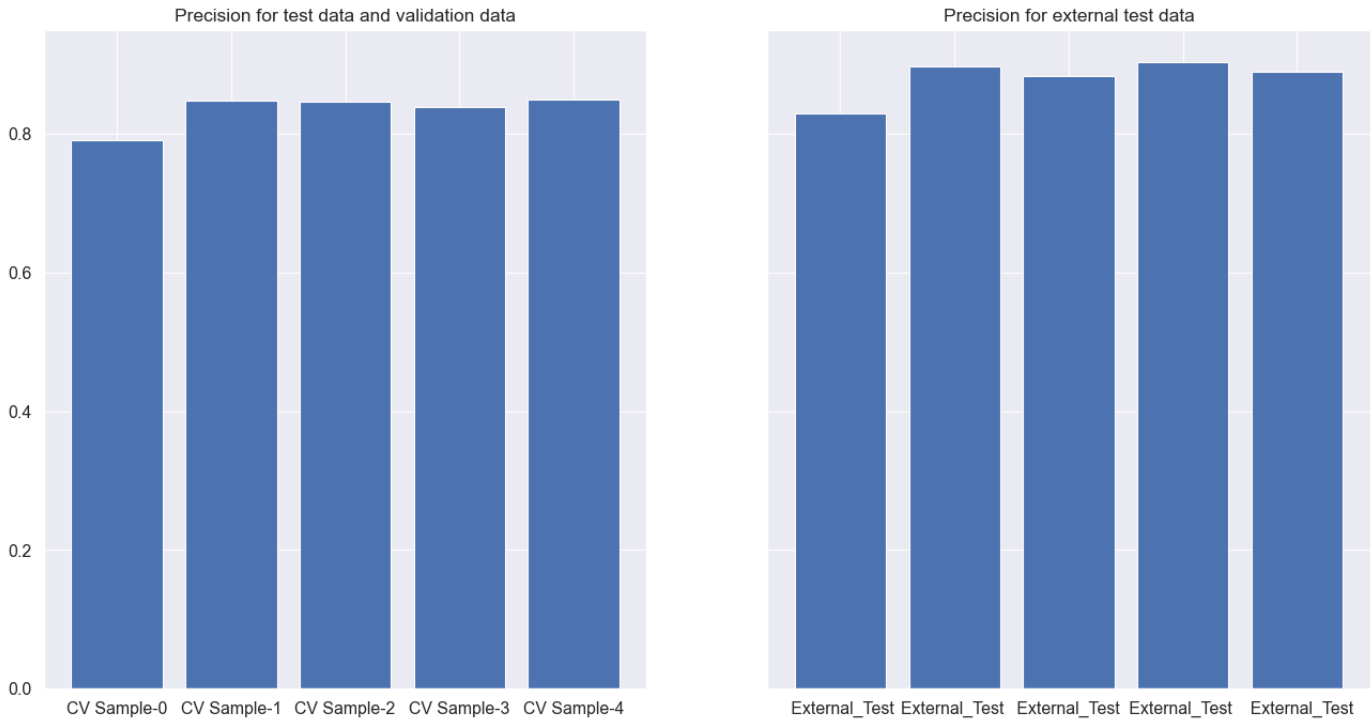


Figure 7.9.2 performance of Xgboost tree model with Boruta method for feature selection on cross-validation test, validation, and external test data for hotel total booking cancellation prediction

7.9.3 Car Sales

Lasso regression performed the best for car sales data. For cross-validation test and validation data, RMSE was 233161, whereas for the external test data it is 260101. It is better than the results obtained in chapter 6.

Figure 7.9.3 shows the model performance across different cross-validations. For the Lasso feature selection, results between external test data and other test and validation data are smaller than previously achieved results. It still suffers from 2 issues. Firstly, the RMSE is still higher than acceptable limits. As 200000 Indian rupees is still a very high error margin. Also, RMSE is not very consistent across all cross-validations. Hence, we might need more improvements to find a model with an acceptable RMSE.

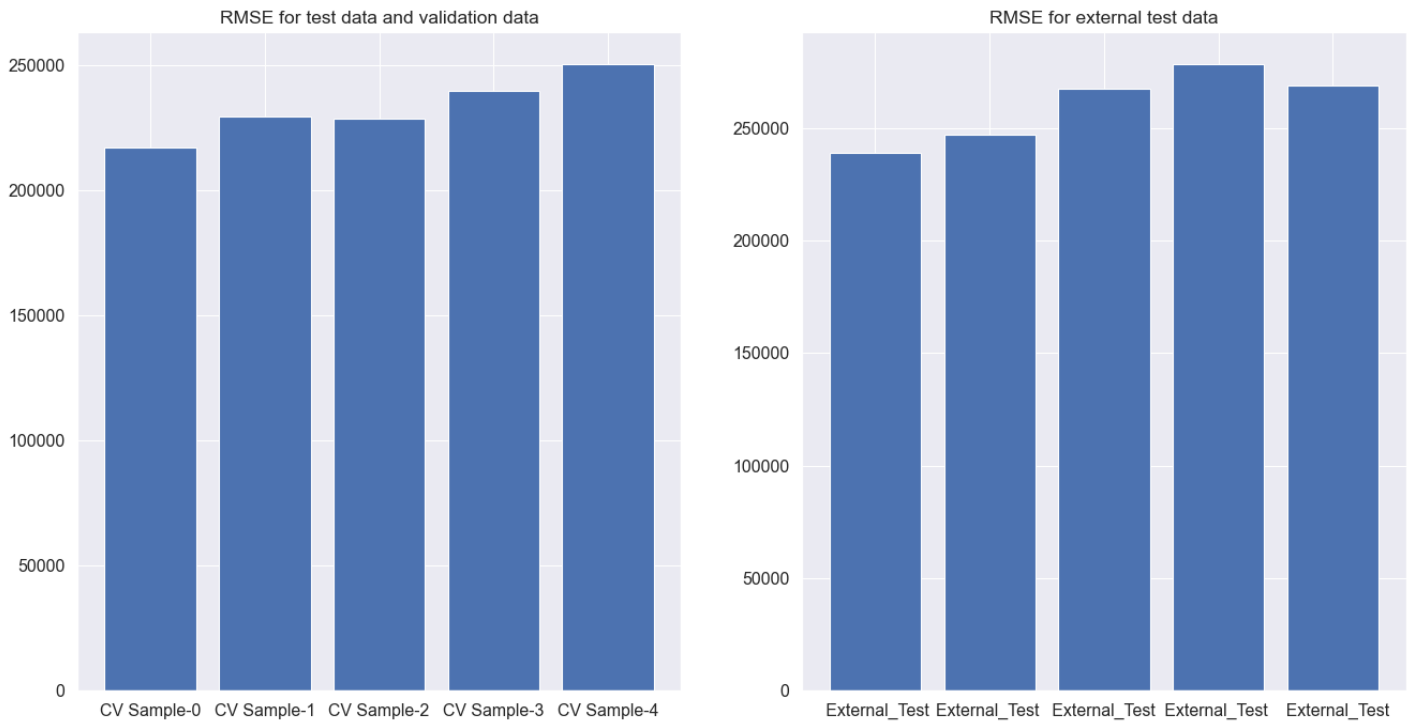


Figure 7.9.3 performance of the Lasso regression model on cross-validation test, validation, and external test data for used car price prediction.

7.9.4 Coupon Recommendation

We tried different models and feature selection methods. Of all the methods, the Xgboost classifier, when used with the feature importance for the top 90 percent of features method, gave the best performance. For the cross-validation test, validation data precision was recorded at 0.700, and 0.753 for external test data. This explains that the results are worse than the previously recorded best performance in chapter 6. In addition to this, recall worsened than previous results to a small extent. The detailed results for each cross-validation for precision can be seen in figure 7.9.4.

We will like to try the metaheuristics feature selection methods to see if these methods can bring any improvements.

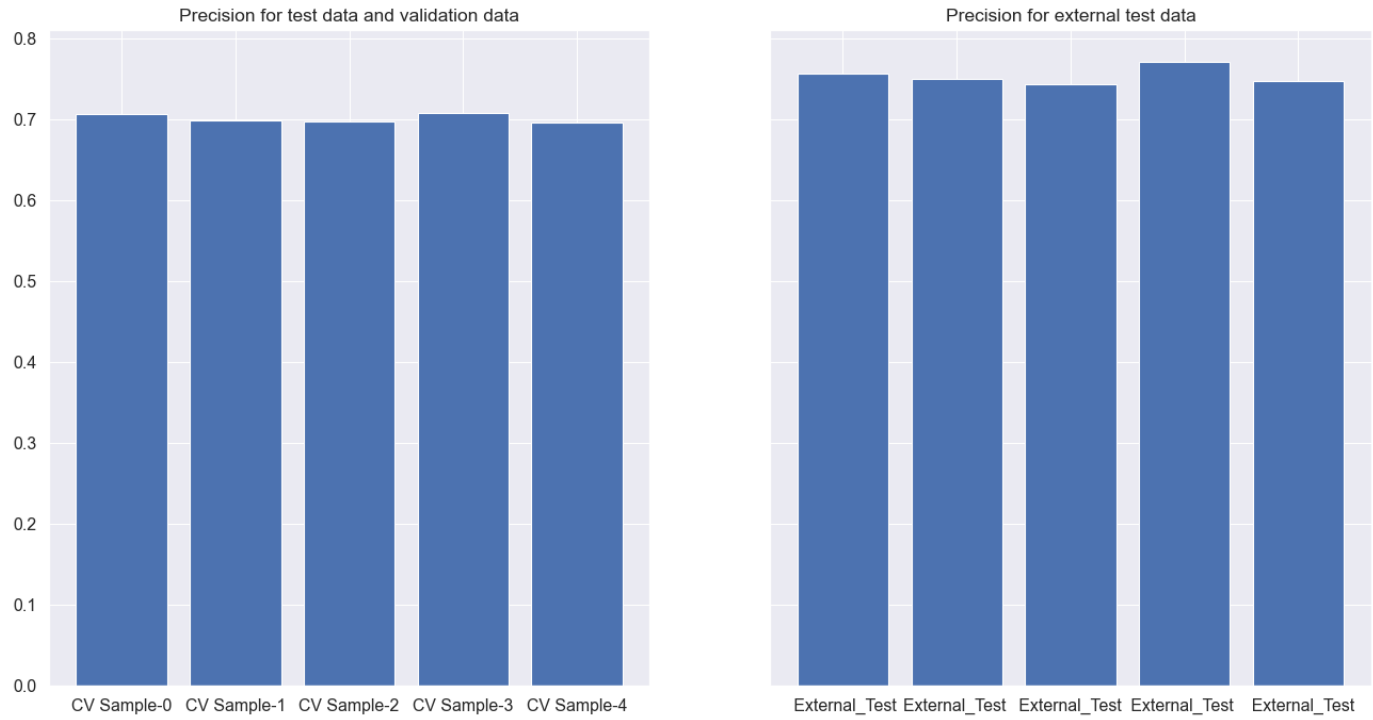


Figure 7.9.4 performance of Xgboost tree model with feature importance feature selection for top 90 percent of features on cross-validation test, validation, and external test data for coupon recommendation dataset.

7.10 Conclusion

We discussed different feature selection methods in this chapter. These methods can help us improve the model performance marginally. We cannot expect these methods to provide the best possible models. This is especially true when the model performance is already low. Most of the methods discussed in this chapter are not computationally expensive. The ideal scenario for using these techniques will be when we already have a well performing model and the number of features are very high. We can obtain a marginally better performing model with a smaller number of features with these methods.

Chapter 8: Feature Selection Using Metaheuristic Algorithms

Each feature in a dataset can have a main effect and an interaction effect, because of which, different combinations of features have varying model performance. This makes feature selection an inherently combinatorial problem. We need to find the combination of features that gives the best model performance. As the number of features keeps increasing, the number of possible combinations keeps increasing, and so does the computational cost of trying all the possible combinations. Metaheuristic algorithms can help us solve this problem by searching for a limited and lesser number of solutions. It does so by searching for better solutions iteratively. At the beginning of the algorithm, it starts with randomly generated solutions and tries to improve the solution at each iteration. Metaheuristic algorithms are procedures that can find a good solution for optimization problems, which are difficult and complex otherwise to solve manually. These partial search algorithms may provide a good enough solution, if not a perfect solution. These are very useful for feature selection, as they can help us find better feature sets than otherwise possible through manually trying different combinations.

We will discuss 4 metaheuristics algorithms in this chapter. These are genetic algorithm, simulated annealing, ant colony optimization, and particle swarm optimization. We have developed a companion python library MetaheuristicsFS, which has all 4 metaheuristics feature selection algorithms. Its module FeatureSelection helps us perform the desired feature selection.

Some parameters are common across all metaheuristic algorithms in this library. For example, cross-validation dataset, validation data set, and name of all input features. Imagine a scenario where you want to try multiple metaheuristic algorithms. You will need to enter these common parameters repeatedly for all the algorithms separately. To avoid this situation, we have used the ‘singleton’ approach in the MetaheuristicsFS library. The first step creates a feature selection object by providing common input parameters. In the second step, we can initialize any desired metaheuristic algorithm from the 4 listed algorithms. We will discuss this in the subsequent sections.

First, we will import the FeatureSelection module from the MetaheuristicsFS python library using the below syntax.

```
from MetaheuristicsFS import FeatureSelection
```

Let’s understand all the required input fields for the module FeatureSelection.

columns_list: It is a python list object and contains the names of all the features as strings, separated by a comma. These feature names are present in the training, test, validation, and external validation datasets. For example, if there are 3 features x1, x2, and x3, it will be represented as columns_list = [‘x1’, ‘x2’, ‘x3’]. Based on this input list of features, search algorithms create different combinations, to find the best possible feature combination.

data_dict: It is a python dictionary object and contains training and test data for multiple cross-validations.

Its key represents each unique cross-validation. For example, 0,1,2,3,4 represent 5 separate cross-validations. If a user wants to perform 5-fold cross-validation, data_dict should have 5 key-value pairs with 0,1,2,3, and 4 as keys. Each pair is created by shuffling the dataframe and splitting into train and test.

Each key has a nested dictionary containing training and test data. The value against each cross-validation key is a nested dictionary and contains features and dependent variables as a dataframe object. Key 'x_train', and 'x_test' contain feature dataframe for training and test data as value pairs. Similarly, the 'y_train', and 'y_test' key contains a dependent variable as a dataframe object for training, and test data respectively.

Below is what the dictionary structure looks like for 2-fold cross-validations.

```
{
        0:{
                "x_train":x_train_dataframe,
                "y_train":y_train_array,
                "x_test":x_test_dataframe,
                "y_test":y_test_array
        },
        1:{
                "x_train":x_train_dataframe,
                "y_train":y_train_array,
                "x_test":x_test_dataframe,
```

```
                "y_test":y_test_array

        }
}
```

x_validation_dataframe: It has feature dataframe for validation data set.

y_validation_dataframe: It has the dependent variable, stored as a dataframe for the validation data set.

model: It is the initialized model, stored as an object. For example, for the linear regression function in the Sklearn python library, the model object can be initialized as model = LinearRegression()

This object model will then be used for training the model by using training data and predicting for test and validation data. It should have a .fit attribute for training, and a .predict attribute for predicting. Sklearn models, as well as Xgboost and other major modeling techniques, have these 2 functionalities. It does not support deep learning models, however.

cost_function_improvement: There are 2 values for this parameter, depending on the goal of the optimization. We can select either ‘increase’ or ‘decrease’ as the string value.

Setting the value for ‘increase’ will enable the feature selection algorithm to look for solutions where model metric values increase in each iteration. One example can be f1 score for classification model. We will like to obtain a model that gives us highest F1 score.

Setting the value as ‘decrease’ will enable the feature selection algorithm to search for solutions where cost is lowest. For example, for regression models, RMSE is a commonly used cost function. It is desirable to obtain a model which has lowest amount of RMSE. Setting ‘decrease’ for this parameter will enable the algorithm to search for a feature set which gives lowest RMSE.

cost_function: Cost function is for finding cost between actual and predicted values. For a regression problem, some examples are root mean square error, mean absolute error, etc. For a classification problem, some examples are f1 score, precision, and recall. The cost function should have 2 input parameters 'actual' and 'predicted' as arrays. It should return the cost between actual and predicted values. It supports all the cost functions available in Sklearn.

It also supports custom-made cost functions, as long as there are 2 parameters in the cost function, actual value, followed by predicted values, and returns cost value.

average: In the case of multi-class classification problems, cost functions such as precision, recall, f1 score, etc. in Sklearn have a parameter 'average'. This criterion specifies the type of averaging to be used for the cost function if the dependent variable has multiple classes. We can assign the average value for Sklearn cost functions for multi-class classifications in this parameter.

Now let's initialize a feature selection object for a regression problem, where we will use a linear regression model with 3 features, and mean squared error as a cost function.

```
from Sklearn.metrics import mean_squared_error
from Sklearn.linear_model import LinearRegression
from MetaheuristicsFS import FeatureSelection


columns_list = ['feature 1', 'feature2', 'feature 3']

data_dict = {
        0:{
                "x_train":x_train_dataframe,
                "y_train":y_train_array,
                "x_test":x_test_dataframe,
                "y_test":y_test_array
        },
        1:{
                "x_train":x_train_dataframe,
                "y_train":y_train_array,
                "x_test":x_test_dataframe,
                "y_test":y_test_array

        }
}
```

```
model = LinearRegression()

fsObj = FeatureSelection(columns_list = columns_list,data_dict = data_dict, x_validation_dataframe =
x_validation_tree, y_validation_dataframe = y_validation_tree, model = model,
cost_function_improvement = 'decrease', cost_function = mean_squared_error)
```

After the feature selection object has been initialized, it can be used for executing a specified metaheuristic algorithm. Before we get into metaheuristic algorithms, let's understand why we need these algorithms in the first place by going through the first section of the chapter “exhaustive feature selection”.

8.1 Exhaustive Feature Selection

Exhaustive search method is a complete search method. It follows a brute-force approach. In this method, all possible combinations of features are tested to find the best possible combination of features. This guarantees an optimal solution. However, the computational resource and time required for an exhaustive search are very high. It requires 2^N combinations for N features. Even with a relatively smaller N, search space can increase exponentially. This makes it computationally expensive and sometimes computationally prohibitive.

Python library mlxtend has a module ExhaustiveFeatureSelector for performing the exhaustive search. It also allows deciding the minimum and the maximum number of acceptable features. Based on these thresholds, all possible feature combinations are generated. This can reduce the computational complexity to a certain extent, as the number of possible feature combinations is restricted. However, this comes at the cost of possibly missing out on a better combination that wasn't explored because it was higher or lower than the acceptable number of features specified in the module.

It is computationally prohibitive to use exhaustive feature selection in its true sense to find the best combination of features for a model, from amongst all possible permutations and combinations. Hence, we will not be using this method for feature selection. In the rest of the sections of this chapter, we will try to solve the limitations of exhaustive feature selection with the help of metaheuristic algorithms.

8.2 Genetic Algorithm

The genetic algorithm technique is an optimization technique. It is a population-based metaheuristics search algorithm, inspired by the natural phenomenon of evolution. Parents produce offspring, who inherit the strengths and abilities of both parents in the form of genes. This happens through a process called crossover. In the genetic algorithm, the crossover is carried out at one or more random cut-off points and the gene is swapped from both parents based on crossover probability. Figure 8.2.1 below illustrates the process of crossover. The cut-off point for the genes is kept in the 4th column for the parents and highlighted in red. Genes are swapped at the cut-off points and child 1 and child 2 have the swapped genes. Genes from the beginning until the cut-off point are the same as parents. From the 5th point onwards, genes are obtained from the other parents.

Parent 1	0	1	1	0	1	1	0	1	1	0
Parent 2	1	1	0	0	0	0	0	1	0	0
Child 1	0	1	1	0	0	0	0	1	0	0
Child 2	1	1	0	0	1	1	0	1	1	0

Figure 8.2.1: Crossover in genetic algorithm

There is also a likelihood of genetic mutation when a gene is passed from parents to offspring. In the case of the genetic algorithm, a mutation is introduced in the form of new patterns to randomly refresh the population. This encourages the search for unexplored feature combinations. A small probability (1%–2%) is selected for each possible feature as mutation probability. A random number is generated. If the random number is less than the mutation probability, then the feature is mutated and its status is changed. If it was originally supposed to be included, it is excluded and vice versa. Figure 8.2.2 explains the mutation process in the genetic algorithm.

Cells highlighted in red were mutated and values were changed. If the original value was 1, it was changed to 0, and vice versa.

Child 1	0	1	1	0	0	0	0	1	0	0
Child 2	1	1	0	0	1	1	0	1	1	0
Child 1	0	0	1	0	0	0	1	1	0	0
Child 2	1	1	0	1	0	1	0	1	1	1

Figure 8.2.2: Mutation in genetic algorithm

An initial set of solutions are created, called population. Each solution in the population is called a "chromosome". Each chromosome has a length equal to the total number of features. Individual features are represented as binary 0/1. 0 stands for a specific feature being removed from feature space and 1 stand for the feature being included. As explained previously, through the process of crossover, a list of the best chromosomes is selected and passed through mutation. This process is repeated several times, based on how many generations are defined by the user. Imagine there are 10 features x1 to x10 and we want to try 7 solutions at a time. i.e., a population of 7. Figure 8.2.3 illustrates what the initial solution will look like for 10 features and a population of 7. Each row will be a chromosome and all chromosomes combined will be called population. Figure 8.2.4 illustrates the genetic algorithm flowchart.

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
0	1	1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	1	0	0
0	0	1	1	1	0	0	0	0	0
0	1	1	1	0	0	0	0	0	1
0	1	0	0	1	0	0	1	0	0
1	0	1	0	0	0	0	0	1	0
1	1	1	1	1	0	0	1	1	0

Figure 8.2.3: Solution matrix for 10 features and population of 7

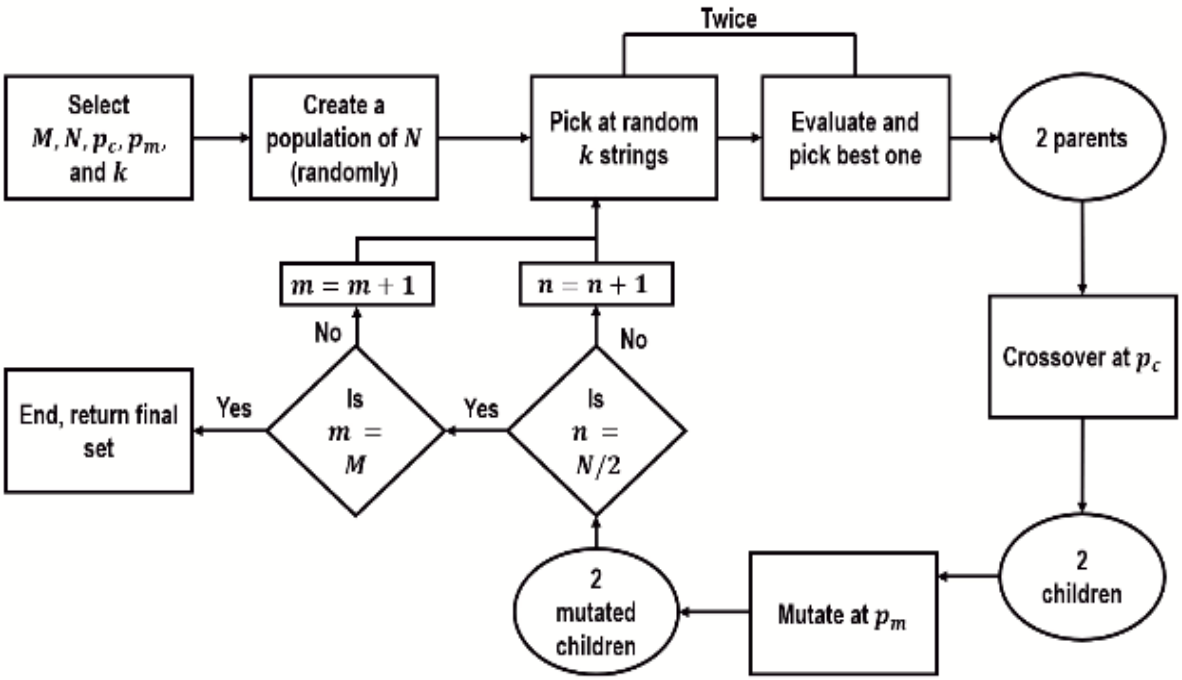


Fig 8.2.4: Genetic algorithm flowchart

M: Number of generations

N: Population size

p_c: Probability of crossover

p_m: Probability of mutation

k: Number of contestants

Number of generations, population size, crossover, and mutation probability are some of the hyperparameters of genetic algorithm.

