VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY
INTERNATIONAL UNIVERSITY
DEPARTMENT OF MATHEMATICS

GRADUATION THESIS

SALE PREDICTION USING MACHINE LEARNING ALGORITHMS

Submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF SCIENCE

IN APPLIED MATHEMATICS

SPECIALIZATION IN FINANCIAL ENGINEERING

AND RISK MANAGEMENT

Student's Name:	Tran Thanh Dat
Student's ID:	MAMAIU17036
Thesis Supervisor:	Nguyen Minh Quan, Ph.D

Ho Chi Minh City, Vietnam

August 2022

SALE PREDICTION USING MACHINE LEARNING ALGORITHMS

By

Tran Thanh Dat

Submitted to DEPARTMENT OF MATHEMATICS,
INTERNATIONAL UNIVERSITY, HO CHI MINH CITY
in partial fulfillment of requirements for the degree of
BACHELOR OF SCIENCE
IN APPLIED MATHEMATICS
SPECIALIZATION IN FINANCIAL ENGINEERING
AND RISK MANAGEMENT
August 2022

Signature of Student: [illegible]

Tran Thanh Dat

Certified by: [illegible]

Nguyen Minh Quan, PhD.
Thesis Supervisor

Approved by: ______

Prof. Pham Huu Anh Ngoc, PhD.

Declaration

I hereby declare that this thesis represents my own work which has been done after registration for the degree of Bachelor at International University and has not been previously included in a thesis or dissertation submitted to this or any other institution for a degree, diploma, or other qualifications. I have read the University's current research ethics guidelines, and accept responsibility for the conduct of the procedures in accordance with the University's Committee. I have attempted to identify all the risks related to this research that may arise in conducting this research obtained the relevant ethical and/or safety approval (where applicable), and acknowledged my obligations and the rights of the participants.

Acknowledgements

First and foremost, I would like to express my deep and sincere gratitude to my thesis supervisor, Dr. Nguyen Minh Quan, who directly enthusiastically guided as well as provided the necessary documents and scientific information during the process of the graduation thesis. Although he is very busy with teaching, he always encourages, guides, and helps me very conscientiously.

Besides my supervisor, I sincerely thank the lecturers in the Department of Mathematics, International University for their enthusiasm in conveying a lot of useful knowledge. The knowledge to be acquired in the learning process is not only a platform for the research thesis but it is also valuable to inventory them to apply in practice firmly and confidently.

Last but not least, many thanks go to my parent for their support of me in all the mental strength and physical, who were always beside me with consolation and encouragement. I will be grateful forever for their love.

Thank you very much!

Abstract

This thesis is to investigate the sales prediction accuracy by trial and error from different machine learning models and study the accuracy of these models. We perform predictions based on machine learning techniques on shops' total sales with two main approaches: Least square regression, and Tree-based methods. We collect and analyze the Shopee data and use the two approaches. With the evaluation using two metrics Mean Absolute Errors (MAE) and Soft Interval Accuracy (SIA), the model that gives the best results is random forests. We show that the regression shrinkage models can provide faster results with a trade-off of slightly higher errors.

Key words: Sales forecast, Forecast accuracy, Multiple Linear regression, Ridge Regression, LASSO Regression, Decision Tree, Random Forest, Shopee.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Data scientists define machine learning as a set of tools for making inferences and predictions from data. While prediction is about the outcome of future events, the inference is vaguer because it is about drawing insights. Machine learning methods are taken primarily from statistics and computer science. Essentially, the models learn patterns from existing data and apply it to new data. In academic study, machine learning is widely used in stock market trading with its characteristics of there is always a risk of ups and downs in price shares.

In our daily life, we are using machine learning even without knowing it. It is due to the fact that Machine learning is widely used by various e-commerce and entertainment companies such as Amazon, Netflix and Google for product recommendations to the user. Among all of the aforementioned applications, machine learning algorithms could also be usefully applied in sale prediction. The prediction is utilized across many different industries from small businesses to corporations for tasks such as financial planning, customer success management, and supply chain control. Decision-makers in businesses need advanced warnings of any potential issues, such as in the event of an unexpected downturn in economic conditions. Understanding the importance of sales prediction enables them to take preemptive action to avoid losses in traffic or sales while allowing their businesses to become more malleable in adjusting to new and sudden changes. Yet for most companies coming up with an accurate sales prediction is still a major challenge. For machine learning to be successful, it needs high-quality data and a scientist being able to discover and communicate insights from the data.

One suggestion for high-quality data could be obtained from Shopee. The quality of the data is high since the company's business expands ever-increasingly over the past decade, making the brand one of the most popular e-commerce platforms all over the cities in Vietnam. When visiting the shop homepage, we select the categories and then call the query by page to get data for the search set. Having collected data from Shopee, the author performed machine learning prediction on shops' total sales with two main approaches, Least Square Regression and Tree-based Methods. The prediction was then evaluated using two metrics Mean Absolute Errors (MAE) and Soft Interval Accuracy (SIA).

The purpose of this thesis is to investigate how to increase sales prediction accuracy by trial and error from different machine learning models and explain why the highest accuracy models register themselves as the best ones. In this thesis, five main machine learning algorithms are described and analyzed to determine how effective each technique is at predicting future performance. The algorithms examined are Linear Regression, Ridge Regression, LASSO Regression Decision Trees and Random Forest. Typically, linear regression is used to predict programming performance and while this is a well-regarded statistical technique it is restricted by underlying assumptions, including normal distribution and linear relationship requirements. When these assumptions are not satisfied, subsequent work arouses, leading to interpretation problems and other difficulties. Machine learning algorithms tend to have less stringent requirements and thus may be used to solve problems where linear regression fails. This work provides the foundation for future work on the application of artificial intelligence techniques to this problem and encourages computer science.

For the main limitations, because there are multiple distinct machine learning approaches to use when constructing a sales prediction model, the authors may have included additional strategies in the research. The thesis work could also investigate a larger dataset that can fully generalize the model to all cities in Vietnam.

One expectation is that inaccuracies in forecasts are not necessarily because of the forecasting technique but can be a result of an unorganized forecasting process and inefficient information data. Another expected outcome could be that it is important not only to review the machine learning models within the data but also to modify the model hyperparameters rather than the default values so that data scientists can improve a forecast's accuracy.

Along with the research and analysis of machine learning model algorithms, the implementation of the complex mathematical models will be closely explained with programming languages as well as why we should use programming languages to solve real-world problems with huge amounts of data.

This Thesis clearly acknowledges many papers and books, the chiefly crucial of which is An Introduction to Statistical Learning [1]. Other materials are mentioned to facilitate the use of these statistical learning techniques by practitioners in science, industry,

and other fields, each chapter contains a tutorial on implementing the analyses and methods.
The structural representation of this thesis could be briefly described as follow. Firstly, we discuss some important literature reviews to illustrate the author's inspiration for making the focus on the researched objects. The next chapter, Materials Methodology, studies the theoretical approach of the statistical tools needed to perform the machine learning algorithms. The algorithms will then be individually applied to the empirical data so that the comparisons could be made and turned into informative sale predictions for future data. After discussing the research limitation of the research, we will come to the conclusion with further orientation for the researched results.