This method tries to find the best possible feature combination by searching in a computationally efficient manner. This also means it does not search for all possible combinations. As a result, the best-performing features obtained from the genetic algorithm might not be the best combination of features. If the number of features are small, the genetic algorithm can be a really useful method. However, for high dimensional

data in which the number of features is numerous, the genetic algorithm might not be computationally efficient, and the resultant features obtained from the genetic algorithm might not be useful either.

We have earlier initialized a feature selection object in this chapter as `fsObj = FeatureSelection`. We can use the object `fsObj` for performing the genetic algorithm by using the `GeneticAlgorithm` function in the object `fsObj`. Below is how the syntax will look like.

```
best_columns = fsObj.GeneticAlgorithm(generations=20, population=75,
run_time=1200)
```

In this syntax example, we have used 20 generations and 75 population for the genetic algorithm. We will execute genetic algorithm feature selection for 1200 minutes, i.e., 20 hours and will stop the execution after that.

8.3 Simulated Annealing

Simulated annealing is an optimization algorithm whose objective is to find global optima. It is inspired by the process of annealing performed to toughen metals. For example, a piece of steel is repeatedly heated at extremely high temperatures and then slowly cooled. This process of heating and slow cooling helps molecules in the steel to improve and reorganize their position, thereby strengthening the steel. It is an improvement on the hill climbing algorithm, which is good at finding local optima, but performs poorly on global optima.

The algorithm is executed for several iterations as specified by the user. The temperature for annealing is set by the user before the start of the algorithm. For the first iteration, a random subset of features is drawn and its performance is measured for the dataset. The given feature set is perturbed by randomly removing and adding new features to the feature set. We can specify the percentage of features that can be perturbed. Usually, this percentage is kept between 1 and 5. After the features are perturbed, the new perturbed features are used for evaluating the dataset. We then compare model performance between features drawn at the beginning of iteration and perturbed features. If the new subset is better than the initial subset, we replace the initial subset with the better subset. If on the other hand, the perturbed subset performs worse than the initial subset, we calculate an acceptance or rejection probability, which uses both the model metric as well as temperature. The calculated probability is compared against a random probability. For the regression problem, if the random probability is less than accept reject probability, we take the worse-performing perturbed feature set. For classification problems, if the random probability is

more than accept reject probability, we take the worse-performing perturbed feature set. The new feature set thus generated is used for the next iteration, and the temperature is cooled based on a predefined alpha value. One thing to note about perturbing is that in an iteration, we can perturb more than once.

Simulated annealing tries to improve the feature set randomly generated in the first iteration. In some cases, the random feature set generated could be very close to global minima or global maxima. Hence, for the same feature set, in some cases, the extent of improvement obtained through simulated annealing could be worse in comparison to some other instances. Apart from the hyperparameters of the algorithm, the random feature set generated in the first iteration also has an impact. This is one of the limitations of simulated annealing, as it is beyond the control of the user.

After completion of simulated annealing, we should test the resultant feature set on external test and validation data. Figure 8.3.1 below is a flowchart of the simulated annealing algorithm [1]

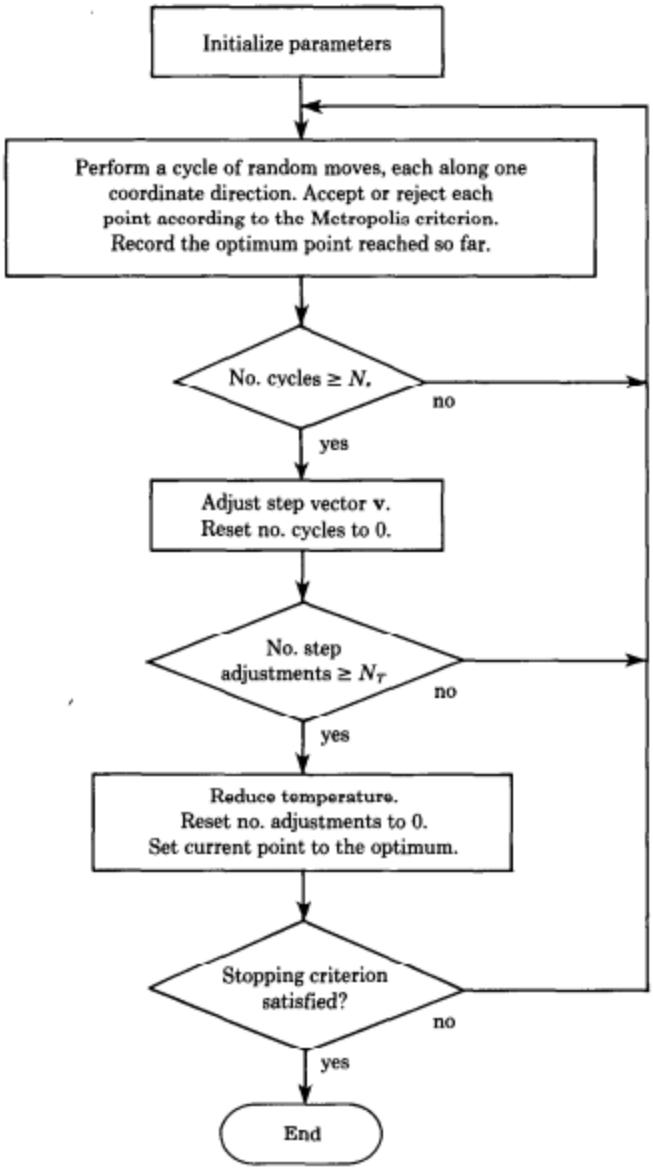


Figure 8.3.1: Simulated Annealing algorithm flowchart

We can use the function SimulatedAnnealing in the feature selection object `fsObj` for performing simulated annealing. Below is how the syntax will look like.

```
best_columns = fsObj.SimulatedAnnealing(temperature = 1500,
                                        iterations = 25,
                                        n_perturb = 75,
                                        run_time=1200,
                                        n_features_percent_perturb=1,
                                        alpha=0.9)
```

In this syntax example, we have set a high temperature of 1500. We have executed simulated annealing for 25 iterations. Within each iteration, we have perturbed 1 percent of features. We have allowed perturbation to happen for 75 times in each iteration. Execution time limit is set for 1200 minutes. We have set alpha as 0.9.

8.4 Ant Colony Optimization

Ants as insects are social by nature and live in colonies. Their objective is to survive collectively as a colony, instead of survival as an individual ant. Ants communicate with each other using sound, touch, and pheromones. Ant colony optimization (ACO) focuses on pheromones. Ants carry a portion of the food. While returning, they leave traces of pheromone. Other ants try to find an optimal path by following the pheromone trail left by other ants on the ground. ACO tries to mimic the behavior of ants to find the shortest path for food as an optimization algorithm. Figure 8.4.1 below illustrates the ACO metaheuristic algorithm.

```
procedure Ant colony optimization
Set Initialize parameters, pheromone trails
while (termination condition not met)
do
        Construct Ant Solution
        Update Pheromone Trails
        Daemon Actions
end
end
```

Figure 8.4.1: ACO metaheuristic.

In the first step, the pheromone trail and parameters are initialized. Each ant then constructs a complete solution for the problem using a pheromone trail and heuristic information.

Starting nodes, and a path between the nest and food, which passes through nodes are selected at random. Paths with a higher number of pheromone trails are given a higher probability. Ants travel between food and nest by passing through multiple nodes. This process is called a 'tour'. Each completed tour is a solution.

After pheromone trails are generated, their update is initiated. Pheromone can evaporate. As a result of evaporation, the level of pheromone in the pheromone trail can increase or decrease. Pheromone update considers this for improving the solution search. Pheromone update happens in two steps. In the first step, a fraction of the pheromone evaporates. In the second step, each ant updates the pheromone trail by depositing the amount of pheromone, proportional to how good or bad the trail is. Better solutions receive more pheromone trails, whereas worse solutions receive fewer pheromone trails.

In the next step, a new cycle is performed and this process is repeated until most of the ants follow the same tour on most of the cycles. At each iteration, the pheromone tends to concentrate within the reduced search space. The global optima are searched within the reduced search space in subsequent iterations, using the best values obtained from the new ant colony from the previous iteration.

Figure 8.4.2 below illustrates the pseudocode of ACO, and figure 8.4.3 explains all the variables used in ACO.

```
Init pheromone τ_ij ;

repeat for all ants i: construct solution(i);

       for all ants i: global pheromone update(i);

       for all ants edges: evaporate pheromone;

                              (τ_i-j := (1-ρ) τ_i-j)


construct_solution(i):

init ant;

while not yet a solution:

     expand the solution by one edge probabilistically according
     to the pheromone;

                              (τ_ρi-j / Σ_ρi-j'.τ ρi-j';)


global_pheromone_update(i):

for all edges in the solution:

     increase the pheromone according to the quality;

            (Δτ_j-j' := 1/length of the path stored)
```

Figure 8.4.2: ACO pseudocode

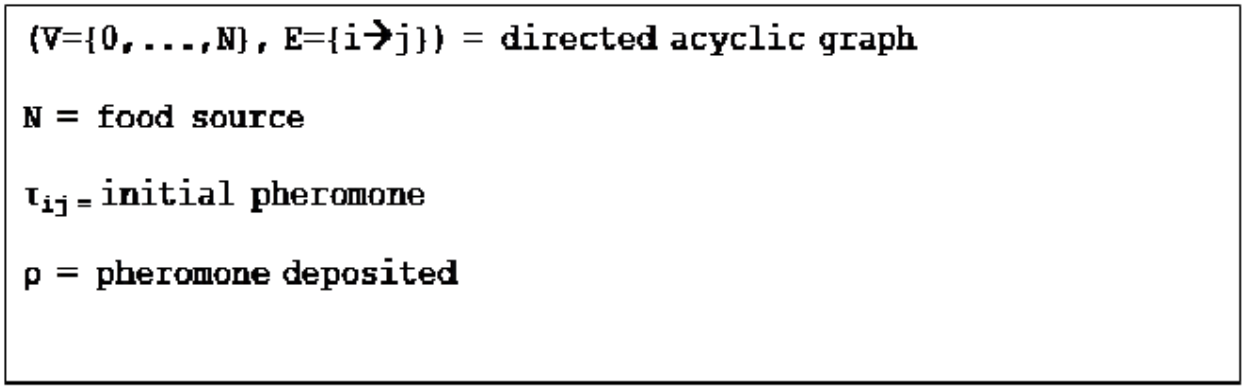


Figure 8.4.3: Variables used in ACO pseudocode

After understanding all the details of the ACO algorithm, we should remember that it is useful for a smaller number of features. For a large number of features, it may or may not give the best solution.

We can use the function AntColonyOptimization in the feature selection object `fsObj` for performing ant colony optimization. Below is how the syntax will look like.

```
best_columns = fsObj.AntColonyOptimization(iterations = 25,
                                           N_ants = 75,
                                           run_time=1200,
                                           evaporation_rate=0.8,
                                           Q=0.2)
```

In this syntax example, we have executed ant colony optimization for 25 iterations. Within each iteration, we have 75 ants. Execution time limit is set for 1200 minutes. We have set evaporation rate for pheromones as 0.8 and Q as 0.2.

8.5 Particle Swarm Optimization

The particle swarm optimization (PSO) is an optimization algorithm that tries to iteratively improve candidate solutions. Individual candidate solutions are known as particles. The population of all candidate solutions, which comprises all particle candidate solutions, is called a swarm. It tries to move the candidate particles in the search space using the position of the particle and velocity. Each particle tries to move toward the best-known position, based on the better position found by the particle itself and also the better position found by other particles collectively in the swarm. This helps the swarm to collectively move towards better solutions. This process is iteratively repeated to find the best possible solution.

It is inspired by the collective behavior of animals such as birds. For example, a swarm of birds is a swarm of insects. Let's take the example of a flock of birds. A flock of birds flies to find a place where food availability is maximum and the presence of predators is minimum. If a bird has to change its direction from its flock, it does so by changing its direction and redirecting its force in the new direction. Birds also share information amongst themselves about changes in direction. Once the best place is found by one of the swarm's members, the rest of the birds follow the direction of the first bird to change direction. By sharing information, birds of the flock together have a greater chance of survival than if they had to survive alone.

Let's try to understand the mathematics behind PSO. X is an n-dimensional vector that tries to optimize a given objective function f(X). Here n is the number of factors or variables that have an impact on the outcome. For birds, it can be the longitude and latitude of the place where they want to land. For P particles in the swarm that goes through the 't' number of iterations, the position vector can be represented as $X^t_i = (x_i1\ x_i2\ x_i3 \ldots x_in)^T$. The velocity vector can be represented as $V^t_i = (v_i1\ v_i2\ v_i3 \ldots v_in)^T$. Individual particles are denoted as 'i' and have values from 1 till P. Position and velocity vectors are updated through j dimensions with the help of the below 2 equations, where j has values from 1 till n.

Equation 1: $$V_{ij}^{t-1} = wV_{ij}^{t} + c_1 r_1^t \left(pbest_{ij} - X_{ij}^t\right) + c_2 r_2^t \left(gbest_j - X_{ij}^t\right)$$

Equation 2: $$X_{ij}^{t+1} = X_{ij}^t + V_{ij}^{t+1}$$

Equation 1 is used for updating the velocity of the particles, whereas equation 2 is used for changing the particle's positions. In the above two equations, c1 and c2 are positive acceleration constants. The constant c1 relates to previous learning by the same particle, whereas the constant c2 is a social learning constant. r1 and r2 are random values between 0 and 1 and help PSO to avoid getting stuck at local minima/maxima and increase the likelihood of finding global minima/maxima. w is inertia weight constant and is a positive number. It helps update velocity by multiplying the previous velocity and the weight constant. If it is kept above 0 but less than 1, the swarm will be able to explore more solutions in the searching domain and have more chances of getting global minima. Although, as a consequence, it can increase the computation time.

The pbest represents individual learning, whereas gbest represents global learning. If the pbest term ($pbest_{ij} - X_{ij}^t$) increases, it can help individual i^{th} particles towards the best possible position. If the gbest term ($gbest_j - X_{ij}^t$) increases, it can help find the global best of the swarm.

Figure 8.5.1 below is the flowchart of PSO [3].

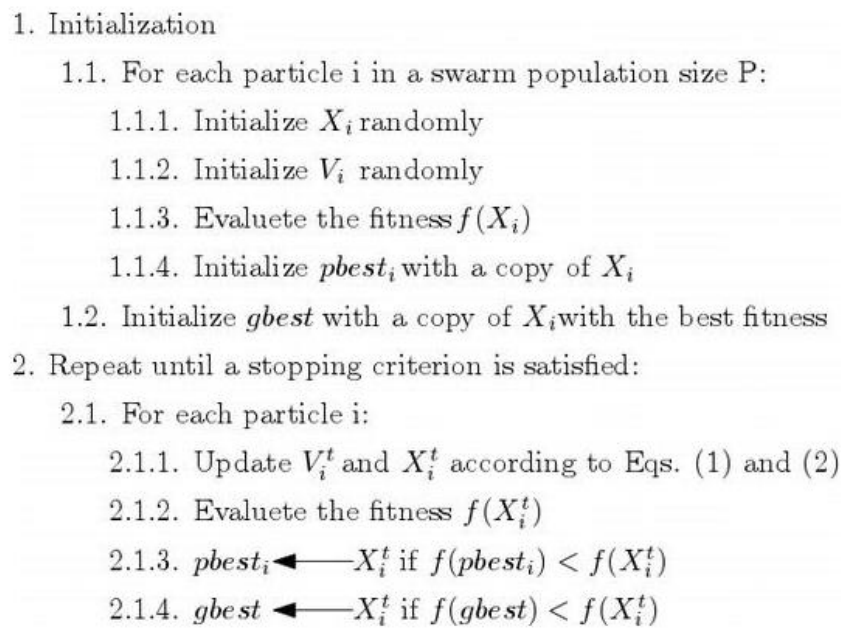


Figure 8.5.1: Flowchart of particle swarm optimization

We can use the function `ParticleSwarmOptimization` in the feature selection object `fsObj` for performing particle swarm optimization. Below is how the syntax will look like.

```
best_columns = fsObj.ParticleSwarmOptimization(iterations = 25,
                                               swarmSize = 75,
                                               run_time=1200)
```

In this syntax example, we have executed particle swarm optimization for 25 iterations. Within each iteration, we have swarm size of 75. Execution time limit is set for 1200 minutes.

8.6 Putting Everything Together

For certain models, such as Xgboost and linear models, we have used GPU for training the model. For Lightgbm, we have used a 64GB RAM machine to compute faster.

These algorithms provide the best possible combination of features. These are not necessarily the best and ideal combination of features. As the result, there might still be opportunity for improvement. If we use the feature set generated by a metaheuristic algorithm, and perform feature selection again, there might be

chance for even better performance. We will see this for hotel total room booking prediction modeling, where we perform genetic algorithm multiple times. In each new iteration of genetic algorithm, we use output set of features from previous iteration of genetic algorithm. In each iteration we try to get better performance than the previous iteration.

8.6.1 Hotel Total Room Booking

We used all 4 metaheuristic algorithms with the 3 models. Amongst all, Lightgbm and genetic algorithm performed the best. While it performed best, there might still be a chance of improving the feature combination. We used the list of features obtained from the genetic algorithm as input features and performed the genetic algorithm a few more times to refine the feature list even further. We performed this exercise a total of 5 times and saw improvement for the 2nd, 3rd, and 4th iterations. Beyond 4th iteration, there was no more improvement.

For the genetic algorithm, we used a population of 75 and executed 25 generations for the first iteration. For subsequent iterations, the number of generations was reduced to 20. Output from the iteration of the genetic algorithm was used as input for the next generation of feature selection. At each iteration of the genetic algorithm, execution time was limited to 1200 minutes.

At 1st iteration, the number of features from the genetic algorithm was 44. At the 2nd, 3rd, and 4th iterations, the number of features reduced from 44 to 20, 10, and 9 respectively. Also, at the 4th iteration, RMSE was reduced to 8.63 for test and validation datasets. RMSE for external test data also decreased to 8.79. Results from each cross-validation are presented in figure 8.6.1.

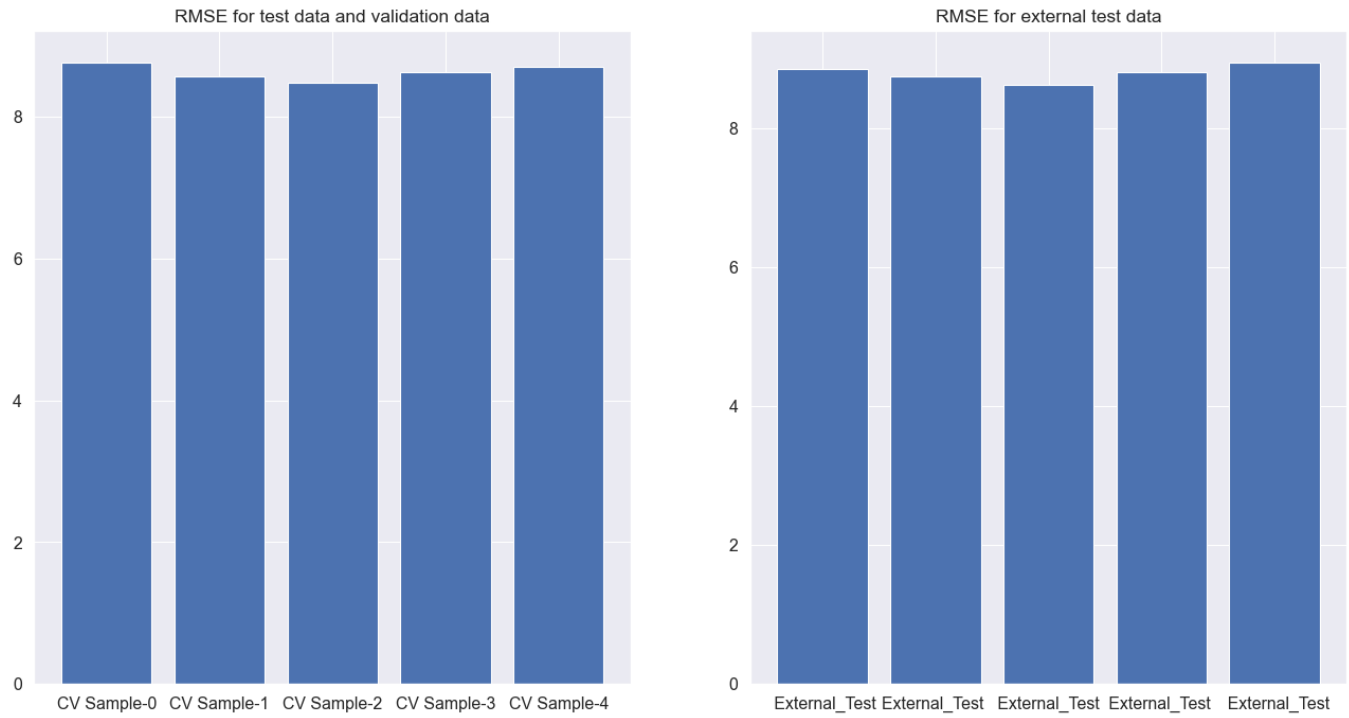


Figure 8.6.1 performance of Lightgbm tree model with genetic algorithm feature selection on cross-validation test, validation, and external test data for hotel total room booking prediction

This is the best result found so far for the hotel total rooms prediction dataset. RMSE of error is both the test datasets are very similar, and there is little to no difference in results across different cross-validations. Both of these factors suggest that the model will generalize well on unseen data.

8.6.2 Hotel Booking Cancellation

Xgboost and simulated annealing performed the best for the hotel booking cancellation dataset. It has 0.88 as precision for cross-validation test and validation dataset. For the external dataset, precision was noted as 0.93. This solution did have a downside with a low recall at 0.41 for the external test data. For simulated annealing, we used 35 iterations and 75 perturbs for performing feature selection. Figure 8.6.2.1 explains the model performance for each cross-validation.

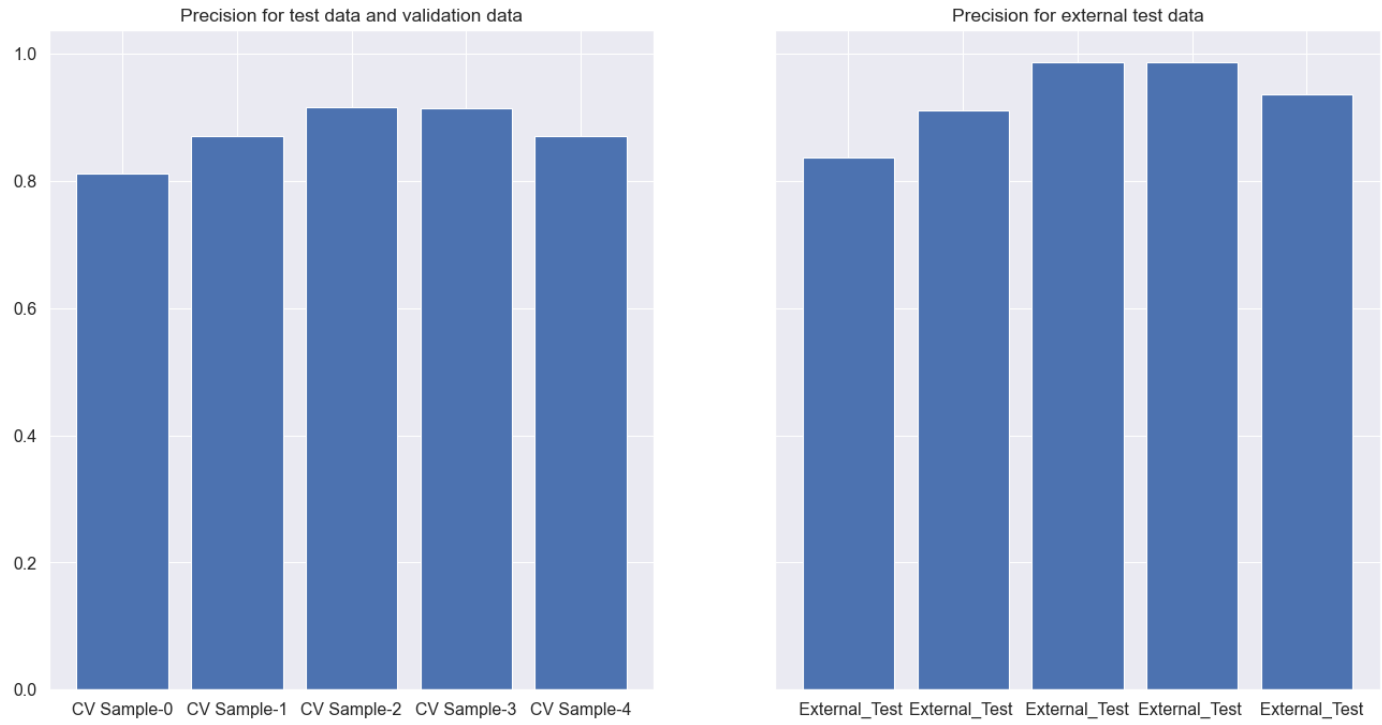


Figure 8.6.2.1 performance of Xgboost tree model with simulated annealing feature selection on cross-validation test, validation, and external test data for hotel booking cancellation

There is another solution which is the combination of Xgboost and genetic algorithm, and it performed as second best. It has 0.86 as precision for cross-validation test and validation dataset. For the external dataset, precision was noted as 0.92. This solution has a relatively better recall at 0.41 for the external test data. We used 20 generations for 75 chromosomes for the algorithm execution. Figure 8.6.2.2 explains the model performance for each cross-validation.

Both the solutions, especially the first solution is not perfect, but nearly useful. If the precision can be reliably said to be above 0.9 and close to 0.95, the hotel can use the predictions from the model for overselling the rooms. Even though the model has a lower recall. To overcome this, we can try an ensemble of two or more models to see its impact on precision and recall. Imagine a model that can predict hotel booking cancellations with a high degree of reliability. Even if it can identify only 40 percent of cancellations and cannot identify the rest of the 60 percent of the cancellations, it will still be useful for the hotel to minimize loss occurring because of 40% of cancellations.

Comparing the solutions obtained through all 4 metaheuristics algorithms, these are the best result achieved for this dataset, in comparison to other feature selection methods.

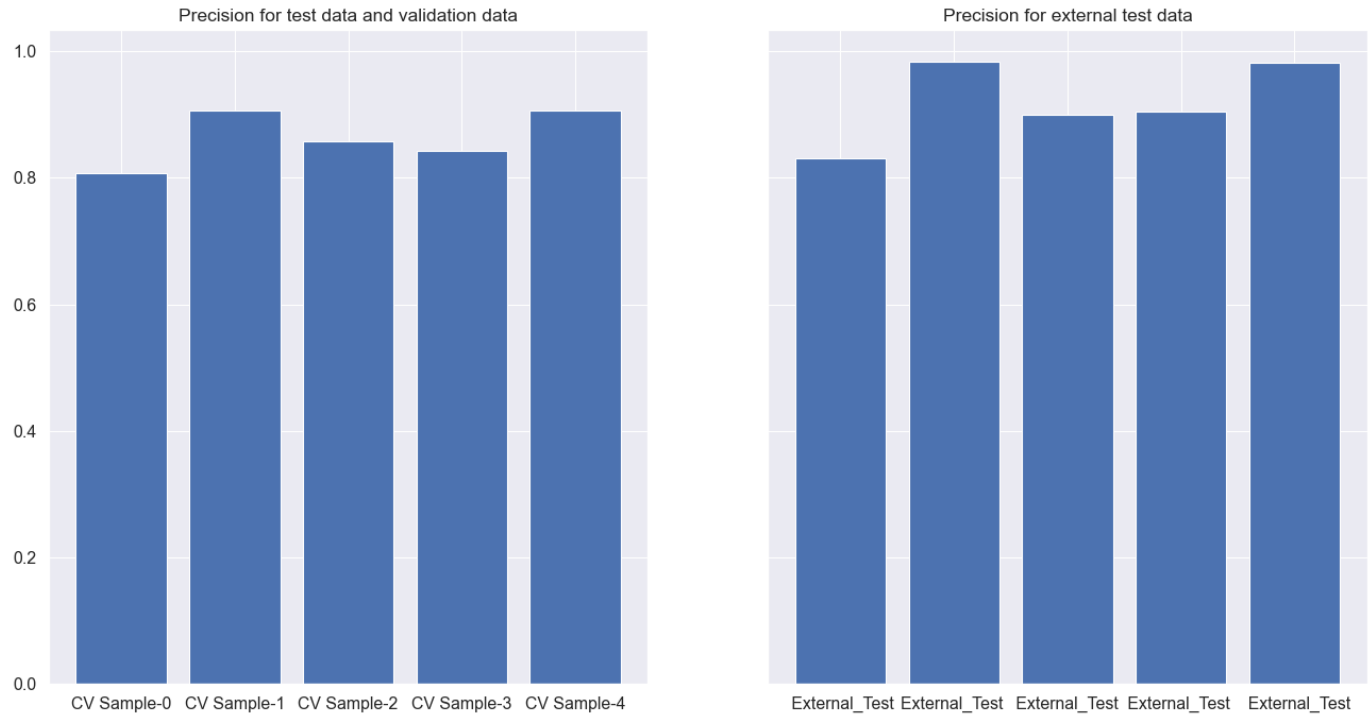


Figure 8.6.2.2 performance of Xgboost tree model with genetic algorithm feature selection on cross-validation test, validation, and external test data for hotel booking cancellation

8.6.3 Car Sales

We tried different combinations of metaheuristics algorithms and models. The best performance was achieved by the combination of Lightgbm and simulated annealing. For cross-validation test and validation data, RMSE was 197495, whereas for the external test data it is 224034. It is better than the results reported in chapter 7. For simulated annealing, we used 35 iterations with 75 perturbations.

Figure 8.6.3 shows the model performance across different cross-validations. RMSE is not very consistent across all cross-validations. RMSE is different across different test datasets. However, the major issue in this dataset is that RMSE is still very high. For this dataset, none of the feature engineering and feature selection helped us achieve a workable model that can predict the price of used cars reliably. This indicates that the dataset requires data cleaning and domain-specific feature engineering. Domain knowledge in particular can help in organizing data that can be easier for the model to learn, finding anomalies, and treating it to ensure a dataset with the least amount of noise, and finally in creating features that permeate from domain knowledge.

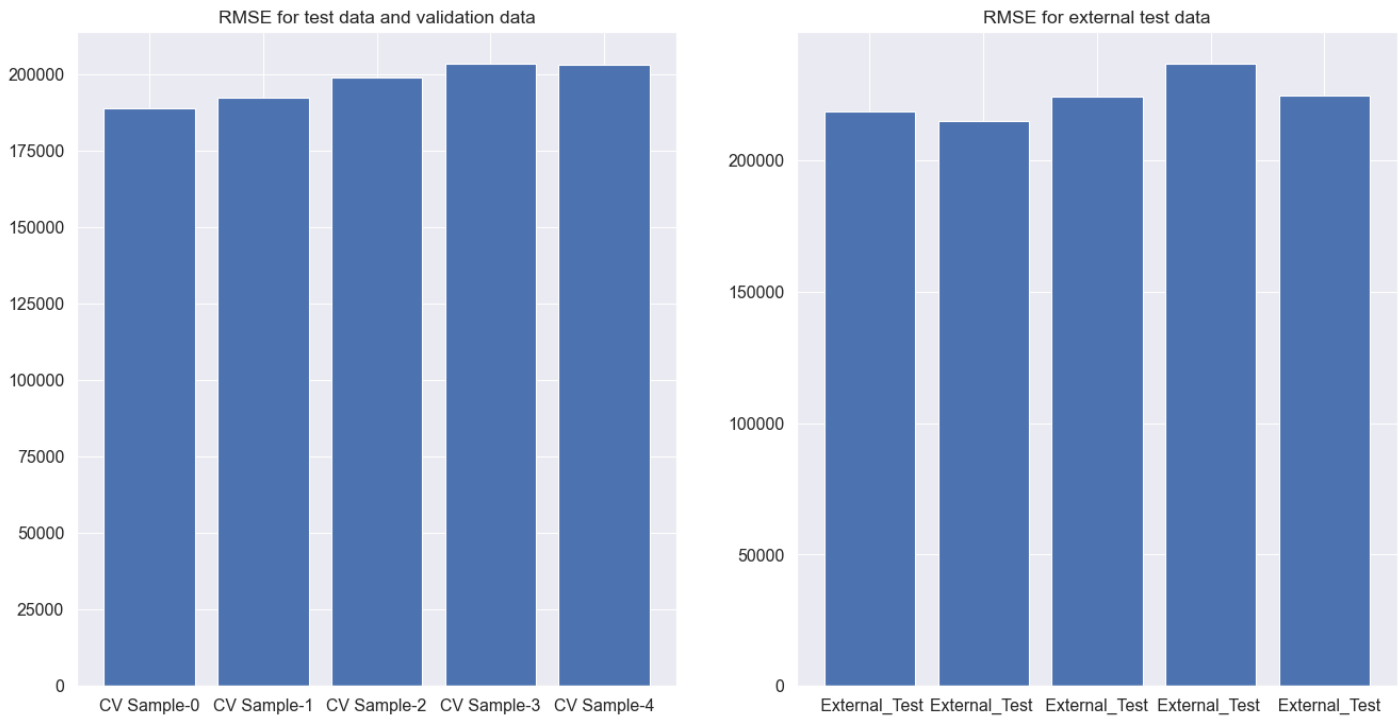


Figure 8.6.3 performance of the Lightgbm model with simulated annealing feature selection on cross-validation test, validation, and external test data for used car price prediction.

8.6.4 Coupon Recommendation

For this dataset, Xgboost performed the best with simulated annealing feature selection. It has 0.78 as precision for cross-validation test and validation dataset. For the external dataset, precision was noted as 0.79. This solution did have a downside with a low recall at 0.59 for the external test data. For simulated annealing, we used 35 iterations and 75 perturbs for performing feature selection. Figure 8.6.4 explains the model performance for each cross-validation.

This is the best performance that could be achieved for the dataset, across all feature selection methods and model techniques. However, these results are not good enough to be accepted as a reliable model. If either the precision or recall could have been close to 0.9 or higher, we could have obtained a reliable model. Hence, although model performance across different cross-validations is very similar, and there is a very small difference between the two different test datasets, we will discard this model.

In the absence of domain knowledge for this dataset, we at this point halt any further effort to improve the model for this dataset.

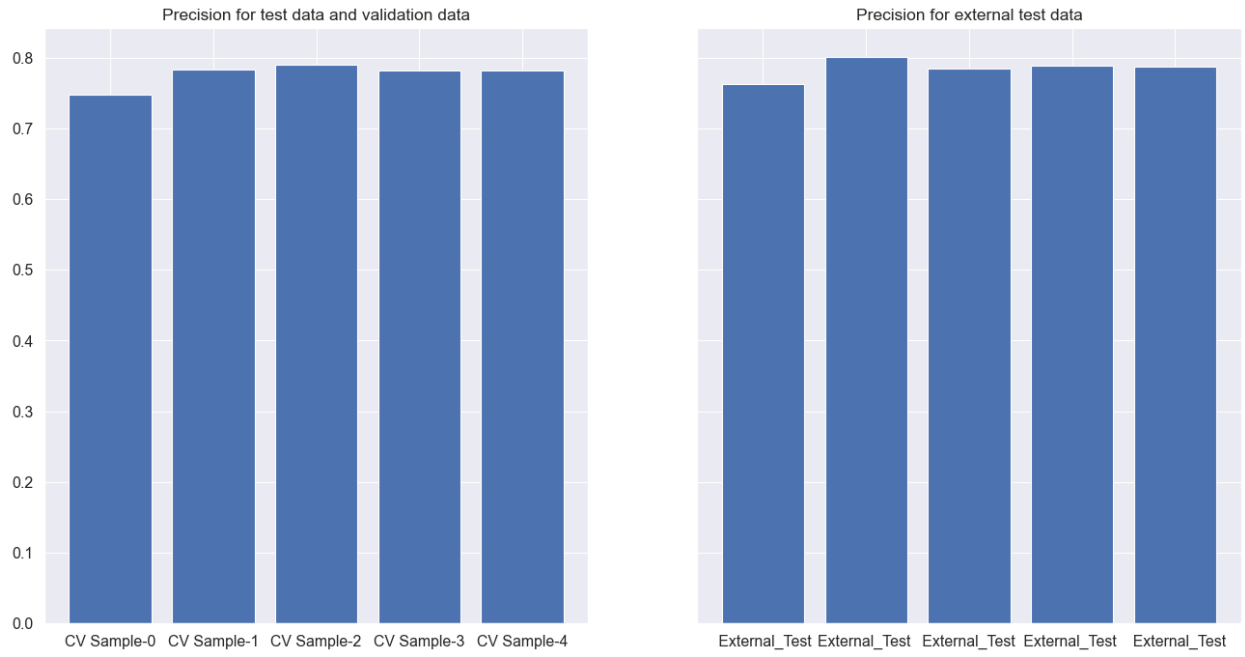


Figure 8.6.4 performance of the Xgboost model with simulated annealing feature selection on cross-validation test, validation, and external test data for coupon recommendation prediction.

8.7 Conclusion

Metaheuristics algorithms are computationally expensive. However, after comparing with similar computationally expensive methods in previous chapters, we can conclude that metaheuristics algorithms provide better solutions than methods discussed in previous chapters. Metaheuristics algorithms also require knowledge about each individual algorithm. Especially, how each parameter of the algorithm affects the search process. We have discussed these parameters for each method in this chapter, as well as shown in each worked example in accompanying Jupyter notebook.

8.8 References

[1] A. Corana, M. Marchesi, C. Martini, and S. Ridella, Minimizing Multimodal Functions of Continuous Variables with the "Simulated Annealing" Algorithm, ACM Transactions on Mathematical Software Volume 13 Issue 3Sept. 1987 pp 262–280 https://doi.org/10.1145/29380.29864

[2] Nor Farhah Binti Saidin, Chuii Khim Chong, Yee Wen Choon, Lian En Chai, Safaai Deris, Rosli M. Illias, Mohd Shahir Shamsir & Mohd Saberi Mohamad, Using Ant Colony Optimization (ACO) on Kinetic Modeling of the

Acetoin Production in Lactococcus Lactis C7, Studies in Computational Intelligence book series (DSCC,volume 477), http://dx.doi.org/10.1007/978-3-642-37137-0_5

[3] BSG de Almeida, Victor Coppo Leite, Particle Swarm Optimization: A Powerful Technique for Solving Engineering Problems, Swarm Intelligence - Recent Advances, New Perspectives and Applications, 2019 https://doi.org/10.5772/intechopen.89633

Section IV: Model Explanation

Chapter 9: Explaining Model and Model Predictions to Layman

9.1 Introduction

Interpretability for machine learning models is the degree to which a layman can understand the cause of a specific prediction.

As long as further action will be taken on prediction results that cost time or money, it will be helpful to understand the root cause behind algorithm prediction. Unless the model is for a toy dataset or if it's not going to be used for any insight or action, it is useful to have justification for model predictions. An answer could be sought for the exact pattern that the model captured, to give the specific prediction.

Even though prediction results came from a scientific and high-end algorithm, it still warrants a justification. Explainability can help identify incorrect predictions more clearly. This can again be useful during periodic model retraining. Usually, we retrain models when covariate shifts happen. We also retrain the model when the model predicts the wrong labels for a class that was under-represented in the data. After we have enough labeled data for the class that was previously underrepresented, we can consider retraining. Explainable models can help identify false positives easily, as these cases will be hard to justify. This helps in building robust and high-quality model retraining datasets to further improve the model.

Some models are inherently explainable, while others can be explained with the help of explanation techniques.

9.2 Explainable models

Explainable models have inherent attributes that explain the impact that each feature has as weights or by creating a tree structure to explain the hierarchy of relationships amongst different features. The most prominent explainable models are linear regression, logistic regression, and decision trees.

9.2.1 Linear Regression

The linear regression models can explain at an overall level, the importance of each feature through the beta coefficient.

Its functional form is $Y = \beta_0 + \beta_1 x_1 + .. + \beta_n x_n$

Y is the predicted value of the dependent or outcome variable. β_0 is a constant term, β_1 is the weight of the first feature x1 and β_n is the weight of the nth feature x_n. If features are standardized and on the same scale, we can make comparisons amongst features to identify the most and least impactful features. In addition to this, through adjusted R^2, we can ascertain to what extent the model is better than a simple horizontal line through the mean value of the dependent variable.

By changing the value in each feature, we can observe the impact on the prediction outcome. Predictions are the sum product of weights and feature values. For numerical features, values can be increased or decreased. Categorical features can be represented as binary encoded 1 or 0 dummy features and the effect of their presence or absence can be compared with the model outcome. Modeling fashion clothing involvement [1] as the outcome, we can make inferences from table 9.2.1.

Normative, Informative and Demographic Variables	Standardised Coefficients		
	β	t	p
(Constant)		3.088	.002
Normative	.410	8.496	**.000
Informative	.036	.718	.473
Age	-.165	-2.161	*.031
Marital status	.120	1.740	.083
Education	.046	.758	.449
Income	-.037	-.728	.467
R^2	.233	F=17.644	** p ≤ .01
Adjusted R^2	.220		*p ≤ .05

Dependent Variable: Fashion Clothing Involvement

Table 9.2.1 Regression output for fashion clothing involvement

Although the model didn't explain all the variance in the data as evident from a low R square, it did however indicate that normative behavior is the biggest predictor of fashion clothing involvement. Normative influences are defined as the degree to which people conform to the expectations of society. Apart from this, age is negatively associated with fashion clothing involvement. Young people tend to be more involved in fashion clothing than those who are old. To a certain extent, married people are more fashionable than those who are unmarried.

9.2.2 Logistic Regression

The logistic regression follows the sigmoid function, which takes values between 0 and 1. 1 is the desired outcome. In total, the odds of events happening and not happening is 1. The cut-off is 0.5, this is otherwise known as the ' ; ' ı boundary.

$$\frac{1}{1+e^{-z}}$$

Its functional form is

Where z is a linear regression function and is equal to $\beta_0 + \beta_1 x_1 + .. + \beta_n x_n$

Predictions are probability values. For numerical features, values can be increased or decreased. Categorical features can be represented as binary encoded 1 or 0 dummy features. The effect of change in numerical and categorical dummy binary features on the outcome variable can be observed through the change in the position of the predicted outcome for the decision boundary.

For example, if for the baseline case the predicted outcome is 0.4 and after changing the value in features predicted outcome is 0.55, we can infer that the predicted class has changed from 0 to 1. For the value 0.4, as per the decision boundary of 0.5, it will be reduced to 0 as the outcome and for the predicted value of 0.55, it will be changed to 1.

The coefficients returned by logistic regression for each feature are the log of odds. Mathematically, it can be represented as a *log (probability of event / (1-probability of the event))*. Before interpreting the coefficients of the logistic regression beta coefficient, we need to convert the log of odds into the interpretable format. This can be done by first converting the log into the exponential form using the formula odd = exponential (original form log of odd). Finally, by doing odds/(1 + odds), we can compare the coefficients of each feature if they are all standardized.

We can analyze the psychological impact of COVID-19 among primary healthcare workers through logistic regression from the below logistic regression output [2]. Table 9.2.2 has output from the logistic regression model.

Variables	Sig.	Exp (B)	95% C.I. for EXP (B)	
			Lower	Upper
Category age	.277			
Age < 34 years	.999	605.69	.000	.
Age 35-44 years	.120	2.67	.773	9.286
Age 45-54 years	.056	3.33	.972	11.431
Age ≥ 55 years	.029	4.19	1.162	15.111
Were not worried of family members getting infected by (0)	.033	.452	.218	.937
Were not worried of getting infected (0)	.032	.103	.013	.822
Constant	.011	.200		

Table 9.2.2 Regression output for psychological impact of COVID-19 among primary healthcare workers

Older healthcare workers "55 years or older" were four times more vulnerable to depression than younger workers. Healthcare workers who were neither worried for themselves nor for their families were found to be less likely to have depression disorder in comparison to those who worried for themselves and their families.

In the case of text classification, we can obtain logistic regression weights for each corresponding word feature. This will help us explain the relative importance of each word.

9.2.3 Decision Tree

The decision trees can be trained to identify different classes for a classification problem. It can also be trained for a regression problem. Training the decision tree is otherwise called growing a decision tree. It is performed through a splitting process. Adding a section to a tree is called grafting whereas cutting a tree or its node is called pruning. The dependent variable is split with the help of independent variables at nodes with the most optimal value into branches based on the best split. The best split is decided based on either of the methods such as the Gini index, information gain, or chi-square.

CART (Classification and Regression Trees) uses Gini as the method to evaluate split into data. For a pure population, the score is 0. This is most useful in noisy data. If the dependent variable has ***n*** number of classes, for each feature, the Gini impurity is calculated and the one with the lowest impurity is chosen. In the case of information gain or entropy, after calculating the information gain for each feature, the one with the highest is selected for the node splitting.

For numerical features, exclusion points are selected and values less than and higher than the exclusion point are encoded as ordinal levels. This makes the feature binary. Through n possible exclusion points, the one which gives maximum information gain or minimum Gini impurity is selected.

A study for potential diabetes mellitus created a decision tree to model potential risk for diabetes based on diet, lifestyle, and past family history of diabetes [3]. Figure 9.2.3 has the decision tree diagram.

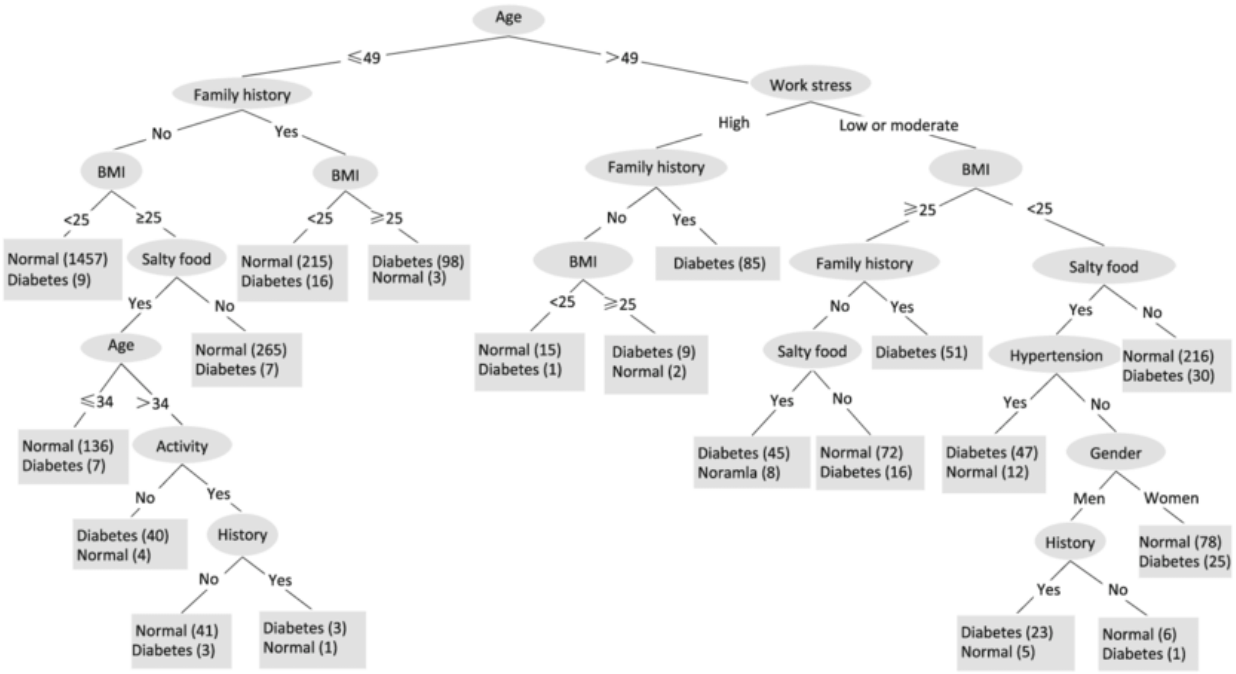


Figure 9.2.3 Potential risk for diabetes using decision tree

If we traverse from top to bottom of the decision tree across 'nodes', data is automatically filtered as a subset based on the 'and' condition. For example, let's traverse the right edge of the tree as Age(>40) -> Work Stress(High) -> Family History(Yes) -> Diabetes(85). The node and edges can be translated as people with age above 40 and who also have high work stress and family history of other family members

having been diagnosed with diabetes, it is certain that the person will have diabetes. All the 85 observations falling under this group are diagnosed with diabetes.