Chapter 2

LITERATURE REVIEW

2.1 Motivation of the models

2.1.1 Linear Models

The problem of selecting grouped variables (factors) for accurate prediction in regression arises naturally in many practical situations with the multi-factor analysis and has been discussed by Ming Yuan and Yi Lin [2]. We focus on the accuracy of estimation and consider extensions of the LASSO, reducing the features list for factor selection. The LASSO recently proposed strong regression methods that can be used to select individual variables. We study and propose efficient algorithms for the extensions of these methods for factor selection and show that these extensions give superior performance to the traditional approach.

We show that for this the general situation, a Group LASSO with a different choice of penalty is generally more effective. We give insight into this formulation and show that it is intimately related to the uniformly most powerful invariant test for the inclusion of a group. We demonstrate the efficacy of this method – the "Hyperparameter tuning LASSO"– over the usual LASSO on real data sets. We also extend this to the Ridge Group LASSO to provide within-group regularization as needed.

Since its first proposal by Tibshirani (1996) [3], the least absolute shrinkage and selection operator (LASSO) has generated much interest in the statistical literature.

2.1.2 Tree-based Methods

The key strength of the LASSO lies in its ability to do simultaneous parameter estimation and variable selection. However, recent research suggests that the traditional LASSO estimator may not be fully efficient, and its model selection result could be inconsistent (Leng et al., 2006; Yuan and Lin, 2007; Zou, 2006) [4]. The major reason accounting for such a deficiency is that LASSO applies the same amount of shrinkage for each regression coefficient.

Although the combination of a dimension-reduction step and a standard classical linear regression procedure may suffice for some problems, with greater accuracy, and possibly greater insight, we may be interested in some of the newer techniques such as the Tree-based methods. These methods can be used for either regression and do not require dimension-reduction preprocessing, although such may be desirable for computational feasibility.

We may be interested in some of the more recent methods, such as the Tree-based methods, even though the combination of a dimension-reduction step with a traditional classical linear regression strategy may be sufficient for some issues, with more accuracy, and perhaps greater insights. These methods are widely applied for regression and do not require dimension-reduction, but it may be desirable to use when it comes to computational feasibility.

Cross-validation may be used to select one or more tuning parameters for the two Tree-based approaches that are discussed in the thesis. Trees require comparatively fewer tuning parameters than other machine learning techniques, and they are frequently less sensitive to the selection of these parameters. Tree-based techniques frequently operate well with default settings and do not require any more adjusting.

2.2 Machine Learning Application

The success of today's enterprises depends on the efficiency and quality of their business processes. Software-based tools are increasingly used to model, analyze, simulate, enact and manage business processes. These tools require formal models of the business processes under consideration, which are called workflow models in the following. Acquiring workflow models and adapting them to changing requirements is a time-consuming and error-prone task, because process knowledge is usually distributed among many different people and because workflow modeling is a difficult task, that needs to be done by modeling experts [5]. Thus there has been interest in applying machine learning techniques to induce workflow models from traces of manually enacted workflow instances. The learning algorithms, we are aware of, share some restrictions, that may prevent them from being used in practice. They either apply grammatical inference techniques and are restricted to sequential workflows [5] or they allow concurrency but require unique activity nodes.

Chapter 3

MATERIALS AND METHODOLOGY

3.1 Machine Learning Algorithms

There are three types of machine learning. The first listed, reinforcement learning, is used for deciding sequential actions, like a robot deciding its path or its next move in a chess game. Reinforcement learning is not as common as the others and uses complex mathematics, like game theory. The most common types are supervised and unsupervised learning. Their main difference lies in their training data. When a model is being built and learning from training data, we call this ”training a model”. This can takes nanoseconds to weeks depending on the size of the data.

3.1.1 Supervised learning

In supervised learning, the training data is ”labeled”, meaning the values of our target are known. The magic of machine learning is that we can analyze many features at once, even the ones we're unsure about and find relationships between different features. We input labels and features as data to train the model.

3.1.2 Unsupervised learning

In unsupervised learning, we don't have labels, only features. We can do this usually with tasks like anomaly detection and clustering, which divides data into groups based on similarity. In reality, data doesn't always come with labels. Either it is too much manual work to label or we don't even know what the labels are. This is when unsupervised learning is fully utilized. In this sense, the model is unsupervised and finds its own patterns.

3.1.3 Machine Learning Workflow

The machine learning workflow comprises four steps that go into building a model. The first step is to extract features. Datasets do not typically come naturally with clear features, so there is work to be done in reformatting the dataset. Additionally, we need to decide what features we want to begin with. After that, we need to split the dataset into two datasets: the test and train dataset. To do this, the training dataset is inputted into a chosen machine learning model. There are many different machine learning models to choose from with different use-cases and levels of complexity. We can't assume the resulting model is going to be usable so it needs to be evaluated. This is why we need to split our dataset into two train and test sets.
We can't assume the resulting model is going to be usable. What would be the best way to evaluate the model. We do not want to use any data used to train the model, because the model has already seen that data. Having the model from train data, we put the test dataset, often called "unseen data", into the model to get the model's predictions. There are many ways we could evaluate the performance of our model including calculating the average error of the predictions or the percent of apartment sale prices that were accurately predicted within a 10% margin.
The first thing to look for when evaluating is overfitting. That's when our model performs great on training data, but poorly on the testing data. It means our model learned the training set by heart and is unable to generalize learnings to new data, which is what we originally want. Essentially, we want the difference between the actual value and the predicted value. This can be the distance between the points and the predicted line. There are many ways to calculate this error, such as mean absolute error, and the average error of the predictions. Whatever metric is chosen, a performance threshold needs to be decided. If our model is predicting 80% of the apartments accurately, our model is ready to use. Otherwise, we could return to training the model to tune its inaccuracy model's parameters. Tuning the model can take time and if the performance is not improving, often it means we do not have enough data.
There are two flavors of supervised learning: classification and regression. The classification consists in assigning a category to an observation. We are predicting a discrete variable, a variable that can only take a few different values. While classification assigns a category, regression assigns a continuous variable, that is, a variable that can

take any value. Sometimes, regression identified the trend but is still bad at predicting and hence, more features could make it more accurate.
After we evaluate our model, we have to decide if the performance is good enough. We can improve our unsatisfied model with 2 straightforward options dimensionality reduction and ensemble methods. A dimension denotes the number of features in the data, so dimensionality reduction essentially means reducing the number of features. It may seem counter-intuitive to remove features but some of the features might be completely irrelevant and unlikely to be very useful. Moreover, some features might be highly correlated and carry similar information. We could keep only one feature and still have the most information. We could also collapse multiple features into just one underlying feature. The other option is ensemble methods. This is a technique that combines several models in order to produce one optimal model. With three different models: models A, B and C. In a classification setting, we would use voting. If Model A and C say the event is accepted, and model B says otherwise, then the observation is accepted, as it was the most common prediction amongst all models.