We can scale the Gini index or entropy at each node so that it adds to 100. It can help us compare between different subsets of data created with decision tree "AND" rules. We can narrow down to identify the most and least impactful subsets of data and rules affecting the dependent variable.

9.3 Explanation Techniques

Linear regression, logistic regression, and decision tree discussed previously, have inherent properties available to explain model behavior. Many other non-linear and complex algorithms do not have any such available property to explain model behavior. For any model to be of practical use, it is helpful to have some type of explanation of the model. The explanation is sought for both the overall model behavior and individual predictions. These explanation techniques can be applied to any machine learning technique, including explainable models.

In the following sections, 9.3.1 and 9.3.2, we will try to explain the models and individual predictions for the hotel total rooms booking predictions dataset and hotel bookings cancellations dataset. After finding the best model, the model should be trained on the whole dataset, including validation, and external test data. If, however, the performance of the model worsens, we should revert to the original training data used within the cross-validation training sample. For this chapter, we need data points to explain the model prediction. We will only keep external test data and will train the model on training and validation data.

9.3.1 Explaining Overall Model

There are methods available to explain the model as a whole. It can explain the collective nature of the model.

9.3.1.1 Partial Dependence Plot

It is otherwise known as PDP. It is performed at the feature level, one feature at a time. A predefined test data is used for PDP. For the feature under exploration, each value is processed individually. If the feature under observation is M and it has n rows, then from M_1 to M_n iteratively model predictions are obtained. In each iteration, all the values in feature M are replaced with M_i in the test data matrix, while values for

all other features are kept constant. In the next step, predicted values are obtained and averaged for M_i. Once done, averaged predictions and different values of the feature are plotted together to understand the relationship between the target and the feature.

It can identify if the relationship between the feature and the dependent variable is linear, monotonic, or more complex. More the degree of change in prediction as against the change in the feature, the more important the feature. It suffers from a few limitations. One such limitation is that it assumes that there is no correlation between different features and ignores possible feature interactions.

Let us now look at partial dependence plot in figure 9.3.1.1 for one of the features in the hotel room booking dataset 'CumulativeNumberOfRoomsNet_Quartile_Encoded'. We have used Lightgbm model for the plot.

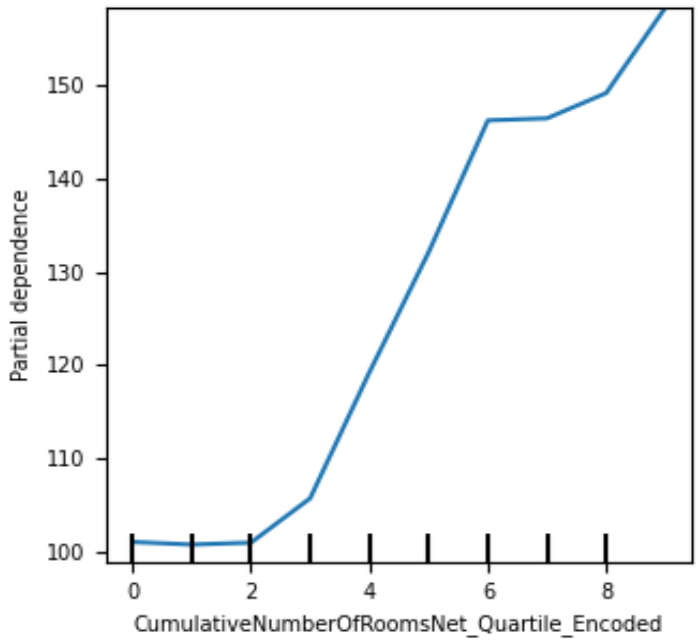


Figure 9.3.1.1 partial dependence plot of Lightgbm regression model for the hotel total room booking dataset for 'CumulativeNumberOfRoomsNet_Quartile_Encoded' feature.

We can see that relationship of the feature with the dependent variable is nearly linear in nature.

9.3.1.2 Accumulated Local Effects Plot

It is abbreviated as ALE plot. It overcomes a major disadvantage of a partial dependence plot and can work even when features are correlated.

It studies the effect of the feature at a certain level against average predictions. At a specific value for a feature, it will suggest to what extent prediction is higher or lower than average prediction. It takes the quantile distribution of features or specified intervals within the domain of the given feature to define intervals. This enables comparison among different features.

Now let us now look at the ALE plot in figure 9.3.1.2 for the same scenario we discussed earlier, for the hotel room booking dataset.

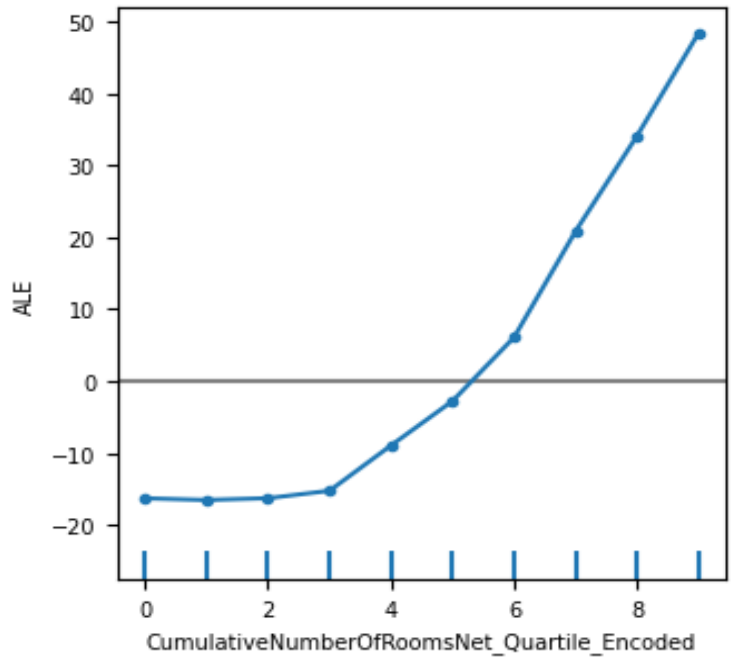


Figure 9.3.1.2 accumulated local effects plot of Lightgbm regression model for the hotel total room booking dataset for 'CumulativeNumberOfRoomsNet_Quartile_Encoded' feature.

As it considers quantile distributions, plot is smooth at the top end. Very high values that stands out in PDP is smoothened for the ACE plot. It looks more stable and easier to interpret, in comparison to partial dependence plot.

9.3.1.3 Permutation Feature Importance

This method uses prediction error as a marker of feature importance. The List of values from the feature is selected based on permutation and shuffling. The Prediction error is obtained for permuted values. If prediction error increases for permuted shuffled values of a feature, it is considered important.

This is done by obtaining original errors for specific test data as the first step. In the second step, for each feature permutation error is obtained by selecting feature values based on shuffling and permutation while keeping the value of other features constant. The permutation feature importance quotient for each feature is calculated as the original error divided by the permutation error of the feature. Finally, the permutation feature importance quotient is sorted in descending order for all features together to obtain the most and least important features in descending order.

Although it takes into account all types of feature interactions amongst different features, if some features are correlated, it can decrease the importance of correlated features.

9.3.1.4 Surrogate Model

Through a surrogate and easy-to-explain model such as linear or logistic regression, we can try to explain a black-box model. For being able to do this, 3 datasets are needed. The first dataset is black-box model training data, which is used for training the black-box model. The second dataset is surrogate model training data and the third dataset is test data.

After the black-box model is trained using black-box model training data, features from surrogate training data are used and predicted values are obtained for surrogate training data and used as a dependent variable of the surrogate model. The surrogate model is trained using predicted labels from the black-box model as the dependent variable and features from the surrogate training dataset. Finally, both the black-box model and surrogate model are used for generating predictions for test data. If the performance of the black-box model and surrogate model is similar and the R-square of the surrogate model is acceptable, then the surrogate model is used for explaining the black-box model.

9.3.2 Explaining Individual Predictions

Understanding how the model performs through the explanation of the overall model is useful. There are methods available for explaining individual predictions from the model as well. These techniques can be

useful when we have to probe the root cause behind certain predicted values from the model at an individual level.

9.3.2.1 Individual Conditional Expectation Plots

It is otherwise known as the ICE plot. It is related to the partial dependence plot method. However, it differs in the aspect that plots are generated for individual values instead of averages. One line in ICE represents one sample. It can help us understand the pattern of change in prediction concerning change in a feature.

Figure 9.3.2.1 has the ICE plot for the 'CumulativeNumberOfRoomsNet_Quartile_Encoded' discussed earlier. We can see that the linear relationship explored between the feature and dependent variable is not always true. In many cases it is polynomial. As we can see the total rooms sometimes increase and then decrease between different quartiles of the feature.

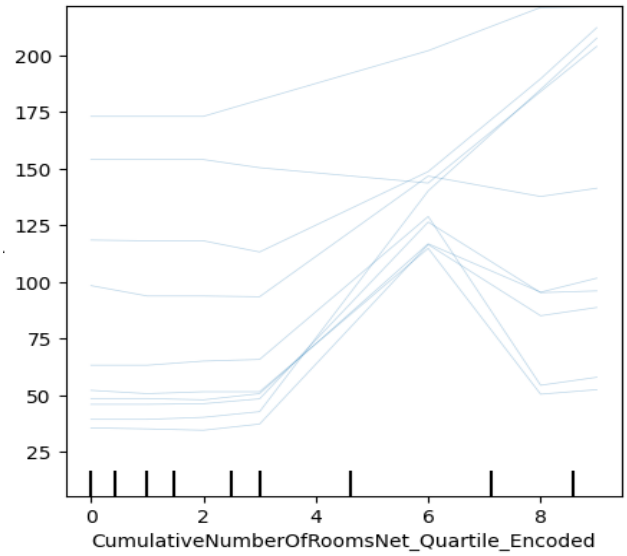


Figure 9.3.2.1 Individual Conditional Expectation plot of Lightgbm regression model for the hotel total room booking dataset for the first 10 rows of external test data, for the feature 'CumulativeNumberOfRoomsNet_Quartile_Encoded'

9.3.2.2 Local interpretable model-agnostic explanations

It is otherwise known as LIME. It uses an explainable model to explain individual predictions of a black-box model. To explain a specific prediction from the black-box model, a perturbed sample dataset is created. To explain a specific predicted value from the black-box model, all the values from the feature matrix are taken and randomly changed for different features. A new dataset is created from this exercise. For this dataset, prediction from the black-box model is obtained.

Perturbed samples are weighted based on proximity to the original feature values which we are trying to explain. The predicted value from the black-box model is used as the dependent variable and weighted perturbed feature values as features for training an explainable model.

Finally, the instance we were trying to explain from the black-box model is explained through an explainable model.

Figure 9.3.2.2 displays the LIME plot of Lightgbm regression model. We have used 4th row of external test data from the hotel total room booking dataset.

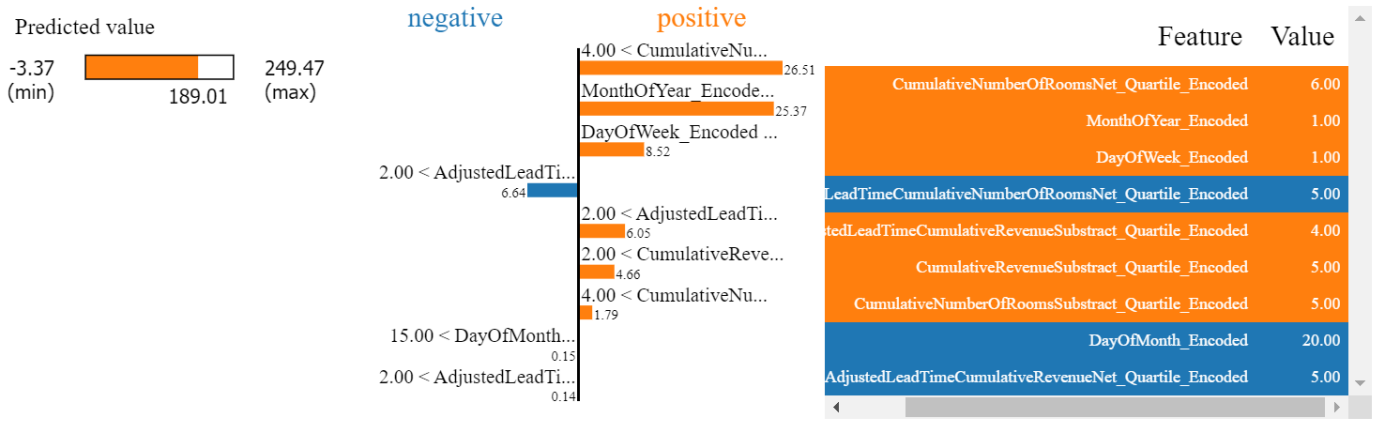


Figure 9.3.2.2 LIME plot of Lightgbm regression model for the hotel total room booking dataset for the 4th row of external test data.

This plot has 3 subplots. First subplot has the predicted values and third subplot has actual feature names and values. The second subplot has a negative and positive relationship indicator against each feature. For example, for the '*DayofWeek_Encoded*' feature, total rooms increase in demand for days that are farther from Monday. Similarly, for the '*AdjustedLeadTimeCumulativeNumberOfRoomsNet_Quartile_Encoded*'

feature, it has a negative relationship with total room demand. This is the interaction between lead time and the net number of rooms quartile feature. The second part of the plot also suggests the current value for the feature, against a threshold set by the model. For example, the '*DayOfMonth_Encoded*' feature, has a negative relationship with the total rooms sold for a check-in date. I.e. Total number of rooms is sold more towards the beginning of the month, and then gradually decreases as the month passes. Here the value is 20, which is higher than the set threshold of 15, and the check-in date for which the model has predicted is farther in the month.

9.3.2.3 Counterfactual Model Explanations

The counterfactual model explanation is a way of explaining a model where the smallest change in a feature is compared against a noticeable change in the predictable outcome. To understand "Noticeable outcome", let's take an example of a model which predicts if someone has diabetes or not by using daily minutes of exercise, a binary-coded feature for a family history of diabetes, age, and a binary-coded feature for the stressful job. For someone who exercises for 45 minutes, with no family history of diabetes, who is 40 years old, and who has no stress job, the model prediction outcome came as non-diabetic. However, by only changing jobs as stressful, the prediction came as diabetic. In this case, the noticeable outcome is changing between the different classes of diabetic vs non-diabetic.

Similarly, let's take a regression prediction problem. We are predicting someone's income potential based on age, highest qualification, and distance from the nearest metropolis. If the age is below 30, the highest qualification is a bachelor's and distance is 200 miles, predicted income is $40000. However, if we reduce the distance from the metropolis to 10 miles, income changes to $65000. In this case, an additional income of $25000 is 62.5% more than the previous salary. It can be considered a "noticeable outcome".

This method tries to identify the smallest change to the features which will bring noticeable outcomes. However, these changed feature matrices should be similar to the original instance we are trying to explain. Minimal changes should be present in additional instances we are using to explain the original instance. These instances are used for explaining the original instance. It will be explained in the lines "if we make 'x' change in 'm' feature, the outcome will change noticeably", as a counterfactual.

Figure 9.3.2.3a and 9.3.2.3b has the Counterfactual plot of Lightgbm regression model for the hotel total room booking dataset. We have plotted the 4th row of external test data.

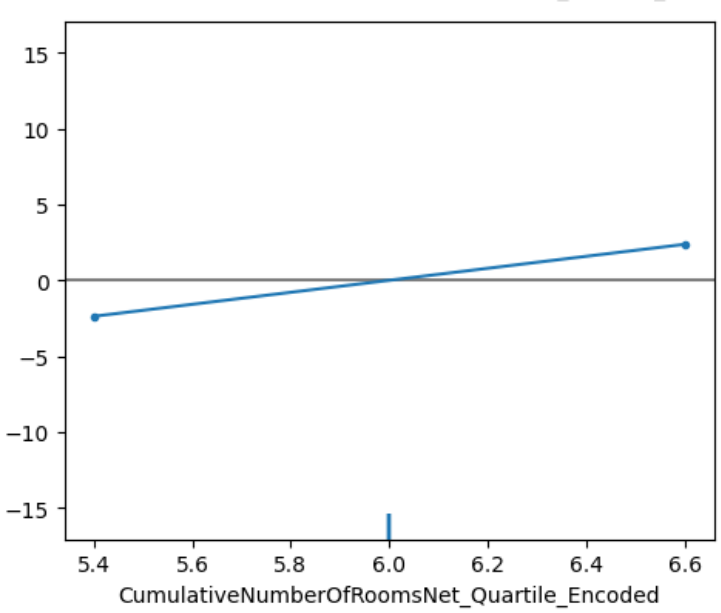


Figure 9.3.2.3a Counterfactual plot of Lightgbm regression model for the hotel total room booking dataset for the 4th row of external test data, for the feature CumulativeNumberOfRoomsNet_Quartile_Encoded feature.

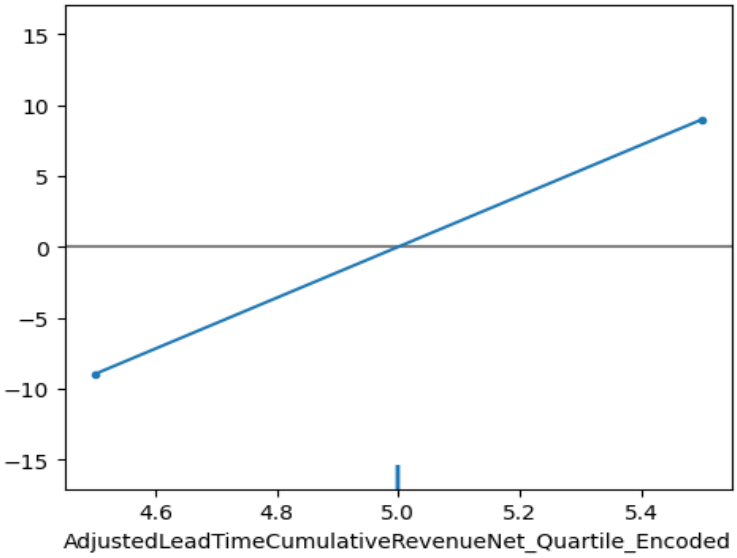


Figure 9.3.2.3b Counterfactual plot of Lightgbm regression model for the hotel total room booking dataset for the 4th row of external test data, for the feature AdjustedLeadTimeCumulativeRevenueNet_Quartile_Encoded.

Previously, we have seen that the feature ‘CumulativeNumberOfRoomsNet_Quartile_Encoded’ has the highest impact on overall model. For this instance of data, this has a relatively stable and lower contribution for the output. Even if we changed the values of the feature, the impact on outcome was marginal. In contrast, for the feature ‘AdjustedLeadTimeCumulativeRevenueNet_Quartile_Encoded’, if we slightly changed the values, the impact on output was higher than previously thought.

9.3.2.4 SHAP

SHAP is the contribution of each feature towards the predicted outcome from the model. To calculate the SHAP value, in the first step model performance is obtained for a sample dataset. In the second step, the importance of individual features is obtained by giving different values to the model and observing whether model performance increases or decreases. It can be positive or negative. To identify the most to least impactful features, the absolute value of SHAP is considered.

We can obtain SHAP feature importance for each observation in the feature matrix. This can help us interpret the model globally by analyzing and summarizing the SHAP values in each observation for each feature.

Now let us look at the SHAP model explanation for the 4th row of external test data for Lightgbm regression in figure 9.3.2.4

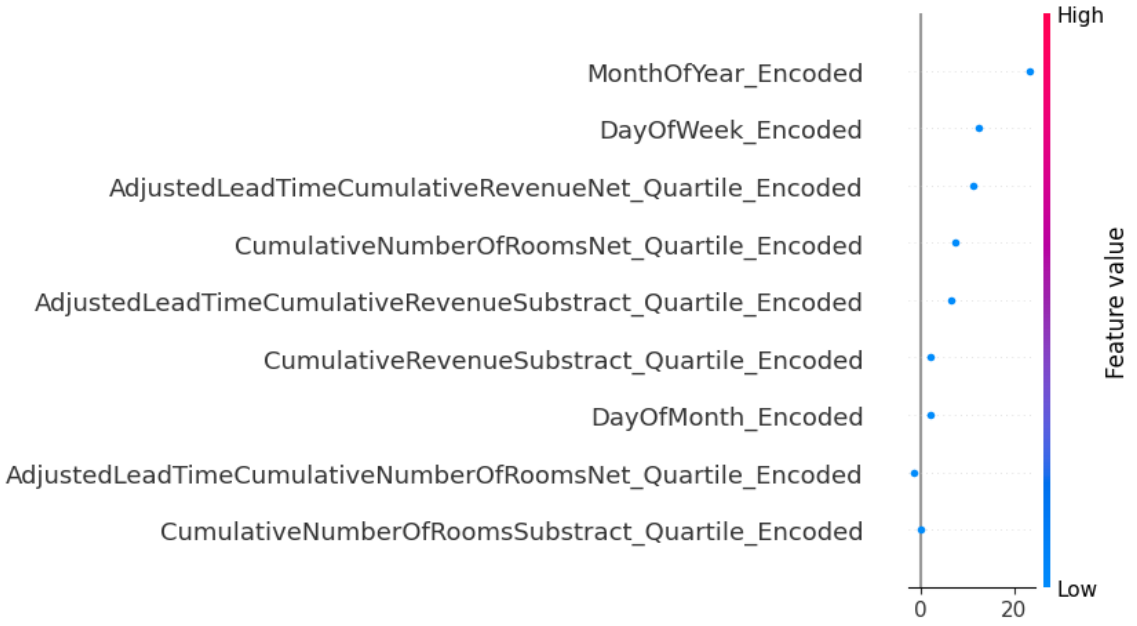


Figure 9.3.2.4 SHAP plot of Lightgbm regression model for the hotel total room booking dataset for the 4th row of external test data.

We can see the most impactful feature is 'MonthofYear_Encoded'. This is followed by 'DayOfWeek_Encoded'. The feature 'AdjustedLeadTimeCumulativeRevenueNet_Quartile_Encoded' was found to be one of the impactful features in counterfactual model explanation method. SHAP found this to be the third most impactful feature for the 4th sample of external test data.

9.4 Putting Everything Together

After understanding different methods of model explainability, now let us try to apply the methods for the hotel room booking prediction, and hotel booking cancellation datasets.

9.4.1 Hotel Total Room Booking

We tried all 4 methods of explaining the Lightgbm model, partial dependence plot, accumulated local effects plot, permutation feature importance, and surrogate model. For these two datasets, we were able to create acceptable level of model performance through metaheuristics feature selection.

We tried a surrogate linear regression model. However, the RMSE of the model was more than 30. Hence, we will not include the surrogate model for understanding the Lightgbm regression model.

Permutation feature importance is the easiest to interpret, as it gives the features in decreasing order of importance for the model. Let us start with this method. This can be seen in figure 9.4.1.1.

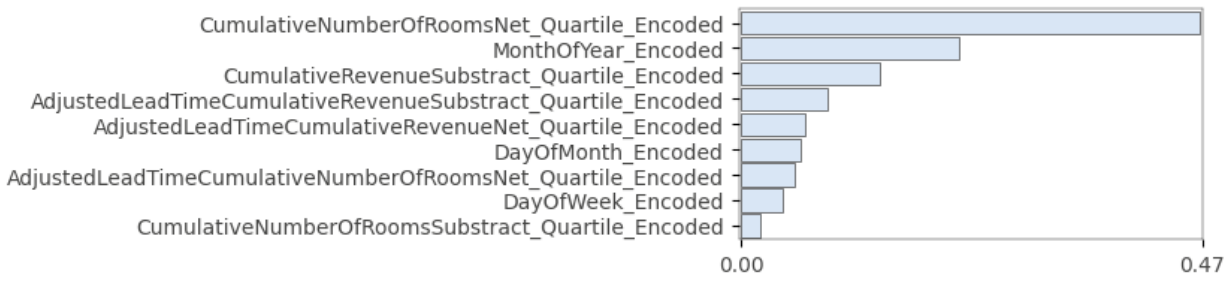


Figure 9.4.1.1 permutation feature importance plot for Lightgbm regression model for the hotel total room booking dataset.

The first feature is a higher order feature of cumulative rooms sold for the hotel, for a specific check-in date, at a lead time. Total rooms to be sold does have an impact on seasonality, as evidenced by the second most important feature, which is the month of the year encoded feature.

To a layman, we can explain that sold rooms inventory for a check-in date, and the monthly seasonality of the booking demand have the biggest impact on total room demand for a check-in date. This can help the machine learning engineer to speak in layman's terms and convince the users to adopt the model.

We will now look at the partial dependence plot for the Lightgbm regression model in figure 9.4.1.2.

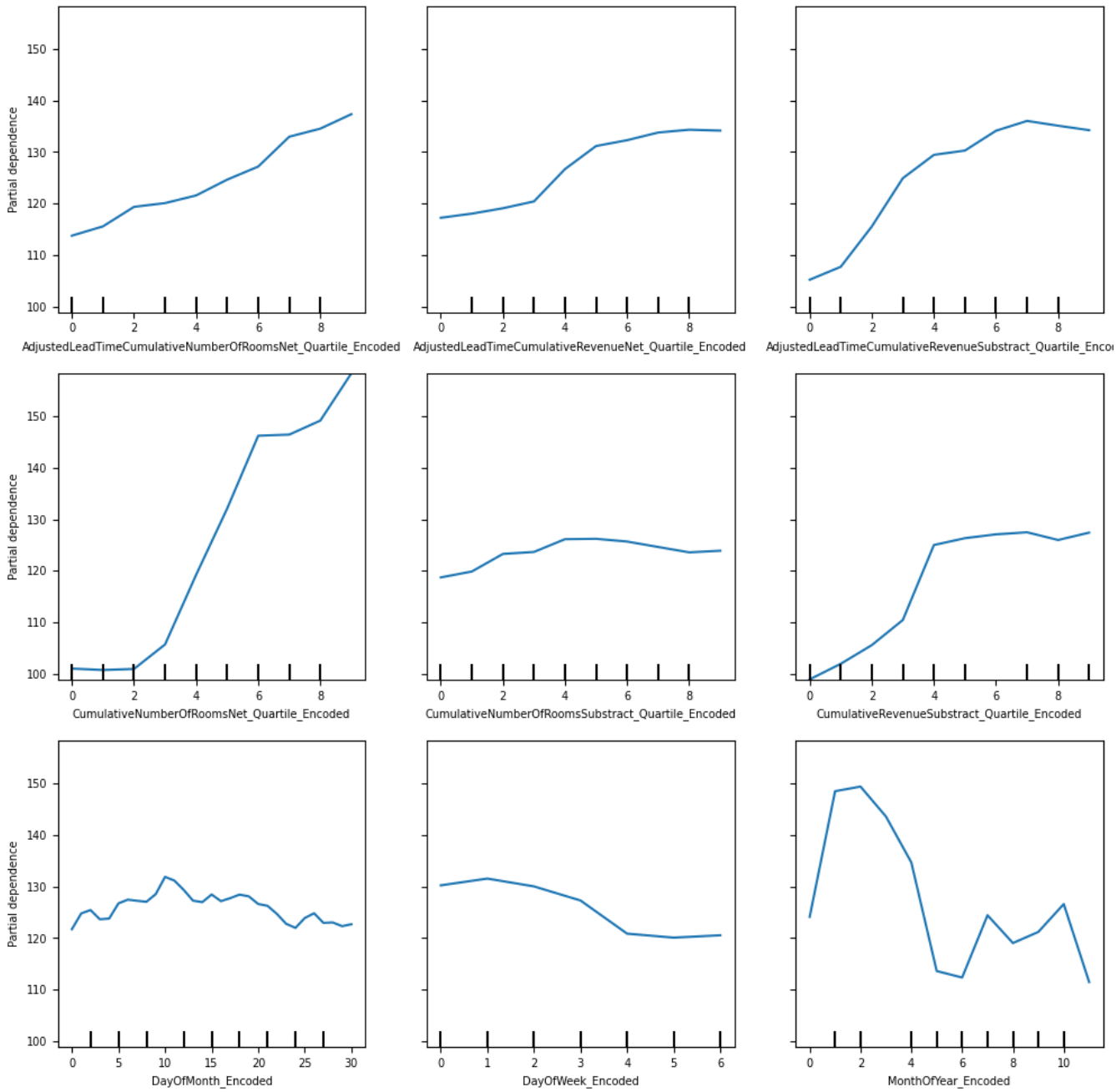


Figure 9.4.1.2 partial dependence plot of Lightgbm regression model for the hotel total room booking dataset.

The most impactful features have the sharpest curves. The most important is the plot represented in the second row, first column. This is a higher order feature of the cumulative number of net rooms sold. This is an almost linear relationship. The second most impactful feature is the last subplot in the third row and third column. This is an encoded feature of the month feature. The months are encoded from 0 to 11.

Since this is a categorical feature, it will not be appropriate to deduct a conclusion based on the shape of the relationship. We can however conclude the different levels of hotel reservations for different months. Now let us look at the accumulated local effects plot for the same dataset in figure 9.4.1.3.

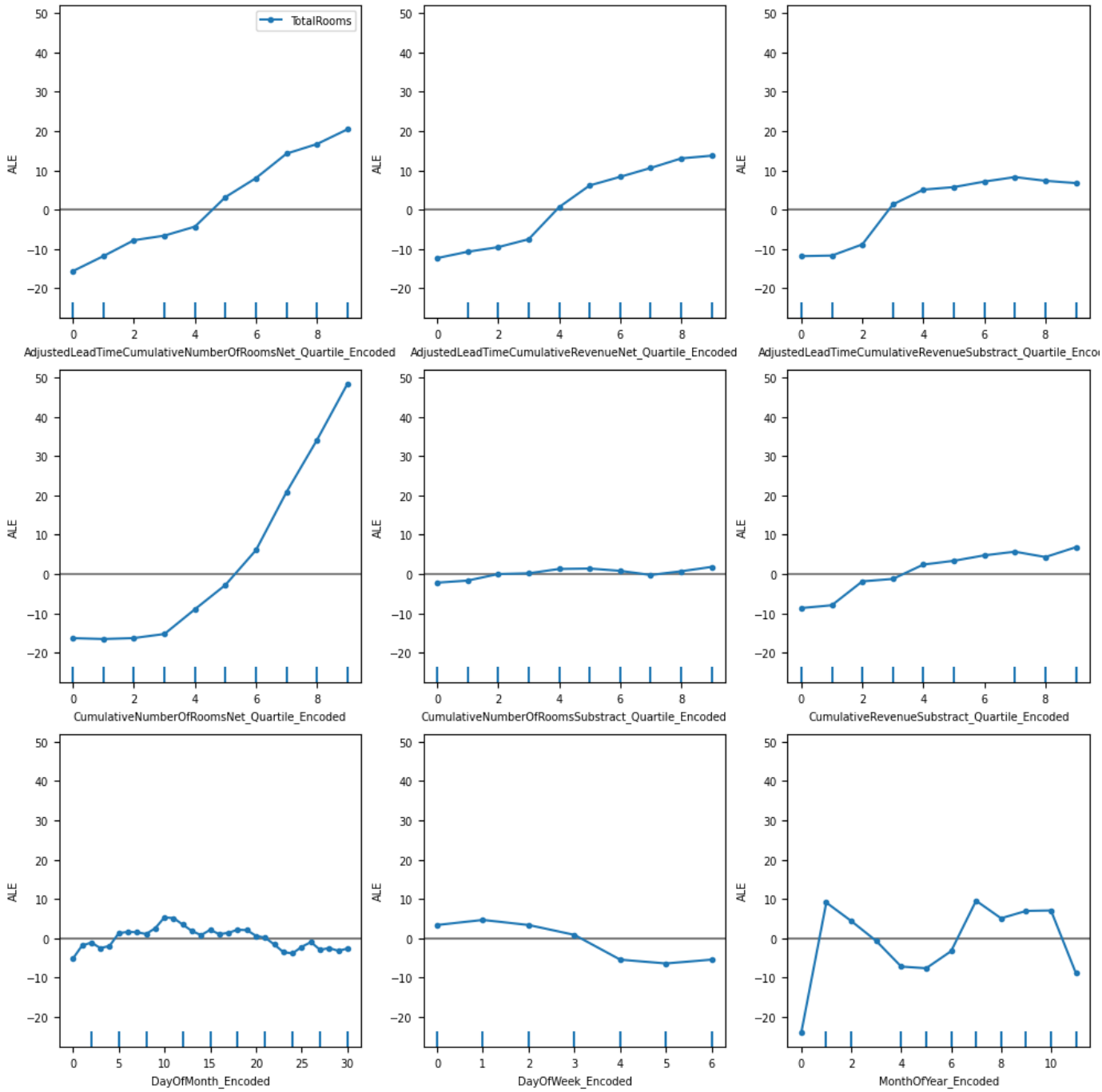


Figure 9.4.1.3 accumulated local effects plot of Lightgbm regression model for the hotel total room booking dataset.

The accumulated local effects plot explains the model performance after accommodating the correlation among features. We can see that the most important feature is the same as it was in the partial dependence plot. For the second most important feature, the extent of the impact is less. However, it still has the second-highest sharp changes for different values of the feature.

In addition to the inferences drawn from the three plots, we see that there is very little difference between the partial dependence plot and the accumulated local effects plot. There is one advantage with the accumulated local effects plot, as it overcomes the disadvantage of a partial dependence plot, which cannot work with correlated features. Although the partial dependence plot and accumulated local effects plots carry more information than the permutation feature importance plot, the latter is more legible and easier to read if the model has a huge number of features.

For the hotel bookings cancellations dataset, we will restrict our investigation for overall model explanation to accumulated local effects plot, and permutation feature importance. For the rest of this section, we will discuss explaining individual predictions of hotel total room booking prediction.

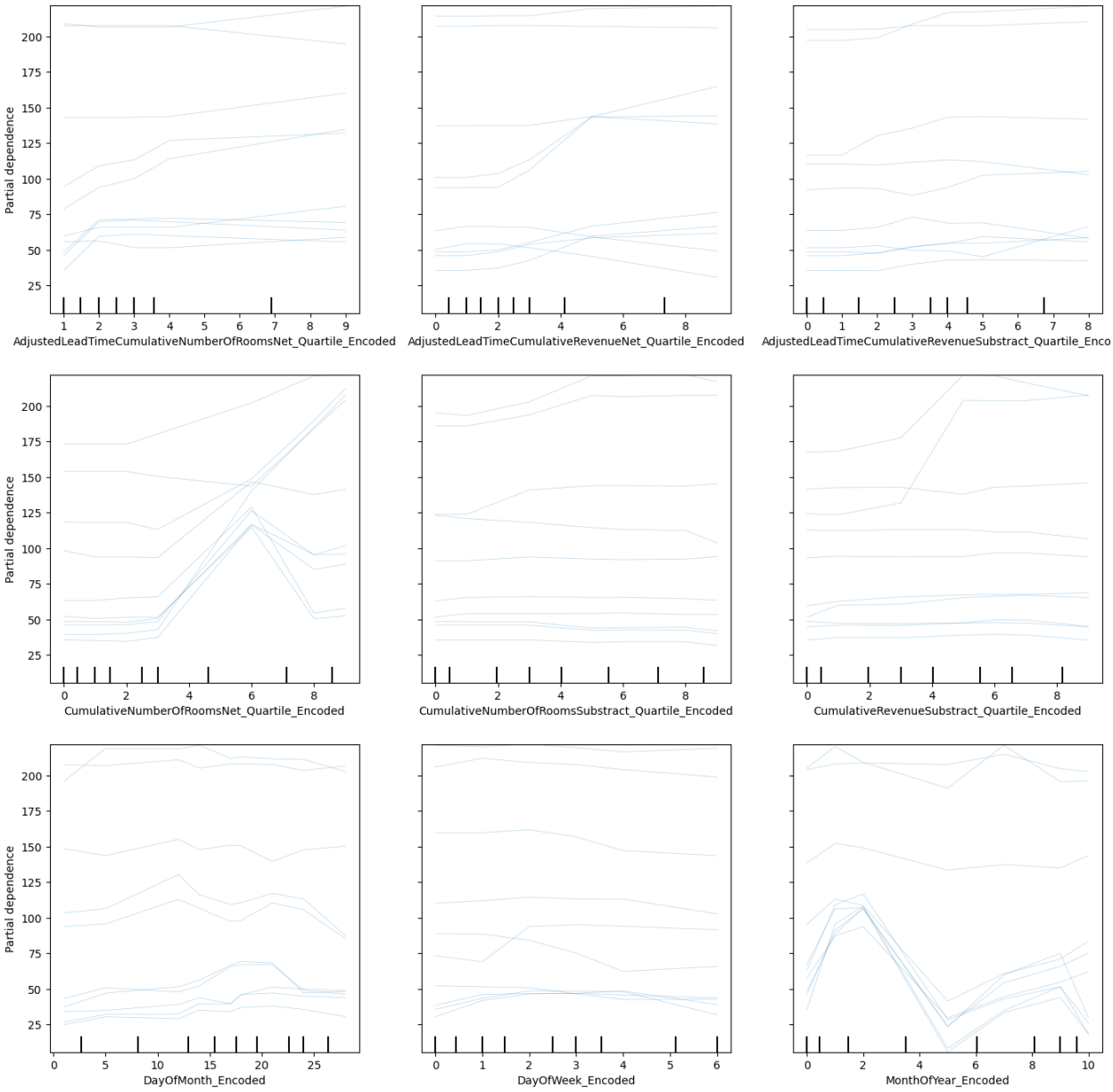


Figure 9.4.1.4 Individual Conditional Expectation plot of Lightgbm regression model for the hotel total room booking dataset for the first 10 rows of external test data.

The ICE plot in figure 9.4.1.4 suggests that there is a degree of non-linearity between the feature '*CumulativeNumberOfRoomsNet_Quartile_Encoded*' and the dependent variable. For many cases, the number of total bookings increases as we move up towards the higher quartile of the number of net cumulative rooms sold. However, in some cases, it decreases after increasing for a short while. Hence the relationship could be non-linear.

Let us now look at the LIME interpretation of a single row of data from the 4th index of external test data, as displayed in figure 9.4.1.5.

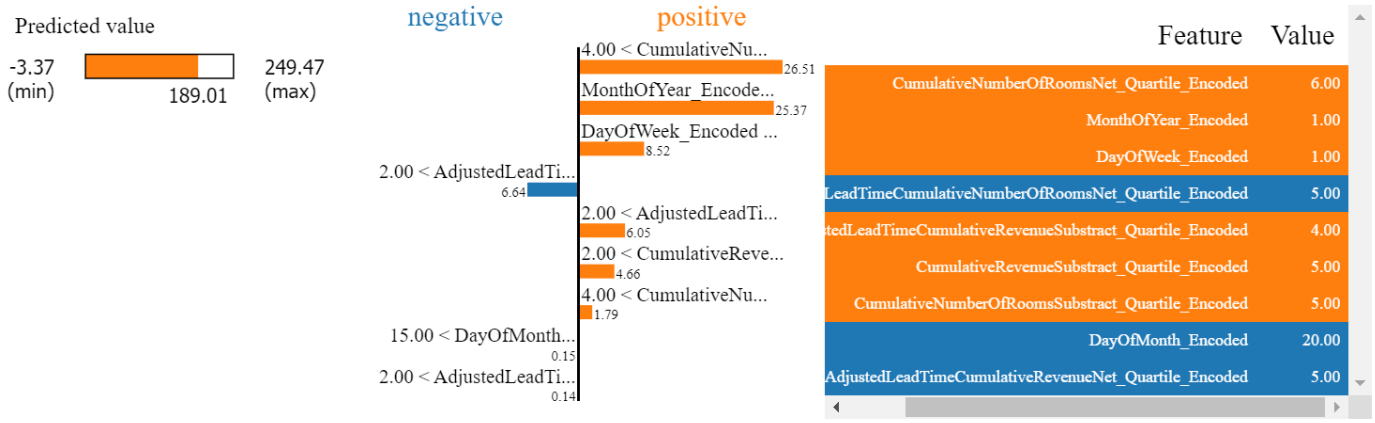


Figure 9.4.1.5 LIME plot of Lightgbm regression model for the hotel total room booking dataset for the 4th row of external test data.

The above plot has 3 parts. Let us understand the first part. The predicted value displays higher values in orange color and smaller values in blue color. The prediction from model 189.01 is a high value. The second subplot has a negative and positive relationship indicator against the feature. For example, for the '*DayOfWeek_Encoded*' feature, total rooms increase in demand for days that are farther from Monday. Similarly, for the '*AdjustedLeadTimeCumulativeNumberOfRoomsNet_Quartile_Encoded*' feature, it has a negative relationship with total room demand. This is the interaction between lead time and the net number of rooms quartile feature. The second part of the plot also suggests the current value for the feature, against a threshold set by the model. For example, the '*DayOfMonth_Encoded*' feature, has a negative relationship with the total rooms sold for a check-in date. I.e. Total number of rooms is sold more towards the beginning of the month, and then gradually decreases as the month passes. Here the value is 20, which is higher than the set threshold of 15, and the check-in date for which the model has predicted is farther in the month.

The third part of the plot is a table and simply denotes each feature in orange and blue color, depending on whether the feature has a positive or negative relationship with the dependent variable. The second column of the table shows the actual values of the feature for the specific row.

Now let us look at the counterfactual model explanation for the same observation in 4th row, in figure 9.4.1.6.

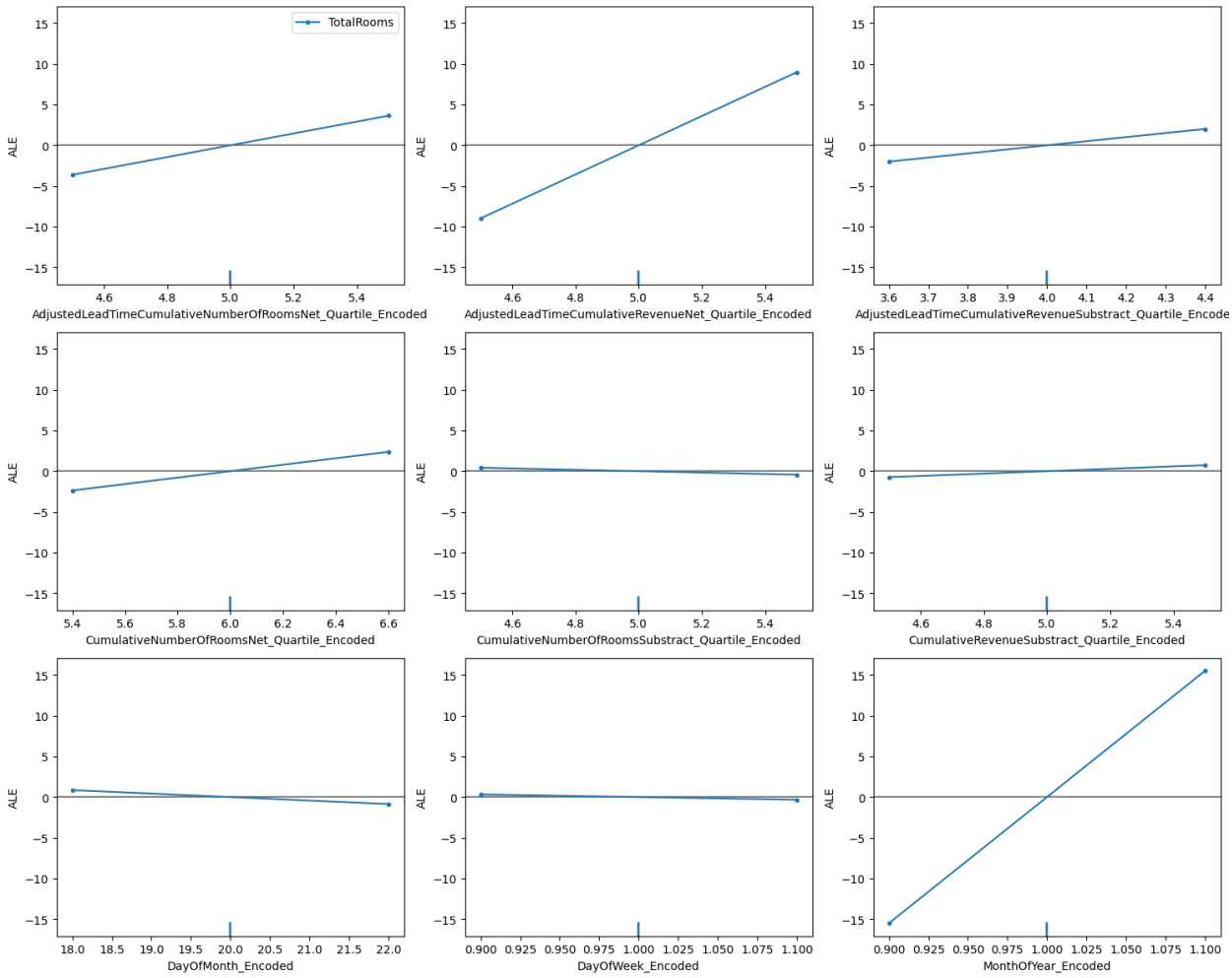


Figure 9.4.1.6 Counterfactual plot of Lightgbm regression model for the hotel total room booking dataset for the 4th row of external test data.

We can clearly see that month of the year encoded feature has highest impact, as identified by the counterfactual plot. A small change the value of month can bring a drastic change in the model prediction. This was followed by the '*AdjustedLeadTimeCumulativeRevenueNet_Quartile_Encoded*' feature.

Now let us look at the SHAP model explanation for the same observation in 4th row, in figure 9.4.1.7.

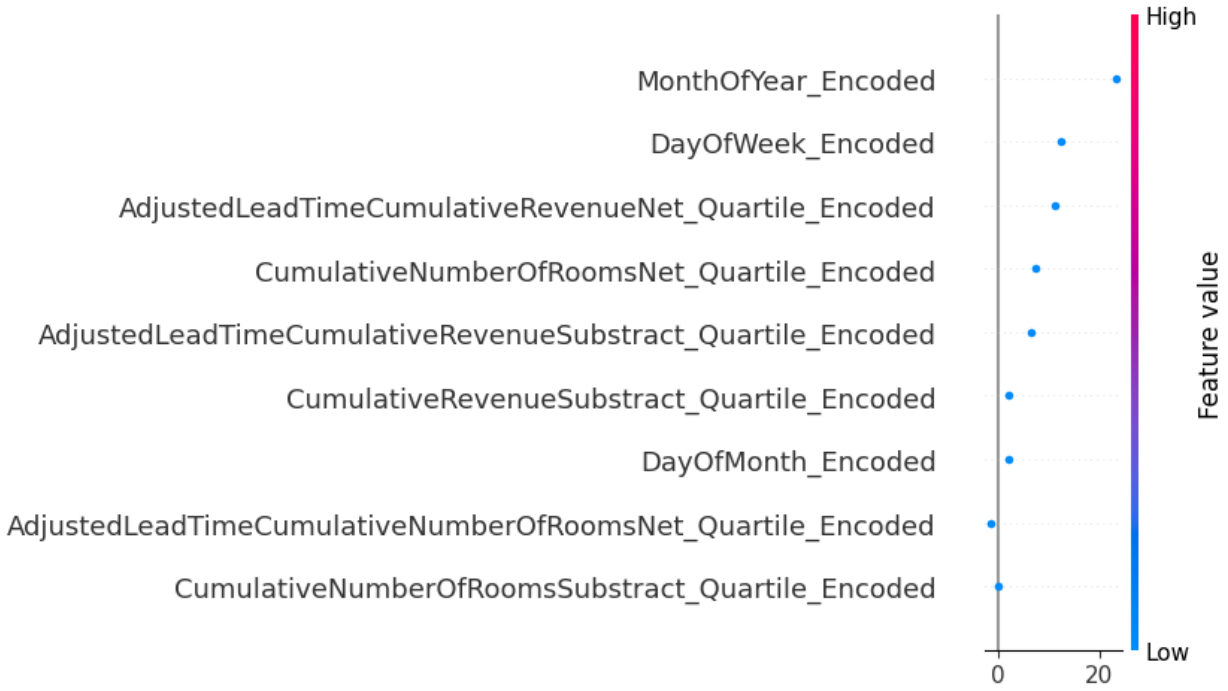


Figure 9.4.1.7 SHAP plot of Lightgbm regression model for the hotel total room booking dataset for the 4th row of external test data.

This plot is a simple and easy-to-understand explanation of the prediction for 4th row of data. The plot ranks the extent of impact each feature had on the specific prediction. The month of the year and day of the week has the most impact on predicting the 4th row of data. This indicates a strong trend and seasonality impact for this check-in date.