3.2 Resampling Methods

Resampling techniques are an important tool in statistics. They include taking samples from a training set on a regular basis and refitting a model of interest on each sample in order to gather more information about the fitted model. We can repeatedly take different samples from the training data, fit a linear regression to each new sample, and then assess how much the resulting fits vary in order to quantify the variability of a linear regression fit. We may be able to gather data using this method that would not be possible if the model were only fitted once using the initial training sample. Because they require fitting the same statistical procedure more than once using different subsets of the training data, resampling approaches can be computationally expensive.
Many statistical learning approaches can be used practically by using the two most popular resampling techniques, cross-validation and bootstrap. Cross-validation can be used to choose the right level of flexibility or to estimate the test error linked to a specific statistical learning approach in order to assess its performance. The process of evaluating a model's performance is known as model evaluation, whereas the process of selecting the correct level of flexibility for a model is known as model selection. The bootstrap is employed in a variety of situations but is most frequently used to assess the precision of a parameter estimate or a specific statistical learning method.

3.2.1 Cross Validation

The test error, also known as a measurement that was not utilized in the technique's training, is the typical error that occurs when a statistical method is used to forecast the outcome of a fresh observation. Given a data set, the employment of a certain statistical procedure is warranted if it results in a low test error. If a designated test set is provided, the test error can be simply determined. Unfortunately, this is frequently not the case. On the other hand, the training error can be quickly determined by using the statistical learning method on the training observations. The test error rate and the training error rate are typically very different, with the former frequently significantly underestimating the latter.
In the absence of a very large specified test set that can be used directly, there are several methods that can be used to estimate the test error rate using existing training data. Some techniques calculate the test error rate by adjusting the training error rate mathematically. We examine a set of techniques for calculating the test error rate by withholding a portion of the training data from the fitting procedure and then applying the statistical learning technique to the withheld data.

3.2.2 The Bootstrap

The bootstrap is a very versatile and potent statistical tool that may be used to measure the degree of uncertainty surrounding a certain estimator or statistical learning technique. The strength of the bootstrap comes from the ease with which it may be used in a variety of statistical learning techniques, including some for which a measure of variability is otherwise challenging to get and is not automatically output by statistical software.
Bootstrapping entails repeatedly sampling with replacements from the real data in order to describe the characteristics of empirical estimators using the sample data points themselves.
Suppose it is desired to estimate some parameter θ of a sample of data, $y = y_1, y_2, ..., y_T$. By looking at a sample of bootstrap estimators, one can approximate the statistical characteristics of θ. This is accomplished by taking N samples of size T, replacing y with each new sample, and recalculating. The distribution of the following estimates can be considered.
Since the distribution used will be that of the actual data, bootstrapping has an advantage over using analytical results in that it enables the researcher to draw conclusions without making significant distributional assumptions. By examining the fluctuation of

the statistic within-sample, bootstrapping includes experimentally estimating the sampling distribution of the value rather than imposing a shape on it. The test statistic of interest is generated from each set of fresh samples that are drawn via replacement from the sample. Treating the sample as a population from which samples can be taken, effectively entails sampling from the sample. Calculated test statistics using the new samples; since some observations might be sampled multiple times and others not at all, the samples are likely to be substantially different from one another and from the original value. As a result, a distribution of values is obtained, and this distribution can be used to determine standard errors or other relevant statistics.

3.3 Least Squares Regression

Independently discovered by Legendre [6] and Gauss [7], the least-squares regression is one of the most standard and widely used statistical methods in data-fitting. We use the settings as in Brooks [8] and Amemiya [9].

3.3.1 Linear Regression

Suppose that a researcher believes that there is a linear relationship between two variables, say x and y, that he or she is concerned about. The researcher then attempts to describe this relationship as accurate as possible, by the *regression line*

$$y = \alpha + \beta x, \tag{3.1}$$

where α and β are unknown parameters. In particular, the researcher makes n observations to obtain n data points $(x_1, y_1), ..., (x_n, y_n)$ and set

$$\hat{y}_i = \hat{\alpha} + \hat{\beta} x_i, \forall i = \overline{1, n}. \tag{3.2}$$

Here the $\hat{y}_i$s are called *fitted values* from the regression line, while $\hat{\alpha}$ and $\hat{\beta}$ are the *estimators* (for α and β) to be determined. The *residual* of each data point is then given by

$$\hat{u}_i = y_i - \hat{y}_i, \forall i = \overline{1, n}. \tag{3.3}$$

The least squares method, also known in this case as the Ordinary Least Squares (OLS) method, chooses $\hat{\alpha}$ and $\hat{\beta}$ that minimize the *residual sum of squares* (RSS) defined as

$$L := \sum_{i=1}^{n} \hat{u}_i{}^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - \hat{\alpha} - \hat{\beta} x_i)^2.$$

Setting the first order partial derivatives of L with respect to $\hat{\alpha}$ and $\hat{\beta}$ equal to zero

$$\frac{\partial L}{\partial \hat{\alpha}} = -2\sum_{i=1}^{n}(y_i - \hat{\alpha} - \hat{\beta}x_i) = 0 \quad \text{and} \quad \frac{\partial L}{\partial \hat{\beta}} = -2\sum_{i=1}^{n}x_i(y_i - \hat{\alpha} - \hat{\beta}x_i) = 0.$$

It yields the OLS estimators for $\hat{\alpha}$ and $\hat{\beta}$ as

$$\hat{\beta} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2} \quad \text{and} \quad \hat{\alpha} = \overline{y} - \hat{\beta}\overline{x}, \tag{3.4}$$

where $\overline{x}, \overline{y}$ are the sample means of the observations.
Note that equations (3.2) and (3.3) implies

$$y_i = \hat{\alpha} + \hat{\beta}x_i + \hat{u}_i, \forall i = \overline{1,n}. \tag{3.5}$$

This implies that it is rarely the case where the regression line (3.1) fits the data perfectly; there usually be some *random disturbances* or *errors* $\hat{u}_i$.

Classical Linear Regression Model

We rewrite equation (3.5) into the form of a *model*

$$y_i = \alpha + \beta x_i + u_i, \forall i = \overline{1,n}. \tag{3.6}$$

We assume that the following constraints hold, known as the Gauss – Markov assumptions:

The errors have zero mean and the same variance $\sigma^2 < \infty$:

$$\mathbb{E}(u_i) = 0 \quad \text{and} \quad \mathrm{Var}(u_i) = \sigma^2 < \infty, \forall i = \overline{1,n}.$$

The errors are pairwise uncorrelated:

$$\mathrm{Cov}(u_i, u_j) = 0, \forall 1 \leq i < j \leq n.$$

The errors are uncorrelated with the regressor:

$$\mathrm{Cov}(x_i, u_i) = 0, \forall i = \overline{1,n}.$$

The model (3.6), together with the Gauss-Markov assumptions, is called the Classical Linear Regression Model (CLRM). The Gauss-Markov theorem proves that the OLS estimators (3.4) are the Best Linear Unbiased Estimators (BLUE) for the CLRM or in short the optimal estimators. For the purpose of making statistical inferences, the errors are assumed to follow a normal distribution and thus the first Gauss-Markov assumption can be restated as

$$u_i \sim \mathcal{N}(0, \sigma^2), \forall i = \overline{1,n}.$$

Multiple Linear Regression

In practice, given a variable y, it is usually the case that y depends on more than one factor (i.e. independent variable). For dealing with such cases, it is natural to develop an extension of the CLRM as

$$y_i = \beta_1 + \sum_{j=2}^{k} \beta_j x_{j,i} + u_i, \forall i = \overline{1,n}, \tag{3.7}$$

where $x_1 = 1; x_2, ..., x_k$ are $k-1$ independent variables, $\beta_1, ..., \beta_k$ are unknown parameters and $(x_{2,1}, ..., x_{k,1}, y_1), ...(x_{2,n}, ..., x_{k,n}, y_n)$ are the observations. Set $y = (y_1, y_2, ..., y_n)^T$, $\beta = (\beta_1, ..., \beta_k)^T, u = (u_1, ..., u_n)^T$ and

$$X = (x_1, ..., x_k) = \begin{pmatrix} x_{1,1} & ... & x_{k,1} \\ ... & x_{j,i} & ... \\ x_{1,n} & ... & x_{k,n} \end{pmatrix}, i = \overline{1,n}, j = \overline{1,k}.$$

Then the model (3.7) can be expressed alternatively in matrix form as

$$y = X\beta + u, \tag{3.8}$$

with the following Gauss – Markov assumptions:

The errors have zero mean and the same variance $\sigma^2 < \infty$:

$$\mathbb{E}(u_i) = 0 \quad \text{and} \quad \text{Var}(u_i) = \sigma^2 < \infty, \forall i = \overline{1,n}. \tag{3.9}$$

The errors are pairwise uncorrelated:

$$\text{cov}(u_i, u_j) = 0, \forall 1 \leq i < j \leq n. \tag{3.10}$$

$\{x_i\}_{i=1}^{n}$ are non-stochastic/fixed; X has full column rank, i.e. $\text{rank}(X) = k$. (3.11)

The RSS is now given by

$$L = \sum_{i=1}^{n} \hat{u_i}^2 = \hat{u}^T\hat{u} = (y - X\hat{\beta})^T(y - X\hat{\beta}) = y^Ty - 2\hat{\beta}^TX^Ty + \hat{\beta}^TX^TX\hat{\beta}$$

and the first order partial derivatives are

$$\frac{\partial L}{\partial \hat{\beta}} = -2X^Ty + 2X^TX\hat{\beta} = 0.$$

It implies the OLS estimator for $\hat{\beta}$ as

$$\hat{\beta} = (X^TX)^{-1}X^Ty. \tag{3.12}$$

We denote

$$\mathrm{Var}(u_i) = \sigma_i^2 = \sigma_{ii}, \mathrm{cov}(u_i, u_j) = \sigma_{ij} = \sigma_{ji}, \forall 1 \leq i < j \leq n$$

and

$$\mathbb{E}(u) = (\mathbb{E}(u_1), ..., \mathbb{E}(u_n))^T, \mathrm{Var}(u) = \Sigma = \begin{pmatrix} \sigma_{1,1} & ... & \sigma_{1,n} \\ ... & \sigma_{i,j} & ... \\ \sigma_{n,1} & ... & \sigma_{n,n} \end{pmatrix}.$$

The assumptions (3.9) and (3.10) can be rewritten as

The errors have zero mean, i.e. $\mathbb{E}(u) = \mathbf{0}$. The errors are pairwise uncorrelated and have the same variance $\sigma^2 < \infty$, i.e.

$$\Sigma = \sigma^2 I_n = \begin{pmatrix} \sigma^2 & 0 & ... & 0 \\ 0 & \sigma^2 & ... & 0 \\ ... & ... & ... & ... \\ 0 & ... & 0 & \sigma^2 \end{pmatrix}, \tag{3.13}$$

where I_n is the $n \times n$ identity matrix.

Note that the full-rank assumption (3.11) is crucial for the OLS estimator formula (3.12) since otherwise the matrix $X^T X$ is not invertible.

Validation of Model

In many situations, it turns out that the regression model delivered by the above construction provides unsatisfied results, e.g. using it for further research purposes yields meaningless conclusions. For such cases, problems may come from the data (which may not follow the Gauss-Markov assumptions) or the user (who may have attempted to investigate some variables which in reality have no correlation). Hence to avoid these issues, researchers are required to test the *validity* of the model, before using it, with the following metrics:

The R^2 ($R-$squared), as known as coefficient determination, determined by the formula

$$R^2 = 1 - \frac{RSS}{TSS} \quad \text{where} \quad TSS = \sum_{i=1}^{n} (y_i - \overline{y})^2.$$

The value R^2 measures the accuracy of the model. An R^2 closer to 1 (0) means high (low) accuracy. For example, a coefficient determination is equal to 0.8 means that the model explains 80% of the variation of the errors. The Goldfeld-Quandt (GQ) test and White's test can be used for confirming the presence of *heteroscedasticity*, i.e. the errors' variances are changing over time. These tests return $p-$values, which can be interpreted as usual. The Durbin-Watson (DW) test detects signs of *autocorrelation*, i.e. correlated errors. It returns a test statistic DW and two extreme critical values, a lower d_L and an upper d_U; presence of autocorrelation is rejected if $DW \in [d_u, 4-d_u]$ and is accepted if $DW \notin [d_L, 4-d_L]$. An alternative test is the Breusch-Godfrey Test, which gives a $p-$value instead. The Jarqua-Bera (JB) test, which returns a $p-$value, is applicable for normality testing of errors. The Durbin–Wu–Hausman (DWH) test verifies the occurrence of *endogeneity*, i.e. correlation of regressor and errors. This test compares OLS estimates with instrumental variable (IV) estimates and returns a $p-$value.

3.3.2 Ridge Regression

Ridge regression is very similar to least squares, except that the coefficients of ridge regression are estimated by minimizing a slightly different quantity. In particular, the ridge regression coefficient estimates β^R are the values that minimize the Loss function:

$$L = \sum_{i=1}^{n} \hat{u_i}^2 = (y - X\hat{\beta})^T(y - X\hat{\beta}) + \lambda\hat{\beta}^T\hat{\beta}. \tag{3.14}$$

This implies the OLS estimator for $\hat{\beta}$ as

$$\hat{\beta} = (X^TX + \lambda I)^{-1}X^Ty, \tag{3.15}$$

where $\lambda \geq 0$ is a tuning parameter, to be determined separately.
The equation (3.15) trades off two distinct criteria. Ridge regression, like least squares, reduces the RSS to find coefficient estimates that fit the data well. The second term, $\lambda\hat{\beta}^T\hat{\beta}$, known as a shrinkage penalty, is small when $\beta_1, ..., \beta_p$ are close to zero, and thus has the effect of shrinking the estimates of β_j towards zero. The tuning parameter λ governs the relative importance of these two terms in estimating regression coefficients. The penalty term has no effect when $\lambda = 0$, and ridge regression produces least squares estimates. However, as $\lambda \to \infty$ increases, the shrinkage penalty becomes more significant, and the ridge regression coefficient estimates approach zero. In contrast, to least squares, which produce only one set of coefficient estimates, ridge regression produces a different set of coefficient estimates, β^R_λ, for each value of λ. It is critical to choose a good value for λ. If we assume that the variables—that is, the columns of the data matrix X—have been centered to have mean zero before performing ridge regression, then the estimated intercept will be of the form $\hat{\beta}_0 = \bar{y} = \sum_{i=1}^{n} \frac{y_i}{n}$.

3.3.3 LASSO Regression

There is one obvious disadvantage to ridge regression. In contrast to best subset, forward stepwise, and backward stepwise selection, which generally select models that involve only a subset of the variables, ridge regression includes all predictors in the final model.