9.4.2 Hotel Booking Cancellation

We will look at permutation feature importance, and the accumulated local effects plot for the overall model explanation in this section. We tried creating a logistic regression surrogate model. However, its precision was found to be very low at 0.19 for the external test data. Hence, we will not try the surrogate model explanation for the hotel booking cancellation dataset.

Amongst the partial dependence plots and accumulated local effects plots, the latter is more robust as it considers the correlation among features. Hence, we will discuss the latter. As the number of features in the model is quite high, we will restrict our model explanation to the topmost features. Let us now look at figure 9.4.2.1 for the top 7 features based on the variation each feature has concerning the dependent variable.

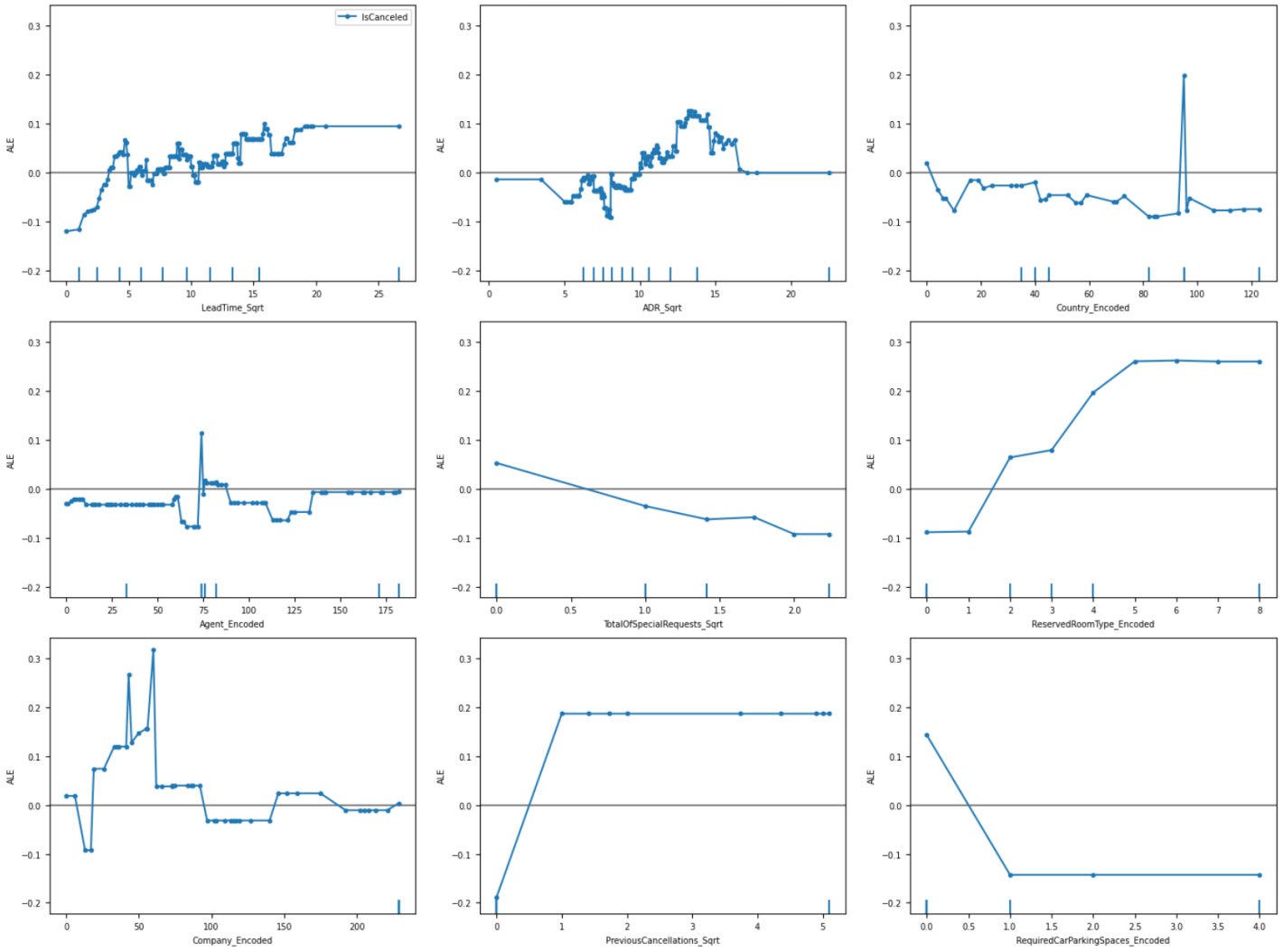


Figure 9.4.2.1 Accumulated local effects plot of Xgboost classification model for the hotel booking cancellation dataset

We can see from the plot that the lead time, followed by annual daily revenue (ADR) makes a huge impact on the dependent variable. This is confirmed by the huge variation in the plot, as well as distinctly visible scatter data points.

Let us now perform permutation feature importance for the top 40 features in figure 9.4.2.2.

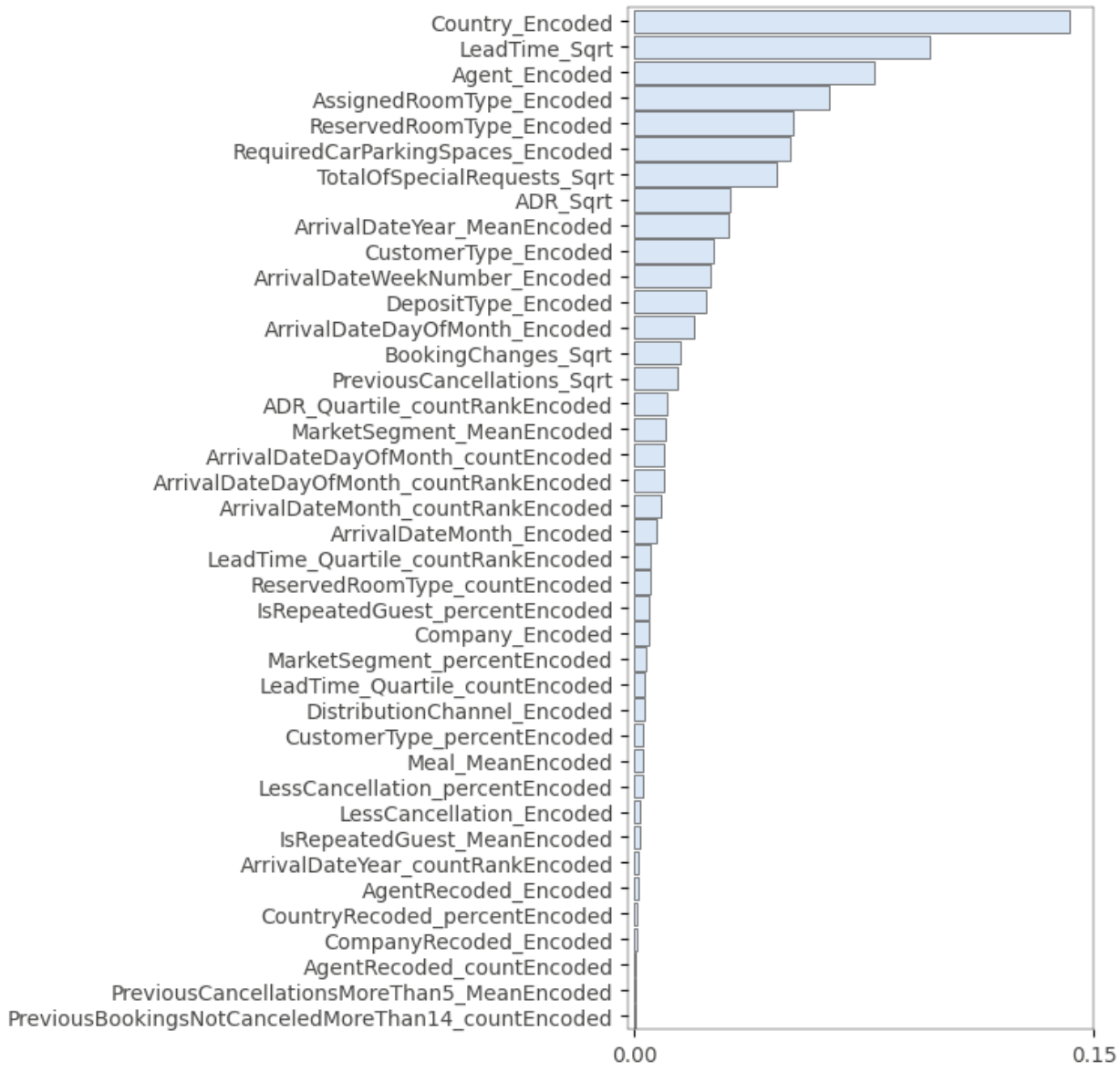


Figure 9.4.2.2 Permutation feature importance plot of Xgboost classification model for the hotel booking cancellation dataset

Permutation feature importance suggests that country is the biggest contributor to booking cancellation in the model. This is followed by lead time and agent. Guests from certain countries, as well as reservations from certain agents, are more likely to lead to cancelation in comparison to others. While reporting this, we also need to consider the ethical aspects of the model, so that it is not inherently biased and discriminatory towards different nationalities.

After understanding the model as a whole, now let us try to explore individual predictions made by the model. Let us start with ICE plots in figure 9.4.2.3 with 10 example observations from external test data.

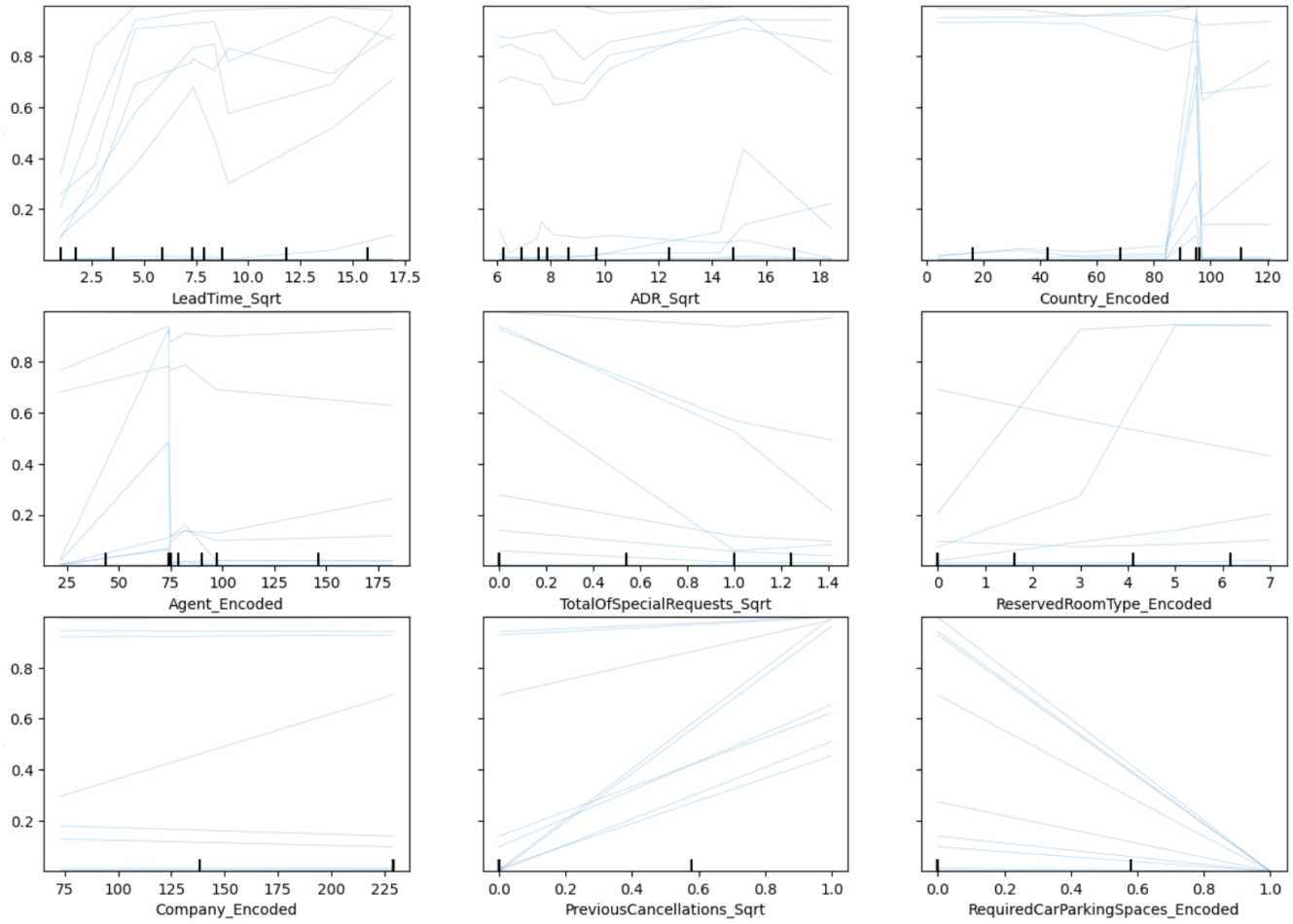


Figure 9.4.2.3 Individual Conditional Expectation plot of Xgboost classification model for the hotel booking cancellation dataset for the first 10 rows of external test data.

The ICE plot in figure 9.4.2.3 suggests that lead time and previous cancellations are clear indicators of the likelihood of cancellation for the hotel reservation. Although in some cases, it is difficult to differentiate, as seen for the lead time square root value between 7.5 and 10. However, in comparison to other features, these features give a clear indication of cancellation behavior.

Now let us look at the LIME plot for the Xgboost model in figure 9.4.2.3. As the number of features is numerous, we will be focusing on top features only.

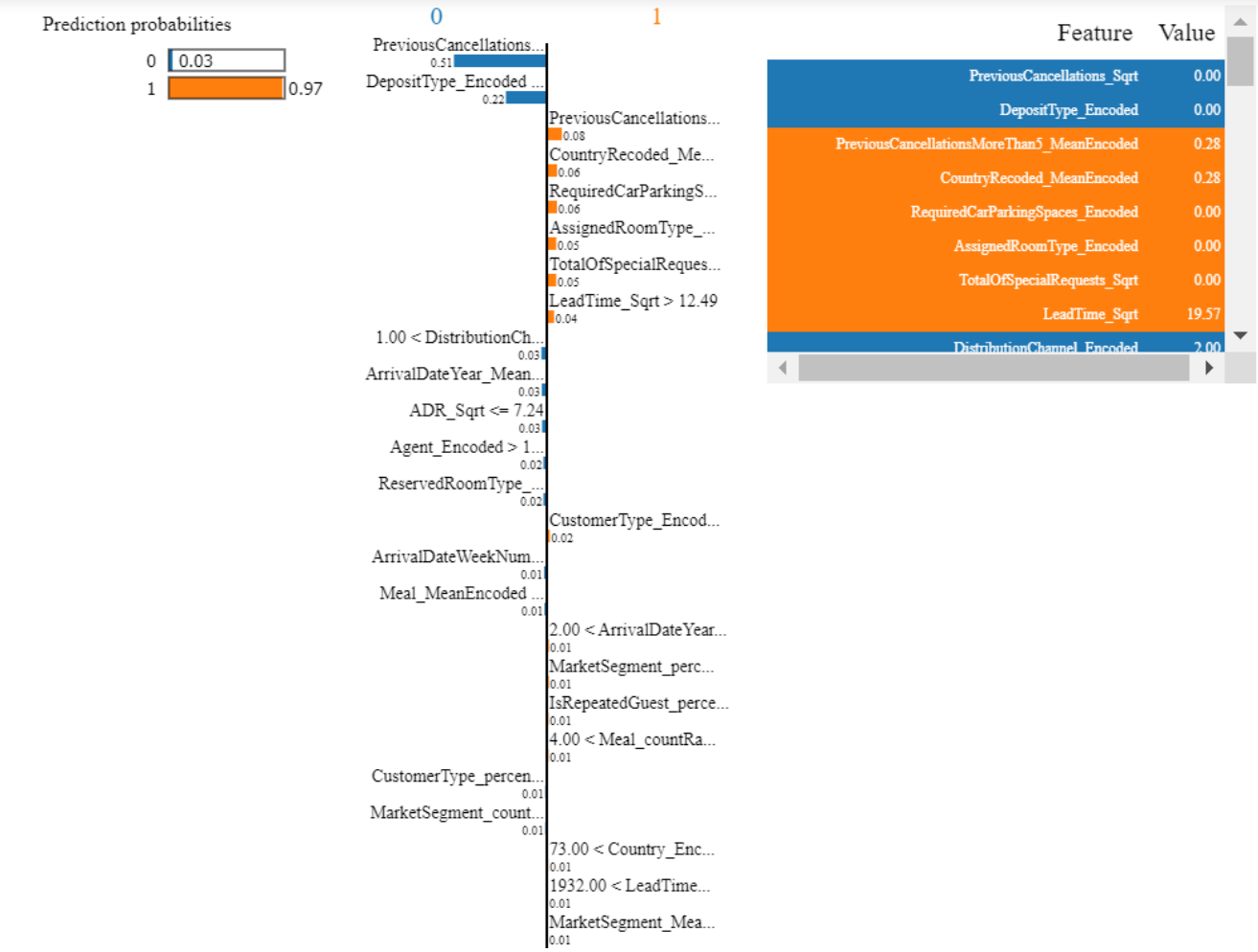


Figure 9.4.2.3 LIME plot of Xgboost classification model for the hotel booking cancellation dataset for the 4th row of external test data.

The prediction value is 0.97 as the probability of the reservation being canceled. The table on the right-hand side in figure 9.4.2.3 has the actual value of different features, based on which the Xgboost model predicted 0.97. The current value for the feature '*LeadTime_Sqrt*' is 19.57, which is higher than the threshold identified by LIME as 12.49 for the feature, beyond which the likelihood of cancellation increases.

We tried counterfactual explanations. However, it was not conclusive for the 4th observation in external test data. Hence, we will look at SHAP explanations for the 4th observation in the external test data for the top 20 features, as identified by the SHAP explanation.

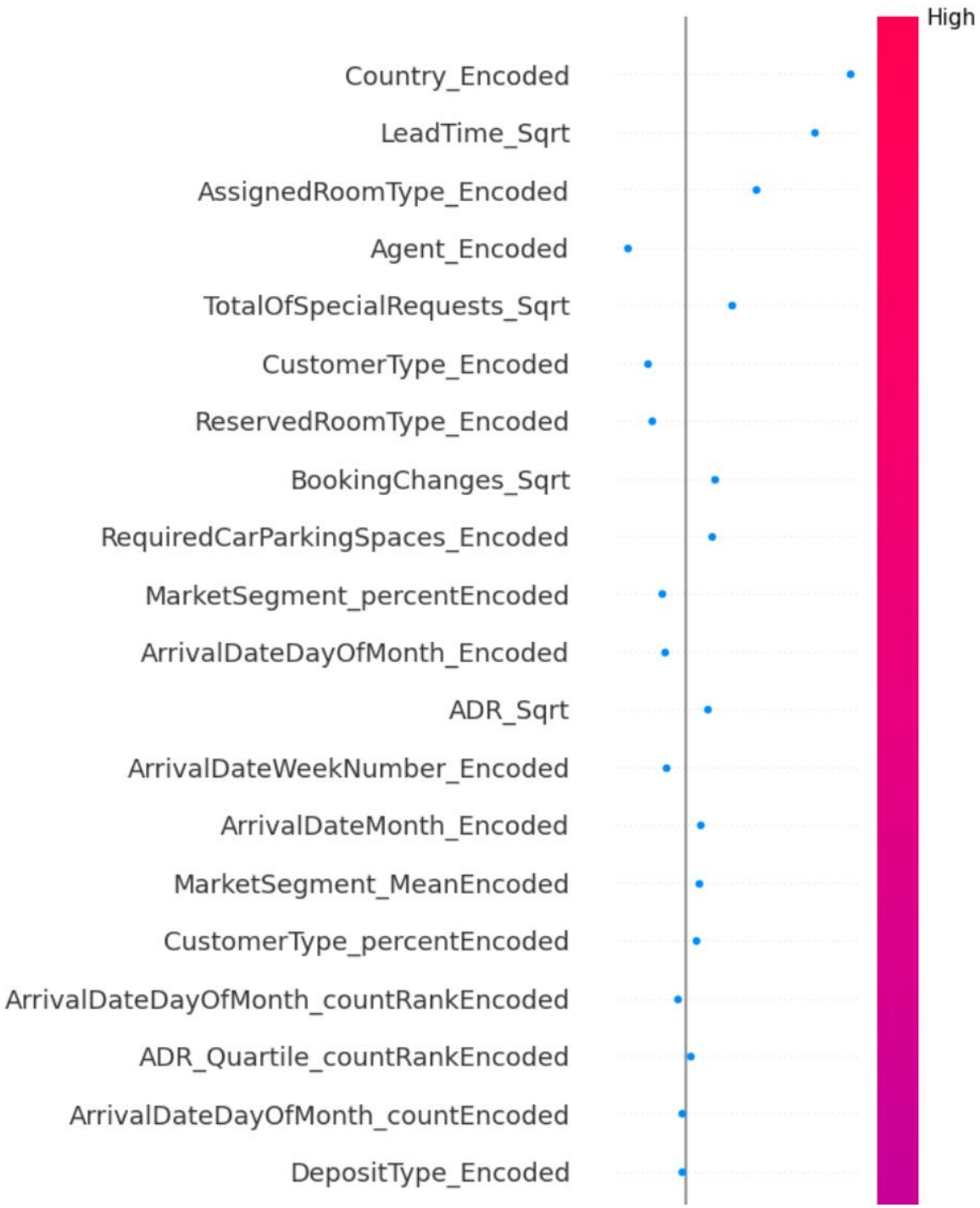


Figure 9.4.2.4 SHAP plot of Xgboost classification model for the hotel booking cancellation dataset for the 4th row of external test data.

The highest impact for the model prediction is made by encoded features of the country and the square root of lead time respectively. This matches with permutation feature importance in figure 9.4.2.2.

9.5 Conclusion

Machine learning models should be highly accurate and easy to explain. This is more applicable to models that affect the daily lives of human beings. For example, the potential use of a model for approving or rejecting loans. This might have moral, ethical, and legal consequences. Hence, it is essential to provide a suitable explanation for denying a loan.

Similarly, if a computer vision model is deployed for the home monitoring, we will expect it to do it without fail. If it is unable to identify intrusion, it might have legal consequences for the company who licensed the model. The company might be required to explain why it couldn't detect cases of intrusion. In some cases, the company might be liable to pay for the damages. In such a situation, the company might use the model explanation in their defense.

Model explainability is important in making the model predictions reliable. When the model is predicting correct values and is aided by proper explanations, it will instill confidence amongst intended users to use the model. Similarly, when the model is giving a wrong prediction, it will help fix accountability by identifying features and the values that contribute most to the prediction.

There are many methods for explaining the overall model. Similarly, there are many methods for explaining individual model predictions. We should use multiple methods for model explanation, before making final conclusions on the model. Similarly, for explaining a single instance of prediction, we should try different methods and compare against different methods. If more than one method suggests us the same conclusion, we should include this in our finding. If there are contradicting conclusions from different explanation methods, we should choose the conclusions which are closer to domain knowledge.

9.6 References

[1] Khare, A., Ceeba, P., Mishra, A. (2012). Influence of normative and informative values on fashion clothing involvement of Indian women. Journal of Customer Behaviour, Volume 11, Number 1, Spring 2012, pp. 9-32(24)

[2] Hoque, M., Buckus, S., Hoque, M., Alam, A., Hoque, M., Singh, N. (2021). Psychological Problems Experienced by Primary Healthcare Workers During COVID-19 Epidemic in South Africa. DOI:10.31487/j.PDR.2021.01.01

[3] Pei, D., Gong, Y., Kang, H. et al. Accurate and rapid screening model for potential diabetes mellitus. BMC Med Inform Decis Mak 19, 41 (2019). https://doi.org/10.1186/s12911-019-0790-3

Section V: Special Chapters

Chapter 10: Feature Engineering & Selection for Text Classification

10.1 Introduction

For text classification, the labels can be binary, multi-class, and multi-label. Each label is mapped to a text corpus. Each of these text corpuses will be used for constructing different types of features that will be further used by a feature extraction process to convert text documents to computer understandable format. To improve the efficiency of classifiers, reduce model complexity, and improve explainability, we can use feature selection before the feature extraction step. To reduce model complexity even further we can perform feature reduction, after feature extraction. Figure 10.1 shows the step-wise process that can be followed as a step before training machine learning classifiers.

Fig 10.1: a step-wise process of converting text corpus into machine learning model readable format.