The penalty $\lambda\hat{\beta}^T\hat{\beta}$ in (3.15) will shrink all of the coefficients towards zero, but it will not set any of them to zero exactly (unless $\lambda = \infty$). This may not be a problem for prediction accuracy, but it can make model interpretation difficult in settings with a large number of variables. Increasing λ tends to reduce the magnitudes of the coefficients but does not result in the exclusion of any of the variables.

The LASSO is a newer alternative to ridge regression that overcomes this limitation. The LASSO coefficients, β_λ^R, minimize the quantity

$$L = \sum_{i=1}^{n} \hat{u_i}^2 = (y - X\hat{\beta})^T(y - X\hat{\beta}) + \lambda||\hat{\beta}||_1. \tag{3.16}$$

Comparing 3.16 to 3.14, we see that the LASSO and ridge regression have similar formulations. The only difference is that the $\hat{\beta}^T\hat{\beta}$ term in the ridge regression penalty 3.14 has been replaced by $||\hat{\beta}||_1$ in the LASSO penalty 3.16.

The LASSO, like ridge regression, shrinks the coefficient estimates towards zero. When the tuning parameter λ is sufficiently large, the penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero in the case of the LASSO. As a result, the LASSO performs variable selection in the same way that best subset selection does. As a result, models generated by the LASSO are much easier to interpret than models generated by ridge regression. The LASSO produces sparse models, which are models that involve only a subset of the variables. As with ridge regression, choosing a good λ value for the LASSO is critical.

3.4 Tree-based Methods

Tree-based methods for regression involve stratifying or segmenting the predictor space into a number of simple regions. In order to make a prediction for a given observation, we typically use the mean or the mode response value for the training observations in the region to which it belongs. Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as decision tree methods. We will see that combining a large number of decision trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss in interpretation.

3.4.1 Regression Trees

There are two steps in the process of building a regression tree.

> We divide the predictor space — that is, the set of possible values for $X_1, X_2, ..., X_p$ into J distinct and non-overlapping regions, $R_1, R_2, ..., R_J$. For every observation that falls into the region R_j , we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

The regions can be of any shape. However, for simplicity and ease of interpretation of the resulting predictive model, we choose to divide the predictor space into high-dimensional rectangles, or boxes. The goal is to find $R_1, ..., R_J$ boxes that minimize the RSS given by

$$\sum_{i=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2, \tag{3.17}$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the j^{th} box.
Unfortunately, considering every possible partition of the feature space into J boxes is computationally impossible. As a result, we employ a top-down, greedy strategy known as recursive binary splitting. The recursive binary splitting approach is top-down because it starts at the top of the tree (where all observations belong to a single region) and then splits the predictor space in stages, with each split indicated by two new branches further down the tree. It is greedy because the best split is made at each step of the tree-building process, rather than looking ahead and selecting a split that will lead to a better tree in some future step.
To perform recursive binary splitting, we first choose the predictor X_j and the cutpoints so that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ results in the greatest RSS reduction. (The notation $\{X|X_j < s\}$ denotes the region of predictor space where X_j has a value less than s.) That is, we consider all predictors $X_1, ..., X_p$, as well as all possible cutpoint values for each predictor, and then select the predictor and cutpoints with the lowest RSS. In more detail, we define the pair of half-planes for any j and s.

$$\sum_{i=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2, \tag{3.18}$$

where

$$R_1(j,s) = \{X|X_j < s\} \quad \text{and} \quad R_2(j,s) = \{X|X_j \geq s\}. \tag{3.19}$$

And we seek the value of j and s that minimize the equation

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2, \tag{3.20}$$

where $\hat{y}_{R_1}$ is the mean response for the training observations in $R_1(j,s)$, and $\hat{y}_{R_2}$ is the mean response for the training observations in $R_2(j,s)$.

Finding the values of j and s that minimize (3.20) binary split is a simple task, especially when the number of features p is small. The process is then repeated, looking for the best predictor and best cutpoint to further split the data in order to minimize the RSS within each of the resulting regions. Instead of splitting the entire predictor space, we split one of the two previously identified regions this time. There are now three regions. Again, we seek to further divide one of these three regions in order to reduce the RSS. The process is repeated until a stopping criterion is reached, for example, until no region contains more than five observations.

We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs once the regions $R_1, ..., R_J$ have been created.

Tree Pruning

The process described above may produce good predictions on the training set, but it is likely to overfit the data, resulting in poor performance on the test set. This is due to the possibility that the resulting tree will be overly complex. A smaller tree with fewer splits (i.e., fewer regions $R_1, ..., R_J$) may result in less variance and better interpretation at the expense of some bias.

One variation on the above procedure is to build the tree only if the decrease in RSS caused by each split exceeds a certain (high) threshold. This strategy produces smaller trees, but it is too narrow-minded because a seemingly insignificant split early in the tree may be followed by a very good split—that is, a split that leads to a significant reduction in RSS later on. As a result, growing a very large tree T_0 and then pruning it back to obtain a subtree is preferable.

Our intuitive goal is to choose a subtree that results in the lowest test error rate. We can estimate the test error of a subtree using cross-validation or the validation set approach. However, because there are so many possible subtrees, estimating the cross-validation error for each one would be too time-consuming. Instead, we require a method for selecting a small number of subtrees for consideration. Cost complexity pruning, also known as weakest link pruning, allows us to do exactly that. We consider a sequence of trees indexed by a nonnegative tuning parameter α rather than every possible subtree. For each α value, there is a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T| \tag{3.21}$$

is as small as possible.
Here, $|T|$ represents the number of terminal nodes in the tree T, R_m represents the rectangle (i.e. the subset of predictor space) corresponding to the m^{th} terminal node, and $\hat{y}_{R_m}$ represents the predicted response associated with R_m—that is, the mean of the training observations in R_m. The tuning parameter α controls the trade-off between the complexity of the subtree and its fit to the training data. When $\alpha = 0$, the subtree T simply equals T_0 because 3.21 only measures training error. However, as α rises, there is a cost to having a tree with many terminal nodes, and thus the quantity 3.21 will tend to be minimized for a smaller subtree.
The equation 3.21 used a similar formulation to control the complexity of a linear model. As we increase α from zero in 3.21, branches are pruned from the tree in a nested and predictable manner, making obtaining the entire sequence of subtrees as a function of α simple. We can choose a value for α by using a validation set or cross-validation. Then we return to the entire data set and find the subtree corresponding to α. The following algorithm summarizes this process.

Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, ..., K$:

(3) (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.
 (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

Average the results for each value of α, and pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Trees, in general, do not have the same level of predictive accuracy and can be quite fragile. In other words, a minor change in the data can result in a significant change in the final estimated tree. The predictive performance of trees can be significantly improved by aggregating many decision trees and employing methods such as bagging, random forests, and boosting.

3.4.2 Bagging

Bagging and random forests are ensemble methods with a regression tree as the basic building block. An ensemble method is a method of combining many simple "building block" models to create a single, potentially very powerful model. These simple building block models are sometimes referred to as weak learners because they can only make mediocre predictions on their own.
The bootstrap is a very powerful concept. It is used in many situations where calculating the standard deviation of a quantity of interest directly is difficult, if not impossible. It is clear that the bootstrap can be used to improve statistical learning methods such as decision trees in a completely different context.
The decision trees discussed in Section 8.1 have a high level of variance. This means that if we randomly divide the training data into two halves and apply a decision tree to each half, the results could be quite different. A procedure with low variance, on the other hand, will produce similar results if applied repeatedly to different data sets; linear regression has low variance if the n/p ratio is moderately large. Bagging, also known as bootstrap aggregation, is a general-purpose procedure for reducing the variance of a statistical learning method. It is especially useful and popular in the context of decision trees.
Remember that if you have a set of n independent observations $Z_1, ..., Z_n$ with high variance σ^2, the variance of the mean Z of the observations is given by $\frac{\sigma^2}{n}$. Averaging a set of observations, in other words, reduces variance. As a result, taking many training sets from the population, building a separate prediction model using each training set, and averaging the resulting predictions is a natural way to reduce variance and increase test set accuracy of a statistical learning method. In other words, we could compute $f^1(x), f^2(x), ..., f^B(x)$ using B different training sets and average them to obtain a single low-variance statistical learning model, denoted by

$$\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^b(x).$$

Of course, this is impractical because we do not usually have access to multiple training sets. Instead, we can bootstrap by taking repeated (with replacement) samples from the (single) training data set. We generate B different bootstrapped training data sets using this method. Then, we train our method on the b^{th} bootstrapped training set to get $f^{*b}(x)$, and finally, we average all the predictions to get the bagging

$$\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x).$$

While bagging can help with many regression methods, it is especially useful for decision trees. To apply bagging to regression trees, we simply build B regression trees with B bootstrapped training sets and average the predictions. These trees have deep roots and are not pruned. As a result, each individual tree has a high variance but a low bias. The variance is reduced by averaging these B trees. Bagging has been shown to improve accuracy significantly by combining hundreds or even thousands of trees into a single procedure.

As previously discussed, bagging typically improves prediction accuracy over prediction using a single tree. Unfortunately, the resulting model can be difficult to interpret. Remember that one of the benefits of decision trees is the visually appealing and easily interpreted diagram that results. When we bag a large number of trees, we can no longer represent the resulting statistical learning procedure with a single tree, and it is unclear which variables are most important to the procedure. As a result, bagging improves prediction accuracy while decreasing interpretability.

Although a bagged tree collection is much more difficult to interpret than a single tree, an overall summary of the importance of each predictor can be obtained. In the case of bagging regression trees, we can record the total amount by which the RSS is reduced as a result of splits over a given predictor, averaged across all B trees. A high value indicates a significant predictor.

3.4.3 Random Forest

As in bagging, we construct a number of decision trees from bootstrapped training samples. We build a number of decision trees from bootstrapped training samples, just like in bagging. When creating these decision trees, a random sample of m predictors is chosen as split candidates from the full set of p predictors. Only one of the m predictors may be used in the split. We take a new sample of m predictors at each split, and we usually select $mapproxsqrtp$, which means that the number of predictors considered at each split is roughly equal to the square root of the total number of predictors.

Random Forest Algorithms:

Sample, with replacement, n training examples from X, Y: X_b, Y_b. Train a regression tree f_b on X_b, Y_b. When building decision trees, each time a split in a tree is considered, and a random sample of m predictors is chosen as split candidates from the full set of p predictors. After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x').$$

When constructing a random forest, the algorithm is not even permitted to consider a majority of the available predictors at each split in the tree. Assume the data set contains one very strong predictor and a number of other moderately strong predictors. Then, in the bagged tree collection, most or all of the trees will use this strong predictor in the top split. As a result, all of the bagged trees will look very similar to one another. As a result, predictions from bagged trees will be highly correlated. Unfortunately, averaging many highly correlated quantities does not result in as significant a variance reduction as averaging many uncorrelated quantities. In particular, this means that bagging will not lead to a substantial reduction in variance over a single tree in this setting.

This means that bagging will not result in a significant reduction in variance over a single tree in this setting. Random forests solve this problem by limiting each split to only a subset of the predictors. As a result, on average $\frac{p-m}{p}$ of the splits will not even consider the strong predictor, giving other predictors a better chance. This process can be thought of as decorating the trees, making the average of the resulting trees less variable and thus more reliable.

3.5 Metrics

3.5.1 Mean Absolute Error

It is extremely unlikely to determine the quality of a forecasting model based on only one forecast and one realization. In practice, forecasts would typically be generated for the entire out-of-sample period, which would then be compared to actual values, with the difference aggregated in some way. The forecast error for observation i is defined as the difference between the actual value and the forecast value. As a result,

simply adding the forecast errors does not work because the positive and negative errors cancel each other out. Therefore, before the forecast errors are aggregated, the values are taken absolutely or quadratically, making them all positive.
MSE provides a quadratic loss function, and so may be particularly useful in situations where large forecast errors are disproportionately more serious than smaller errors. This may, however, also be viewed as a disadvantage if large errors are not disproportionately more serious, although the same critique could also, of course, be applied to the whole least squares methodology. Indeed Dielman (1986) [10] goes as far as to say that when there are outliers present, least absolute values should be used to determine model parameters rather than least squares. Makridakis (1993, p. 528) [11] argues that mean absolute percentage error (MAPE) is 'a relative measure that incorporates the best characteristics among the various accuracy criteria'.
Individually, considering the size of the MAE yields little information because the statistic is unbounded from above. Instead, for the same data and forecast period, the MAE from one model would be compared to those of other models, and the model(s) with the lowest value of the error measure would be argued to be the most accurate. The simple mathematical presentation of MAE is:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_{pred} - y|.$$

3.5.2 Soft Interval Accuracy

In addition to MAE, we could find a metric to evaluate the predictive model. That metric should intend to use a metric that uses difference as accuracy, called Soft Interval Accuracy (SIA), which can be described as follows:

$$SIA(\epsilon) = \mathbb{P}(|y_{pred} - y| < \epsilon).$$

However, there is a hyperparameter that must be selected as ϵ. This metric means that given 1 ϵ value, a prediction is considered true (True) when its value is in the range $(y - \epsilon, y + \epsilon)$ or $|y - \hat{y}| < \epsilon$ (where y is the information from the data and $\hat{y}$ is the predicted value).

Chapter 4

PROCESSING AND ANALYSIS

This chapter describes the basics attributes of the data both in general and details information. After that, we will make some plots to describe the distribution of the data. the chapter ends with splitting the original data into training, testing and validating.

4.1 Attributes and scale of the data

When a store in Shopee posted products to sell online, they would be interested in many issues, including the desire to know the selling ability of the product. Based on this psychology, we chose Shopee as the data source for the problem of predicting the selling ability of stores on Shopee.
When going to the Shopee homepage, we select the categories and then call the query by page to get data for the search set. There are a total of 16 categories to choose from AUDIO, CAPSAC, CHUOTBANPHIM, DIENTHOAI, GAYCHUPHINH, LAPTOP, LINHKIENMAYTINH, MAYBAN, MAYTINHBANG, MIENGDAN, OCUNG, PHUKIEN, PINSAC, SIM, THIETBIMANG, VOBAOOPLUNG. Up to 1000 products can be crawled per category. In total, the team crawled 15648 products. The data collection activities should contain as much information as possible.