Feature construction and feature extraction are the bare minimum needed for training machine learning or deep learning classifiers for text classification. Feature selection and feature reduction help us get superior performance on the created and extracted features.

10.2 Feature Construction

For text classification, the most common features used are 1) N-gram (Unigram, bigram, and trigram) 2) syntactic N-gram (SN-gram) 3) Domain-specific taxonomy features and 4) Meta features. This applies to both deep learning and traditional machine learning algorithms.

10.2.1 N-gram

In human language, words do not occur in isolation. Often, we need other supporting words to understand the whole situation. For example, if you want to invite your friend for coffee. You will not simply tell your friend 'Coffee'. Instead, you might say “Let's go for coffee”. Here we needed 3 words: “Let's go for”, to be able to explain our core objective of inviting friends for 'coffee'. Similarly, in some situations, we will need words after the main context word to understand the situation. This is not always true however, let's imagine your friend is sitting at your house and you prepared coffee in your kitchen. You proceed towards your friend with an extra cup of coffee and offer him after saying only one word 'coffee'. Most likely your friend will understand that you are offering him coffee and will accept it gracefully.

N-grams are a sequence of elements such as words, characters, part of speech, and dependency tags as they appear in the text. “n” in n-grams refers to the number of elements in the sequence.

We can represent a text document in multiple ways in n-gram for feature construction while considering whether to include previous and next words. The most common types are unigram, bi-gram, and tri-gram

Uni-gram is a way of representing textual words in isolation. bi-gram represents texts by considering one surrounding word, in addition to the current word. tr-gram considers 2 previous words, in addition to the current word. We can similarly go for quad-gram and so on. However, the number of unique n-grams increases manifold and becomes difficult to represent as machine learning features embedding and training a classifier. Beyond uni-gram, in n-gram, words are concatenated with each other using a special character such as the underscore “_”.

Example: For a sentence that reads "You are so adorable!". Below is an example of n-grams.

uni-gram: ["you", "are", "so", "adorable",”!”]

bi-gram: ["you_are", "are_so", "so_adorable",”adorable_!”]

tri-gram: ["you_are_so", "are_so_adorable", "so_adorable_!"]

10.2.2 Syntactic N-gram

If n-grams are extracted by the order in which the elements are present in syntactic dependency trees that follow each other in the path of the syntactic tree, and not in the text. We call such n-grams syntactic n-grams (sn-grams) [1].

Human spoken and written languages typically follow a hierarchy structure, such as sentences, clauses, phrases, and words. The dependency tree is a visual representation of the linguistic structure, in which the grammatical hierarchy is graphically displayed. Connecting points in the tree diagram are called nodes, with one word being the head and the other being the dependent of the relation. Syntactic dependencies are obtained through dependency parsing.

Consider the example sentence which can be a sarcastic remark to an obese person “You should not run anymore”. Figure 10.2.2 shows how the dependency parse tree will look like, along with.

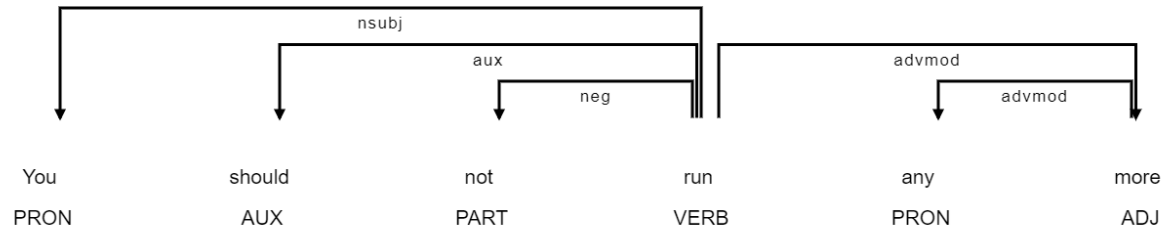


Fig 10.2.2: syntactic dependency tree for example.

In this example, “run” is the root word. Below are the dependency relationships in this sentence.

(run, You), connected by nominal subject relationship.

(run, should), connected by an auxiliary.

(run, not), connected by a negation modifier.

(run, more), connected by an adverb modifier.

(more, any), connected by an adverb modifier.

We follow the arrow-marked path in the dependencies to obtain syntactic n-grams. In this example, all the relationships are between 2 words only, except for “run” -> “more” -> “any”, where the path exists between 3 words. This can be a candidate for syntactic tri-gram.

We used the companion python library SNgramExtractor for this book and extracted the below pairs of syntactic bi-grams and tri-grams from the example sentence.

Syntactic bi-grams: "run_You", "run_should", "run_not", "more_any", "run_more"

Syntactic tri-gram: "run_more_any"

SngramExtractor uses Spacy language model for extracting sn-grams. We can also specify which language model we want to use. This allows us to use non-English language models and extract syntactic n-grams of non-English languages as well. SngramExtractor follows object-oriented pattern. It can process one sentence at a time. For an example sentence 'You should not run any more', below syntax will help us extract bi-gram and tri-gram.

```
text = 'You should not run any more'
SNgram_obj=SNgramExtractor(text,
                           meta_tag='original',
                           trigram_flag='yes',
                           nlp_model=None)
output=SNgram_obj.get_SNgram()
print('Original Text:',text)
print('SNGram bigram:',output['SNBigram'])
print('SNGram trigram:',output['SNTrigram'])
```

In this syntax, we have not specified nlp_model parameter. This has been set as ‘None’. In such a case, we use en_core_web_sm, as the default language model. Below is how the output will look like for the executed syntax.

```
Original Text: You should not run any more
SNGram bigram: run_You run_should run_not more_any run_more
SNGram trigram: run_more_any
```

Unlike traditional n-grams, syntactic n-grams are less arbitrary. Dependency parsing is language-specific and different for each language. Hence by extracting syntactic n-grams, we can include linguistic-rich features in our model. Additionally, as we are not including all the possible n-grams that can be created, their numbers are less than the number of traditional n-grams. Hence, syntactic n-gram features can be

interpreted as a linguistic phenomenon, while traditional n-grams have no plausible linguistic interpretation and they are merely a statistical artifact.

One shortcoming with syntactic n-gram is that it is dependent on the availability of syntactic parser and lexical resources. Not all human language has lexical resources to be able to build SN-gram. For the languages for which we have lexical resources available, it requires the construction of dependency parse trees, which increases processing time. In contrast, traditional n-grams are faster to compute.

10.2.3 Domain-Specific Taxonomy Features

The use of background knowledge is largely unexploited in text classification tasks. To include background knowledge in the form of metadata, ontology or taxonomy-generated features can be incorporated in machine learning classifiers. It acts as a second-level context. It can lead to improved interpretability and performance of the model. For doing so, we focus on semantic structures, derived through hypernym relation between words.

Individual words are mapped to higher-order semantic concepts. For example, for the word “tiger”, its WordNet hypernym is the term “mammal”. It can be further mapped with “animals”. We can ultimately reach the most general term, which is an entity.

Before we construct hypernym-based features, word-sense disambiguation should be performed, get the exact context behind the word. For example, the word bank can have different meanings, in different contexts. Between the sentences "I went to the bank to withdraw money." and "I was relaxing near the bank of the river in the morning.". In both sentences, the meaning of bank is different. In the first sentence, the bank refers to a financial institution and in the second sentence, a bank refers to a recreational place.

For each word in the document, we find a hypernym path between the word and the most general term. Document-based taxonomy thus created, is a tree-like structure. Word-specific tree structures are collated at the document level to create a corpus-based taxonomy at the document level.

One shortcoming of using the Wordnet-based taxonomy feature is that Wordnet is a general-purpose taxonomy. If a customized lexical resource can be created for the specific domain for which the classifier is being developed, it can sharply increase the machine learning classifier and its usability even further.

10.2.4 Meta Features

These features have been used previously in predicting whether a question posted on StackOverflow can be deleted. Stackoverflow is an online forum for software engineers and programmers to post their questions. This website is moderated by volunteers who delete questions that are too broad or off-topic. To reduce manual moderation and automate this task, several algorithms and approaches have been suggested. One such method [2] is to use meta-features from past user-generated content and site-generated statistics. Meta features from the users were grouped into 4 categories: profile, community, content, and syntactic.

Profile features are defined based on the historical statistics of the user who posted the question. For example, how old is the account which posted the question, the number of previous questions, the number of previous answers, the question to the age of account ratio, answer to the age of account ratio. This can be used for example in Twitter, how many tweets have been posted, how many tweets replied to, how many tweets are liked, and how many people follow the Twitter account. Similarly, many such statistics can be computed for the users as scores based on functionalities available on the social media website.

Community features in StackOverflow are average reputation and popularity statistics for the user. For example, the average answer score, the average number of comments, and the average question score. On Twitter, for example, we can calculate the average number of retweets for the tweets of users, the average number of likes, and replies received in the past for their tweets.

Many of the content and syntactic meta-features use LIWC scores. LIWC stands for Linguistic Inquiry and Word Count. LIWC2007 is a software that contains 4,500 words and 64 hierarchical dictionary categories. Given an input natural language text document, it outputs a score for the input against all 64 categories after analyzing writing style and psychometric properties.

Content features were computed based on the textual content of the question. For example, the average number of URLs, number of previous tags, and LIWC score of Pronouns. On Twitter, we can similarly calculate the average number of hashtags used by users in past, and the number of links posted.

Syntactic features are computed based on the writing style of the written text. For example, the number of characters, upper case characters, lower case characters, and digits in the text document. These can be used as it is.

10.3 Feature Selection

Feature selection in text classification is applicable for all features, except meta-features. In the case of text classification, feature selection can be done through 1) the Filter method 2) the embedded method using document frequency with a bag of words or TF-IDF 3) the genetic algorithm 4) Ensemble feature selection.

The biggest problem with text features is that the number of unique features is in the thousands. If you move from unigram to bi-gram, the number of unique bi-gram features increases manifold. Not every token is useful and many are useless noise. We need to be highly discriminatory to select a handful list of such unique features.

10.3.1 Filter Method

In the case of text classification, feature selection using the filter method is performed first and then passed into for feature extraction before being fed into the model. They try to find out the least useful features. When done correctly, these methods can help in reducing computation time and prevent overfitting.

10.3.1.1 Document Frequency

It is the number of times a word or n-gram is present across all documents [3], discarding repetitions. It is different from word occurrence count. Word count is how many times the specific word or n-gram appeared across all documents. It does not discard repetitions.

For example, if we are analyzing 100 documents and the word 'Lion' is present 10 times in only 1 document and not present in any other document and there is another word 'Bear', which is present in 7 documents and out of those 7 documents, it is present 4 times in a single document. Word frequency for both 'Lion' and 'Bear' is 10. However, document frequency is 1 for 'Lion' and 7 for 'Bear'.

If we have to choose between word and document frequency, we should remove features based on document frequency as it has more impact on feature vector representation. A word can have a very high word frequency because it is mentioned in a document many times, but a word that is mentioned in multiple documents but comparatively less frequently within each document can have more bearing on

feature representation. The same applies to high-frequency words, as these have a low degree of bearing in the feature vector.

Words that have either very low or very high document frequency have little bearing on feature representation. We can remove these words. The same principle can be applied to higher-order n-grams, syntactic n-grams, and taxonomy features.

10.3.1.2 Chi-Square

It is otherwise known as X^2 statistic [3] and it measures the lack of independence between term(t) and class(c). It has a natural value of zero if t and c are independent. If it is higher, then the term is dependent. It is not reliable for low-frequency terms

Using the two-way contingency table of a term t and a category c, where A is the number of times t and c co-occur, B is the number of times the t occurs without c, C is the number of times c occurs without t, D is the number of times neither c nor t occurs, and N is the total number of documents, below will be the formula to calculate Chi-square statistic.

$$\chi^2(t,c) = \frac{N \times (AD - CB)^2}{(A+C) \times (B+D) \times (A+B) \times (C+D)}.$$

10.3.1.3 Mutual Information

In this method, rare terms will have a higher score than common terms [3]. For multi-class categories, we will calculate the MI value for all categories and will take the Max (MI) value across all categories at the word level. It is calculated using the below formula

$$I(t,c) \approx \log \frac{A \times N}{(A+C) \times (A+B)}$$

10.3.1.4 Proportional Difference

It calculates how close two numbers are to becoming equal [5][6]. It helps find unigrams that occur mostly in one class of documents or the other. It helps find unigrams that occur mostly in one class of documents or the other. The values of PD are restricted between −1 and 1. Values near –1 indicate that

a word occurs in approximately an equal number of documents in all categories and a 1 indicates that a word occurs in the documents of only one category.

Below is the formula for calculating the proportional difference

$$\mathrm{CPD}(w,c) \quad = \quad \frac{A-B}{A+B}.$$

For multi-class categories, we will calculate the MI value for all categories and will take the Max (MI) value across all categories at the word level

$$\mathrm{CPD}(w) = \max_i \{\mathrm{CPD}(w, c_i)\}.$$

10.3.1.5 Information Gain

It measures the number of bits of information obtained for category prediction by knowing the presence or absence of a term in a document [3][4]. It is calculated using the below formula

$$IG(t_j, c_k) \quad \approx \quad -\frac{A+C}{N}\log(\frac{A+C}{N}) + \frac{A}{N}\log(\frac{A}{A+B}) + \frac{C}{N}\log(\frac{C}{C+D})$$

10.3.2 Metaheuristics Algorithms

The genetic algorithm technique has been found to perform well for the support vector machine technique with conceptual representation wordnet features and TF-IDF feature representation [7]. Instead of randomly selecting several text tokens as features, we can instead use a percentage of text tokens as features from the total number of words [8].

The Genetic algorithm can be used for deciding which text token should be included or excluded from the feature vector. Although metaheuristic methods do not perform well when the number of features is very high, like in the case of text classification, it has still been cited in numerous research papers.

There are similar mentions of other metaheuristics feature selection algorithms for text classification. However, these methods all have one limitation in common, i.e., they cannot perform well on high dimensional data where the number of features is numerous.

10.3.3 Ensemble Feature Selection

There are linear and non-linear models which can be trained individually to identify the target variable. Each technique has its strengths and weaknesses. One way to leverage the strength of different models is to perform ensemble learning by training all the base learner models. We can also leverage the strength of the different types of features such as n-grams and feature extraction methods, such as TF-IDF or bag of words vector.

On one hand, it can significantly improve classification model results, on the other hand, it can make the entire process computationally heavy for actual use. We might need heavy computing power for predicting labels from base models on new data. To solve this, we can 1) reduce the complexity of individual base models and 2) reduce the number of base models. This can be done by performing feature selection at both base models and ensemble models. This entire process can be jointly done through an algorithm called ensemble feature selection [9]. Figure 10.3.3 details the original ensemble feature selection algorithm

Algorithm 1 Complete EFS-BGA Algorithm

Input: training set S, validation set S_v, the number of base learners T, the number of features k;
Output: a subset of features;
1: **for** $i \leftarrow 1$ to T **do**
2: $S_i \leftarrow$ bootstrap resample from S;
3: $FS_i \leftarrow$ the feature subset produced by LASSO;
4: $F_i(x) \leftarrow$ the fitting function learned by LASSO;
5: **end for**
6: $W \leftarrow$ randomly produce a weight vector, the weight value is among 0 and 1;
7: evolve the weight vector **W** through GA module; //the fitness function is as Eq. (12)
8: $W_{opt} \leftarrow$ the evolved best weight vector in GA, the weight value is among 0 and 1;
9: $FS_{vote} \leftarrow \sum_{i=1}^{T} w_i \times FS_i$; //$\mathbf{W} = [w_1, w_2, ..., w_T]^T$
10: $FS_{opt} \leftarrow$ the final optimized feature subset with the most votes in FS_{vote};
11: **return** FS_{opt};

Fig 10.3.3: Ensemble feature selection (Wang et. al. 2020)

This algorithm was created for structured data. For text classification, we can make adaptations at different levels. For the base model, we can use document frequency to reduce words from the lower and higher count of occurrence, instead of the lasso feature selection used in the original paper. Through grid search, we can find the optimal combination of lower and higher document frequency. For lower frequency, we use document count and for the upper frequency, we use occurrence in percent of documents. By reducing the number of words, feature vector size decreases and model complexity also reduces.

The original paper performs bootstrapping. In the adaption, we are performing 5 or 10 cross-validations to avoid overfitting. The entire labeled dataset can be divided into 3 parts 1) training data for training individual models, 2) ensemble training data for training ensemble model and 3) test data. This can be done in a 60:20:20 ratio. For each cross-validation set, training data is used for training base learner models such as logistic regression, random forest, etc. After training is complete, the base model is used for predicting raw class probabilities for ensemble training data and test data. An ensemble learning classifier is trained using raw class probabilities for ensemble training data as features and actual labels as

the outcome variable. Finally, the ensemble learning model is used for predicting outcomes for test data. A performance metric, such as the f1 score is used for each cross-validation and is averaged across all cross-validations to ascertain model performance.

For the ensemble model, we perform feature selection through the genetic algorithm. This step is done as it is in the original algorithm. Features that were discarded by the genetic algorithm are mapped with all the base models. For any base model, if no class probability feature is present in the list of features selected by the genetic algorithm, we can discard those models. This entire process will result in fewer base models, with each base model being less complex.

Python implementation of these filter methods, metaheuristics, and ensemble feature selection exists in the companion python library TextFeatureSelection.

10.4 Feature Extraction

In the case of text classification, feature extraction is done by representing text documents into matrices, which can then be used by machine learning algorithms as features to classify labels. For traditional machine learning models, feature extraction is done mostly by either a bag of words matrix or the TF-IDF matrix.

10.4.1 Bag of Words

It is a scoring method in which the count of individual words is used for presence and 0 for absence. Let us consider example 10.4.1 below with 4 corpora.

Example 10.4.1

Corpus 1: I am very happy.

Corpus 2: I just had an awesome weekend.

Corpus 3: I just had lunch. You had lunch?

Corpus 4: Do you want chips?

Unique words from these 4 corpora, after removing punctuation marks are: 'am', 'an', 'awesome', 'chips', 'Do', 'had', 'happy', 'I', 'just', 'lunch', 'very', 'want', 'weekend', 'you'

Bag of words vector will appear in table 10.4.1.

	Corpus 1	Corpus 2	Corpus 3	Corpus 4
am	1	0	0	0
an	0	1	0	0
awesome	0	1	0	0
chips	0	0	0	1
Do	0	0	0	1
had	0	1	2	0
happy	1	0	0	0
I	1	1	1	0
just	0	1	1	0
lunch	0	0	2	0
very	1	0	0	0
want	0	0	0	1
weekend	0	1	0	0
you	0	0	1	1

Table 10.4.1: Bag of words vector

In the above table, the corpus is present in the column and the words are present in the row. If a specific word is present in the corpus, its count is presented in corresponding cells against the word. If a word is not present in the corpus, 0 is presented in the corresponding cell.

For example, the word "*lunch*" is present in corpus 3. Hence count 2 is present in the corresponding cell. The word "*am*" is only present in corpus 1, for once. Hence the corresponding cells have the value 1. Again, the word "*am*", is not present in any other corpus apart from corpus 1. Hence, in corpus 2, 3, and 4, the value in the corresponding cells is 0.

10.4.2 Term Frequency Inverse Document Frequency

This is otherwise abbreviated as TF-IDF. Term frequency and inverse document frequency of each word are calculated and multiplied to obtain the TF-IDF score.

Term frequency (TF) is calculated by dividing how frequently a term appears in a corpus, by the total number of terms in the corpus.

TF = Count of the number of times the term ‘t’ is present in the corpus) / Count all the terms in the corpus.

Inverse document frequency is calculated by taking the logarithm of the total number of the corpus by the count of documents that has the term.

IDF = log (Total number of documents / Count(documents which have term ‘t’))

TF-IDF vector for the corpora in example 10.4.1 will appear as per table 10.4.2

	Corpus 1	Corpus 2	Corpus 3	Corpus 4
am	0.541736	0	0	0
an	0	0.463709	0	0
awesome	0	0.463709	0	0
chips	0	0	0	0.525473
do	0	0	0	0.525473
had	0	0.365594	0.552779	0
happy	0.541736	0	0	0
i	0.345783	0.29598	0.223761	0
just	0	0.365594	0.27639	0
lunch	0	0	0.70113	0
very	0.541736	0	0	0
want	0	0	0	0.525473
weekend	0	0.463709	0	0
you	0	0	0.27639	0.414289

Table 10.4.2: TF-IDF vector

10.4.3 Word2vec

Word2Vec is a shallow neural network that tries to understand the context of words [10]. In word2vec, individual words are represented as one-hot vectors and are used for creating a vector space. It has a hidden layer, which is a fully-connected dense layer. The weights of the hidden layer are the word embeddings. The output layer outputs probabilities based on the Softmax activation function for the target words from the vocabulary. Such a network is a "standard" multinomial (multi-class) classifier.

The objective function of word2vec is to maximize the log of similarity between the vectors for words that appear close together in the context and minimize the similarity of words that do not. This is called the Softmax function.

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{j\in nb(t)}\log p(w_j|w_t)$$

where $nb(t)$ is the set of neighboring words of word w_t and $p(w_j|w_t)$ is the hierarchical softmax of the associated word

Since classes are actual words, the number of neurons is huge. The Softmax function when applied to such a huge output layer will be computationally expensive. To save costly computation of the Softmax in the output layer, noise-contrastive estimation is used. This converts the multinomial classification problem to a binary classification problem.

$$\arg\max_{\theta}\sum_{(w,c)\in D}\log p(c|w)=\sum_{(w,c)\in D}\left[\log\exp v_c\cdot v_w-\log\sum_{c'}\exp v_{c'}\cdot v_w\right]$$

C is the context word, and *W* is the focus word. V_c is the embedding of context words and V_w is the embedding of focus words.

Given a pair of words, we will predict the context target or not. For this, we will create positive and negative samples. For example, if 'Orange' and 'juice' are positive, it will be inferred that these appear together in the same context. Similarly, if 'Orange', and 'king' are negative, it will be inferred that both words do not appear in the same context. Positive samples are extracted from the context window whereas negative sample is drawn randomly from the dictionary of all words. Depending on the size of the data, if the data is small, negative sample size k is selected between 5<>20 and for large, k is between 2<>5. Accordingly, word pairs are created consisting of the input word, surrounding context word, and target label whether it is a positive sample or negative sample.

For backpropagation, two matrices of the same size are created namely embedding and context. The number of rows in the matrix is the size of the vocabulary in the corpus and the number of columns is defined as the size of the embedding. The embedding matrix has input word representation and the context matrix has representation from context word. At the start of the training process, random values are initialized in these matrices.

Through the dot product of input embedding with each of the context embeddings, we obtain similarities between the input and context embeddings. Resultant dot product numbers are converted into positive numbers ranging between 0 and 1 using the sigmoid function. The prediction error is obtained for each pair of input words and context words by subtracting sigmoid transformation from the actual label value of the positive sample(1) and negative sample(0). The error so obtained is inserted in the embedding layer for each of the words in input for all the combination pairs. This process is repeated through the dataset, based on the number of epochs defined. Finally, the context matrix is discarded and the embedding matrix is used.

10.5 Feature Reduction

There are a few methods available for the feature reduction of sparse vectors. The most notable ones are singular value decomposition (SVD) and non-negative matrix factorization (NMF). The objective of both techniques is to identify latent spaces that explain the original data in a more concise and computationally efficient manner.

10.5.1 Singular Value Decomposition

Bag of words and TF-IDF methods create high-dimensional vectors which contain useful numbers. However, these are very long and sparse.

Storing and processing high-dimensional data is computationally intensive and expensive. Hence, ideally, we will like to reduce the bag of words and TF-IDF vectors into short dense vectors which contain the most important dimensions of variation. A traditional method of doing this is called singular value decomposition. SVD can reduce the TF-IDF matrix into a few columnar features, which makes machine learning classifier training faster. It removes less important parts of the matrix and produced an approximation in the specified number of dimensions. It is a classical second-order eigenvector technique, in which components are generated from the original vector, which is uncorrelated. SVD is used as a data compression technique.

10.5.2 Non-Negative Matrix Factorization

This method is otherwise known as NMF. This is suitable when features are made up of non-negative elements. In the case of text classification, both bag of words and TF-IDF feature matrix, values are positive. The lowest value in the matrix is 0. Hence, we can use it for processing sparse matrices and

finding meaningful components. It is mostly used for topic modeling techniques. Through NMF and topic modeling, we can identify the most common word in social media conversations. This can help us create a lexicon of useful words for the domain. The availability of a comprehensive lexicon can aid in identifying the different classes for the classification model, as well as expanding the understanding of the broad domain for which the classification model is being developed.