3.

	Column	**Dtype**	**Description**	**Examp**
0	id	int64	product id	5
1	name	string	product name	Dien Tl
2	category	string	product category	DIENT
3	n_sold	float64	number sold	7000
4	price	float64	product price	290000
5	shop_address	string	shop address	TP. HC
6	image_url	string	product image	https://
7	url	string	url to product	https://

Table 4.1: Attributes and Sample product of the primary table

4.2 Data Exploratory Analysis

We focus on data exploration to support the data modeling and answer the question of how existing attributes will affect the sales forecast of a product.
Initially, it is thought that all the numerical factors crawled would be affected, especially the number of reviews, the number of likes, and the average rating. However, after calculating the correlation, the data shows that only the number of reviews has a sufficiently large correlation. The number of favorites and the average rating is not satisfactory. Based on the observation when collecting data that there are products that only sell 1 or 2 products but the average rating is 5 stars for the number of likes, it is because these are products that users are only interested in buying but do not mark as favorite.

	Column	**Dtype**	**Description**
0	id	int64	product id
1	name	string	product name
2	avg_rating	float64	product average rating
3	n_reviews	int64	product reviews
4	n_sold	float64	number sold
5	price	float64	product price
6	n_loved	int64	product number of love
7	n_rate_5	int64	product number of 5 stars
8	rate_4	int64	product number of 4stars
9	rate_3	int64	product number of 3 stars
10	rate_2	int64	product number of 2 stars
11	rate_1	int64	product number of 1 stars
12	rate_with_cmt	int64	comments
13	rate_with_imgvid	int64	comments with image
14	shop_name	string	shop name
15	shop_n_review	int64	shop reviews
16	shop_n_product	int64	shop storage
17	shop_rate_feedback	float64	shop feedback
18	shop_time_feedback	float64	shop time feedback
19	shop_age	float64	shop age
20	shop_follower	int64	shop followers

Table 4.2: Attributes and Sample product of the attributes table

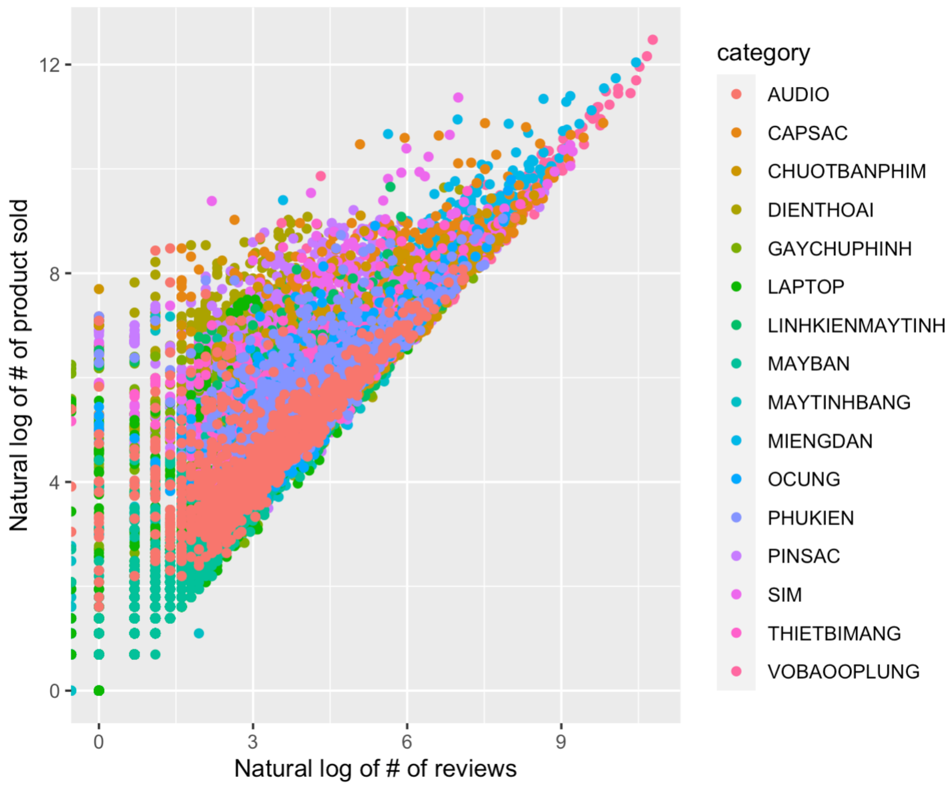

Figure 4.1: Scatter plot comparing number of product sold and number of reviews

Based on this observation, we tried to get the top products selling the least and found that relying on the average rating is not reliable because it is true that many products sold with a minimal number but still 5 stars.

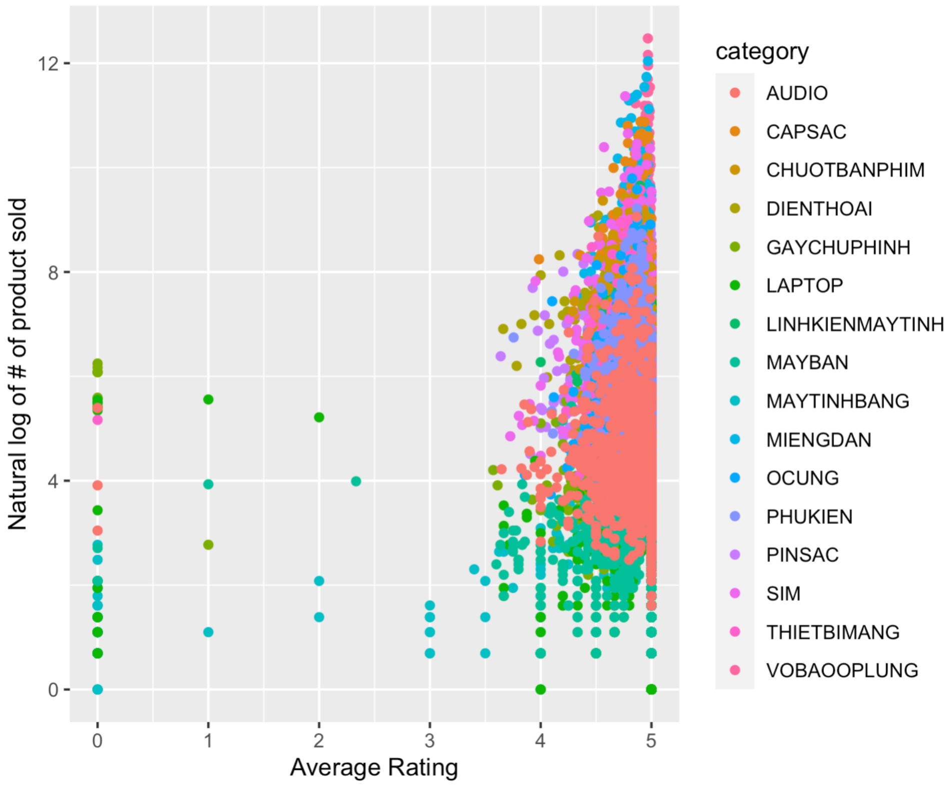

Figure 4.2: Scatter plot comparing number of products sold and rating

Similarly, the group that printed out the products found that the love attribute was very random overall the products were sold a lot but the likes were very low, or medium, or high. However, if we filter it out to get a certain shop name, the correlation value is high, so the number of favorites can still be used in data modeling.