10.6 Conclusion

For creating a machine learning model, no guaranteed method works. This is especially true for text classification models. Using a variety of feature engineering techniques can improve the odds of being able to create models with high predictive performance. In the case of text classification, features are in thousands. More the features, the more the model complexity. Complex models lead to higher computational costs to train and use such models. Feature selection can help us reduce the number of features and model complexity. We can also use feature reduction to reduce the size of the feature matrix, without losing information from features. This can reduce model complexity even further.

Many times, we find different modeling techniques perform differently on the same test data. To harness the power of different models, we can ensemble. The drawback of ensembling is that it will lead to a huge number of base models. This results in higher computational costs for training and using the model.

Ensemble feature selection can help solve all of the above problems by doing feature selection for each base model to identify fewer complex models for different combinations of features, feature representation, and models. Finally, it identifies the best set of models which give the highest predictive performance. Thereby reducing the number of base models.

10.7 References

[1] Sidorov, G., Gupta, A., Tozer, M., Català, D., Catena, A. and Fuentes, S. (2013). Rule-based System for Automatic Grammar Correction Using Syntactic N-grams for English Language Learning (L2). CoNLL Shared Task, pp.

[2] Correa, D., Sureka, A. (2014). Chaff from the wheat: characterization and modeling of deleted questions on stack overflow. Proceedings of the 23rd international conference on World wide web April 2014 Pages 631–642https://doi.org/10.1145/2566486.2568036

[3] Yang, Y., Pedersen, J. (1997). A Comparative Study on Feature Selection in Text Categorization. ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning. July 1997. Pages 412–420

[4] Largeron, C., Moulin C., Géry M. Entropy based feature selection for text categorization. ACM Symposium on Applied Computing, Mar 2011, TaiChung, Taiwan. pp.924-928, ff10.1145/1982185.1982389ff. ffhal-00617969

[5] Simeon, M. and Hilderman, R. (2008). Categorical Proportional Difference: A Feature Selection Method for Text Categorization. In Proc. Seventh Australasian Data Mining Conference (AusDM 2008), Glenelg, South Australia. CRPIT, 87. Roddick, J. F., Li, J., Christen, P. and Kennedy, P. J., Eds. ACS. 201-208.

[6] O'Keefe, T., Koprinska, I. (2009) Feature Selection and Weighting Methods in Sentiment Analysis

[7] Bidi, N., Elberrichi, Z. (2016) Feature selection for text classification using genetic algorithms. 8th International Conference on modeling, Identification and Control (ICMIC). 15-17 Nov. 2016

[8] Kumar, G., Ramachandra, G., Nagamani, K. (2014) An Efficient Feature Selection System to Integrating SVM with Genetic Algorithm for Large Medical Datasets. International Journal of Advanced Research in Computer Science and Software Engineering. Volume 4, Issue 2, February 2014

[9] Wang, H., He, C., Li, Z. (2020). A new ensemble feature selection approach based on genetic algorithm. Soft Comput 24, 15811–15820 (2020). https://doi.org/10.1007/s00500-020-04911-x

[10] Goldberg, Y., Levy, O. (2014) word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method

Chapter 11: Things That Can Give Additional Improvement

11.1 Introduction

There are factors beyond feature engineering and feature selection that can help in improving the predictive power of the model. These are data cleaning, domain-specific feature engineering, ensemble learning, and hyperparameter optimization. We will briefly discuss hyperparameter optimization and ensemble learning in this chapter.

11.2 Hyperparameter Tuning

Many machine learning algorithms have hyperparameters that help the algorithm learn effectively from the data. Increasing or decreasing the value of these hyperparameters can alter the performance of the model for better or worse. For example, random forest and many other tree algorithms have a parameter for defining the number of trees that will be used for the model. By increasing or decreasing the number of trees, model performance might change.

If the number of trees is low, say 1, and we increase it to 5, the performance of the model might improve. If we further increase from 5 to 20, the model might perform even better. However, if we keep increasing the number of trees, it might not always lead to a better-performing model. After a certain number of trees, the model performance may not increase, or worse, performance might decrease. Hence it is very important to identify the number of trees in this case.

Some machine learning techniques have multiple hyperparameters. Not all hyperparameters are equally important for improving model performance. Depending on the computing power and time available, we should try hyperparameter tuning. There are many techniques available for hyperparameter tuning.

The most basic method of hyperparameter optimization is called manual search. In this method, we try all possible combinations for feature selection. A for loop can help us perform all possible combinations to fit the model with different values of hyperparameter and identify the combination at which the model gives the best performance. There is another method for hyperparameter search known as grid-search. This method is an improvised version of manual search hyperparameter optimization. The only advantage this method has is that it saves us from writing multiple nested loops.

Both manual and grid search methods are computationally expensive. Randomized Search method on the other hand trains the model on random hyperparameter combinations. It saves us from trying all combinations of hyperparameter values. One disadvantage of this method is that it may not identify the optimal values of hyperparameters.

Bayes Grid Search uses Bayesian optimization to identify optimal values of hyperparameters. It can be done with the help of the python library skopt. There is another python library optuna, that uses the define-by-run principle to enable the user dynamically construct the search space. It allows the user ways that have never been possible with previous hyperparameter tuning frameworks. It does so by combining efficient searching and pruning algorithms, and thereby greatly improves the cost-effectiveness of optimization.

11.3 Ensemble Learning

For the case of room booking prediction for hotels, let's imagine that linear regression predicts with a high degree of accuracy for December month and random forest for the first 6 months of the year. Xgboost on the other hand outperforms for the remaining months of the year.

If we can leverage the individual strength of different models, we can come up with a highly accurate model for predicting hotel booking. Ensemble learning helps us perform just that by leveraging the strength of different models to provide a highly reliable prediction.

The simplest form of ensembling is called averaging, wherein we take the prediction from multiple models and take a mathematical average of prediction from all the models to arrive at the final prediction. If some modeling technique performs better than others, we can give weights to each model, which signifies the degree to which the model is accurate. This will allow us to perform weighted average ensembling. Ensembling by conditional averaging is another approach in which we take a simple average from 2 models based on some conditions laid upon output values. For example, if there are 2 models and model 1 predicts accurately values under 50 and model 2 predicts values above 50. We can simply add an if-else condition for prediction on model 1.

Bagging is a useful form of ensembling for high-variance techniques. The high variance techniques are prone to overfitting, such as decision trees. We can create multiple models of the same techniques, simply by changing random seed, parameter tuning, changing the number of features and records in the dataset, etc. This can be impactful if we have a larger number of models. We can also use BaggingClassifier and

BaggingRegressor in Sklearn to do the same task. In contrast, for boosting-based ensemble learning, newer models are added sequentially depending on how well the previous model performed. We can also give weight for performing boosting.

Another form of ensemble learning is called stacking. For stacking, training data is divided into two parts. One part is used for training models using a different diverse set of techniques. These individual models then predict values for the second part of the dataset. For the second part of the dataset, predictions obtained from individual models are used as features. By using the dependent variable in the second part of the training dataset, a final ensemble model is trained. For stacking to be successful, we should try a different diverse set of techniques, such as linear, non-linear, etc.

11.4 Signal Processing

It is the domain that deals with analyzing, modifying, and synthesizing signals. A signal can be audio, video, radar measurement, etc. It converts and transforms data to enable us to see things that are not possible via direct observation. The most common applications of signal processing are audio and video compression, speech recognition, improving audio quality in phone calls, oil exploration, etc.

Signal processing can help us extract important parts of the signal, which can then be used as features to train the model. If the features are derived from signals, it can help if we clean the features using signal processing techniques methods. We will discuss two such methods which are useful for machine learning applications. Filtering of signals, and baseline removal for Raman spectra.

11.4.1 Filtering

In signal processing, filtering denotes removing unwanted frequencies and frequency bands from the signal. It helps in increasing the precision of the data without distorting the signal. It is performed through a process known as convolution. It fits subsets of adjacent data with low-degree polynomials using linear least squares. It has wide use in radio, music synthesis, image processing, etc. Savitzky-Golay filter is one of the commonly used methods for removing noise from data. Let’s discuss some practical machine-learning applications that use filtering.

Savitzky-Golay filter has been used [1] in demand forecasting for eliminating outliers and noises in the non-stationary time series. This helps time series models to learn better from the filtered data and forecast more accurately. It can also be used with deep learning. For example, it has been used [2] with one-

dimensional CNN layers to identify abnormal EEG signals, without using any explicit feature extraction technique.

We will analyze the beaver body temperature discussed in chapter 1. There are 2 beavers, and we will only analyze the body temperature of beaver 1 for the sake of simplicity. Let's look at beaver1's body temperature with and without filtering in figure 11.4.1. We can see that filtered temperatures have less volatility and are more stable. It retains the information about patterns in temperature, and at the same time filters and suppresses possible noise and extreme values. We used the polynomial order of 5 and a window length of 11 for filtering. If needed, we can increase or decrease the values, based on observed patterns in the data to help the algorithm filter noise more accurately.

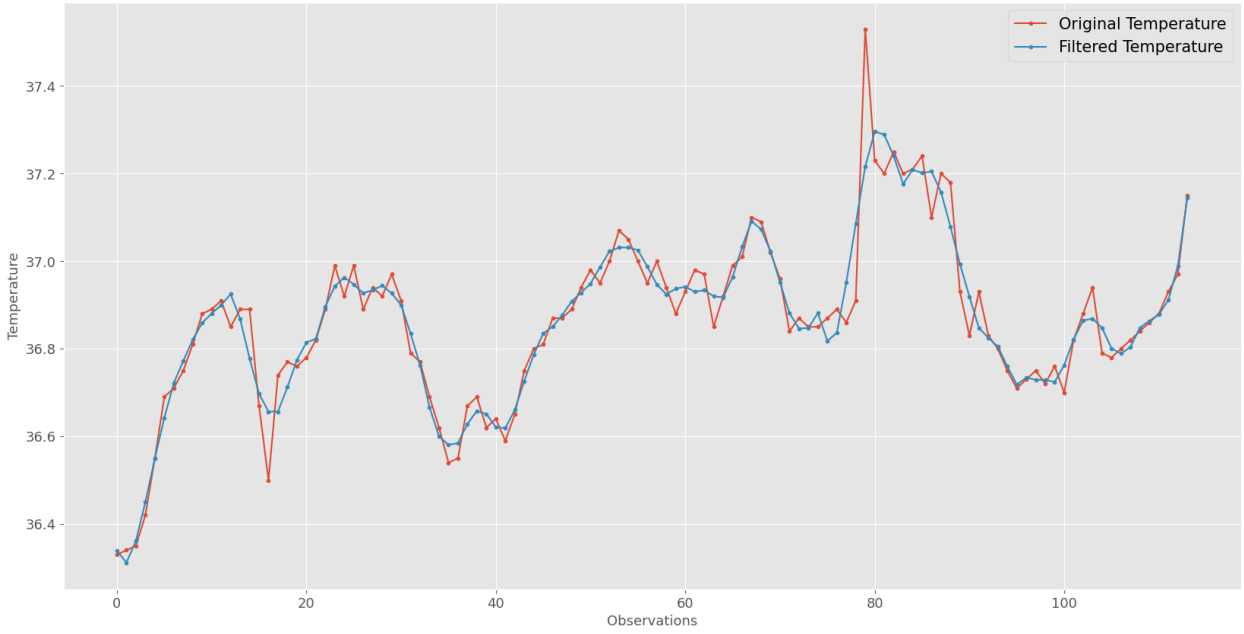


Figure 11.4.1: Beaver1 body temperature with and without Savitzky-Golay filtering

11.4.2 Baseline Removal

Raman spectra are widely used in different scientific fields that focus on studying macromolecules. It allows both chemical and physical structural analysis of materials, using a small sample, without damaging the samples. This is used by law enforcement agencies for identifying contraband items, without physically inspecting them. It can also be used for detecting diseases [3], without any need for further medical diagnostics.

Despite its usefulness, Raman spectra has one issue that needs to be taken care of before using. It carries a background, otherwise known as the baseline. Unless treated and removed, the baseline can cause negative effects in the qualitative and quantitative analysis of spectra. Hence, Raman spectra are fitted and corrected to mitigate this negative influence before being used. There are many methods of correcting the baseline. We will discuss 3 methods with the help of the companion python library BaselineRemoval and see how the three algorithms can help remove background from the spectra.

Modified multi-polynomial fit, also known as ModPoly uses thresholding, to iteratively fit a polynomial baseline to data. Its limitation is that it is prone to variability in data which has a low signal-to-noise ratio. It can smoothen the spectrum by automatically eliminating Raman peaks and leaving behind baseline fluorescence, which can finally be subtracted from the raw spectrum. It uses least-square polynomial fitting functions. Data points are generated from this curve. Data points with higher values than the respective input values are assigned to the original intensity. This exercise is repeated for several iterations, between 25 to 200. The number of repetitions depends on factors such as the relative amount of fluorescence to Raman.

There are some major limitations for ModPoly, as it is dependent on the spectral fitting range and the polynomial order specified. It might not be an ideal solution for high-noise situations, as noise is not appropriately dealt with in ModPoly. ModPoly tends to introduce artificial peaks in the data in places where the original spectrum was free of such peaks. Also, existing large peaks in the data tend to contribute more to the polynomial fitting, which can in turn bias in the results.

Improved ModPoly, also known as IModPoly is an improvement on the ModPoly algorithm and is meant for noisy data. Identifying and removing major peaks is limited to the first iteration only. This prevents unnecessary data rejection. For each iteration of polynomial fitting, lower values of the wave number are selected and concatenated. This is used for constructing a modified spectrum. This in turn is then fitted again. Despite the improved version of the algorithm, IModPoly, just like its predecessor, requires user intervention and prior information, such as detected peaks.

A new method was proposed by Zhang [4], which doesn't require any user intervention and prior information, such as detected peaks. It is named adaptive iteratively reweighted penalized least squares. It is a fast and flexible method that performs adaptive iteratively reweighted penalized least squares. This helps in approximating complex baselines. In each iteration, weights are obtained adaptively by using SSE between a previously fitted baseline and original signals. It uses a penalty approach to control the

smoothness of the fitted baseline. It does so by using the sum squared derivatives of the fitted baseline. Lambda is a parameter controlled by the user. Larger the lambda, the smoother the fitted vector.

Let's now look at data distribution for original spectra and baseline corrected spectra for the skimmed milk samples discussed in chapter 1.

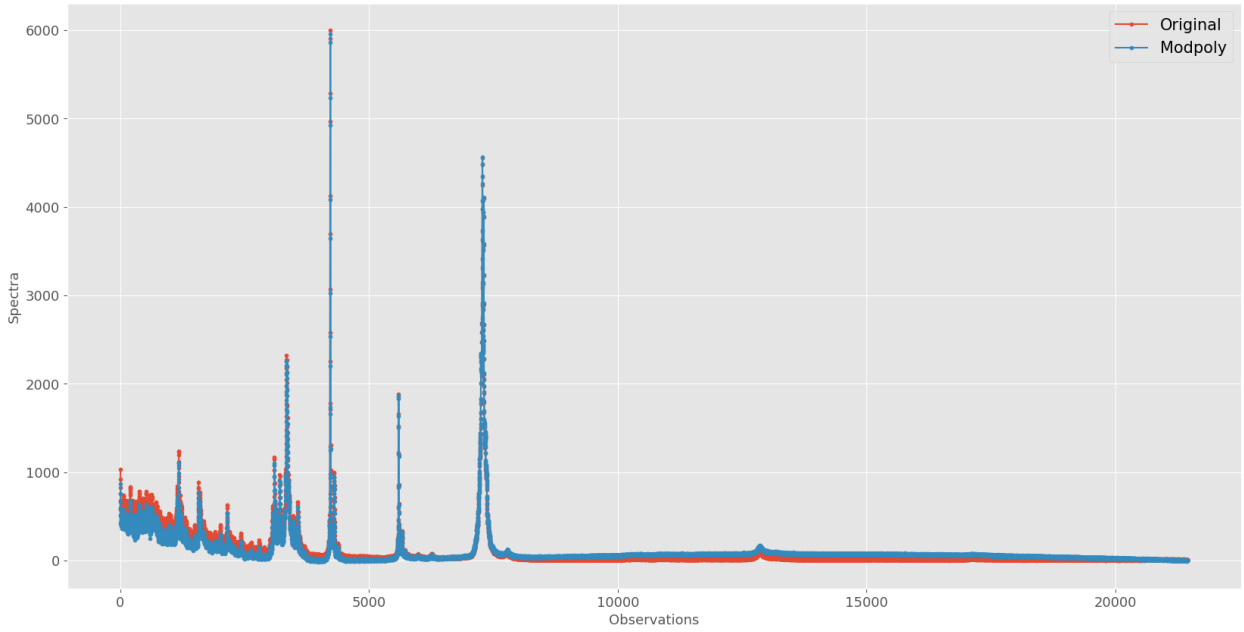


Figure 11.4.2.1: Original Vs. ModPoly corrected spectra of skimmed milk samples

We can see in figure 11.4.2.1 that artificial peaks are introduced by ModPoly for observations at 8000 till 20000. In this region, ModPoly is higher than the original spectrum.

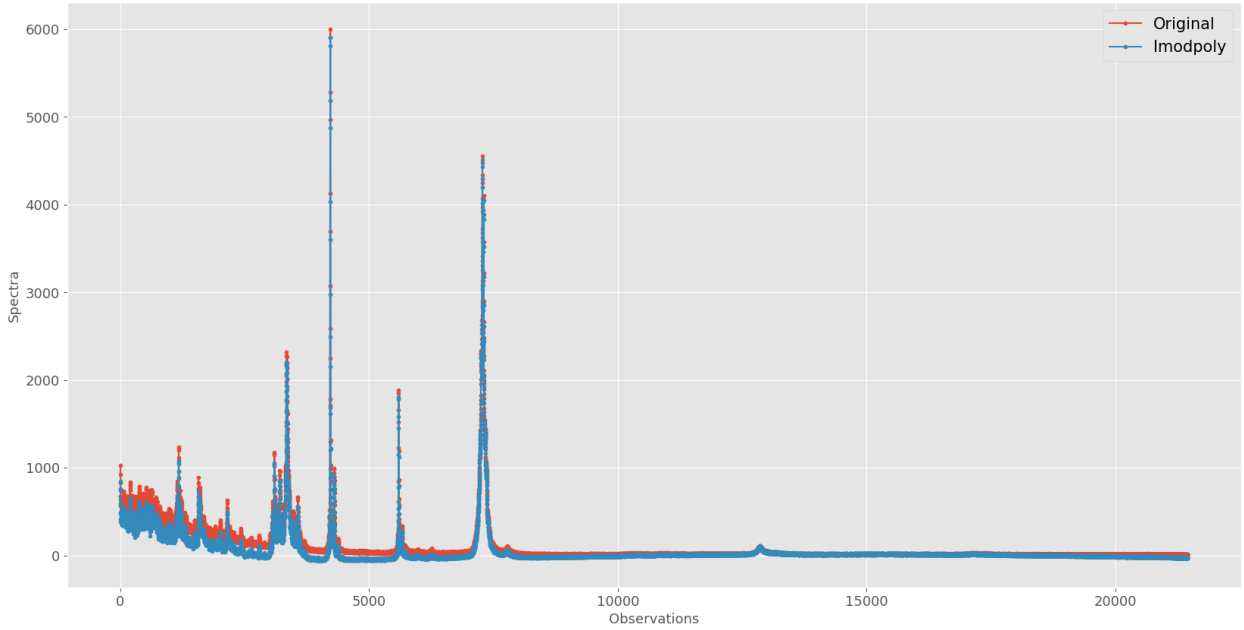


Figure 11.4.2.2: Original Vs. IModPoly corrected spectra of skimmed milk samples

As seen in figure 11.4.2.2, ImodPoly performs better than ModPoly. It removed noise from the spectra, as seen in the plot, between 0 to 8000, and further after 19000. Also, unlike ModPoly, it didn't add an artificial peak.

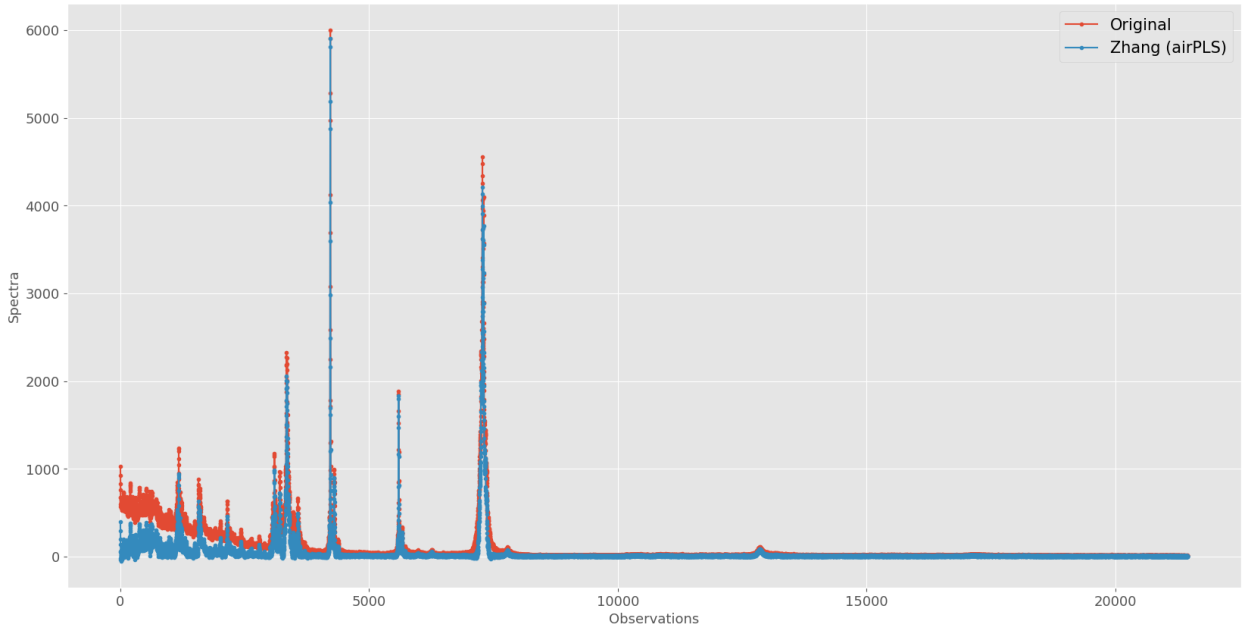


Figure 11.4.2.3: Original Vs. airPLS corrected spectra of skimmed milk samples

We can see in figure 11.4.2.3 for the airPLS method that it removed noise from spectra better than previous algorithms. Especially, for observations between 0 and 2500. As the denoised spectra in this section resemble closely with the rest of the spectra.

11.5 Conclusion

We should honor all the fine details in a dataset. If a dataset has geospatial properties, we should account these properties as features in the model. If the features we obtained as signals from devices, we should perform suitable signal processing techniques before developing a model. Doing so will ensure we have information rich features, less outliers and better performing models.

11.6 References

[1] Jing Bi, Haitao Yuan, LiBo Zhang, Jia Zhang, SGW-SCN: An integrated machine learning approach for workload forecasting in geo-distributed cloud data centers∗∗This paper belongs to the special issue special issue name edited by “Prof. W. Pedrycz.”, Information Sciences, Volume 481, 2019, Pages 57-68, ISSN 0020-0255, https://doi.org/10.1016/j.ins.2018.12.027.

[2] U. Shukla, G. J. Saxena, M. Kumar, A. S. Bafila, A. Pundir and S. Singh, "An Improved Decision Support System for Identification of Abnormal EEG Signals Using a 1D Convolutional Neural Network and Savitzky-Golay Filtering," in IEEE Access, vol. 9, pp. 163492-163503, 2021, doi: 10.1109/ACCESS.2021.3133326.

[3] González-Solís, J.L., Martínez-Espinosa, J.C., Torres-González, L.A. et al. Cervical cancer detection based on serum sample Raman spectroscopy. Lasers Med Sci 29, 979–985 (2014). https://doi.org/10.1007/s10103-013-1447-6

[4] Zhang ZM, Chen S, Liang YZ. Baseline correction using adaptive iteratively reweighted penalized least squares. Analyst. 2010 May;135(5):1138-46. doi: 10.1039/b922045c. Epub 2010 Feb 19. PMID: 20419267.