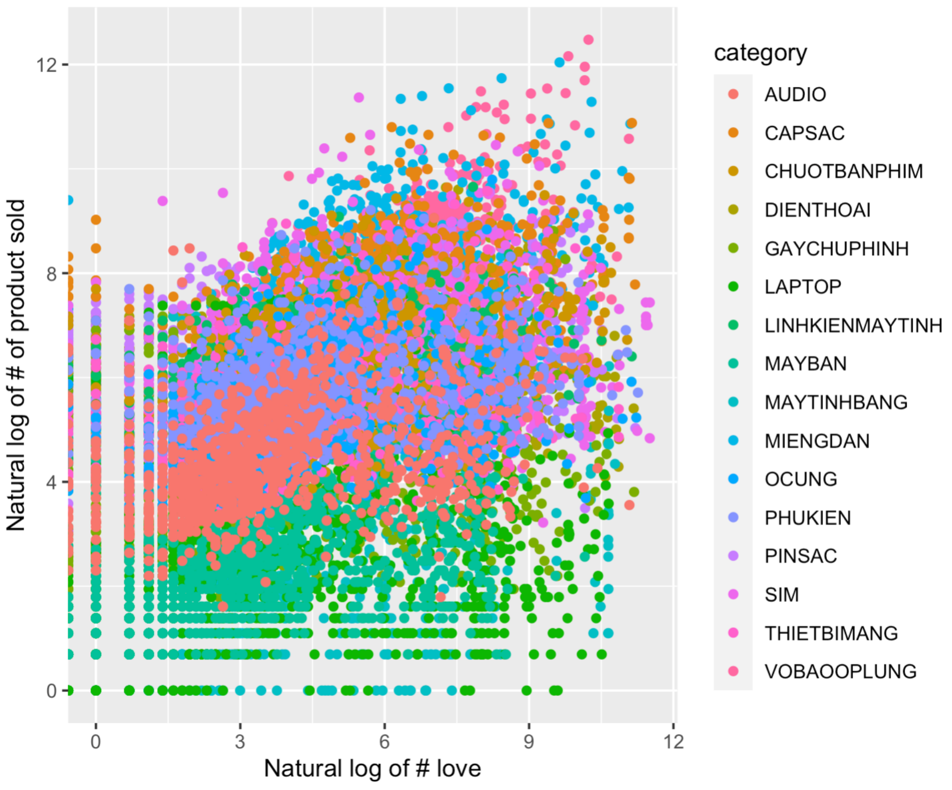

Figure 4.3: Scatter plot comparing number of products sold and number of loves

Furthermore, we could use the histogram to illustrate the distribution of the dependent variable.

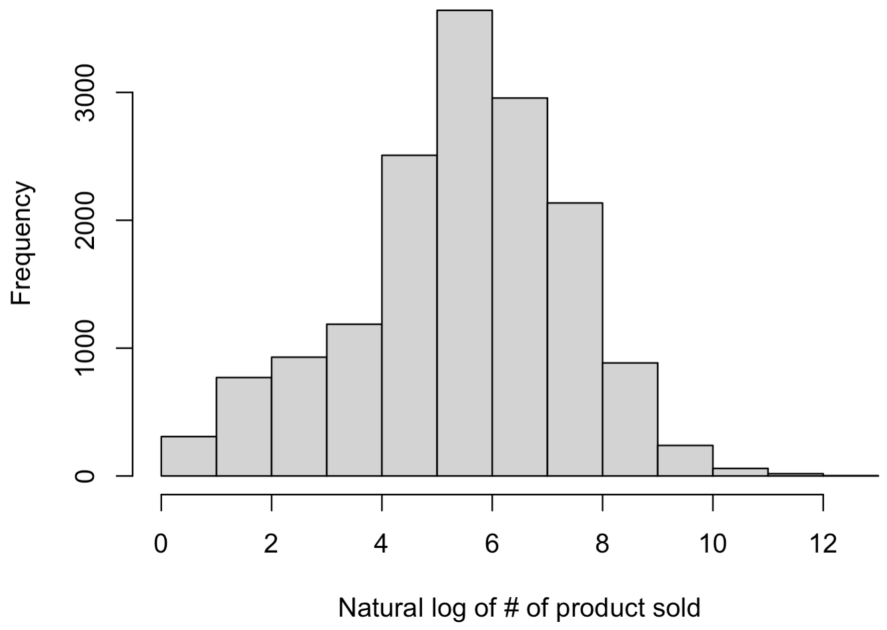

Figure 4.4: Histogram describing the frequency of the number of product sold

4.3 Data Splitting

The data is broken down into three distinct data sets these data sets will consist of a training set a validation set and a test set. The training set is the set of data used to train the model during each iteration, our model will be trained multiple times on this same data in our training set and it will continue to learn about the features of this data the hope with this is that later we can deploy our model and have it accurately predicts new data that it has never been seen before. These predictions should base on what it learned about the training data.

The validation set is a set of data separate from the training set that is used to validate our model during training. This validation process helps give information that may be helpful with adjusting our hyperparameters. With each iteration during training, the model will be training on the training set, it will also be simultaneously validating on the data in the validation set. One of the major reasons we need a the validation set is to ensure that our model is not overfitting to the data in the training set. The idea of

overfitting is that our model becomes really good at being able to classify the data in the training set, it is unable to generalize and make accurate regression on data that it has not been trained on yet. So during training if we are also validating the model on our the validation set and see that the results it's giving for the validation set are just as good as the results it has given for the training data then we can be more confident that our model is not overfitting. On the other hand if the The results of the training data are really good but the results of the validation data are lagging behind then our model is likely to be overfitting
The test set is a set of data that is used to test the model after the model has already been trained this test set is separate from both the training set and validation set. After our model has been trained and validated using our training and validation sets we'll then use our model to predict the output of the data in the test set. One major difference between the test set and the other two sets discussed is that the the test set should not be labeled the training set and validation set have to be labeled so that we can see the metrics are given during training from each iteration so when the model is predicting on unlabeled data in our test set this would be the same type of process that would be used if we were to deploy our model into the field. The entire goal of having a model be able to classify is to do this without knowing what the data is beforehand and expect that we have an the idea about how our data should be organized.

Chapter 5

RESULTS

All of the final results of this thesis are wholly contained in this chapter. After being described by tables, the data will then be plotted as a bar graph for making a comparison between the datasets.

5.1 Models without Hyperparameter tuning

model	**train**	**test**
random_forest	161	469
LASSO	606	613
linear	610	616
ridge	609	616
decision_tree	0.01	641

Table 5.1: Comparing MAE of models in both train and test sets

Looking at Table 5.1, it is absolutely obvious that the Decision Tree model registers the lowest and the highest MAE in the training set and testing set respectively. The lowest number, 0.01 in the train data implies the presence of overfitting, resulting in a bad performance in the test data with an MAE of 641, the highest MAE among the 5 models. By contrast, Random Forest produces the lowest MAE when it comes to the test data set, way lower than that of another number with a difference of 150 MAE or 20%.

Bar plots of the machine learning methods describing the accuracy are made to illustrate the names of the regressors with the corresponding Mean Absolute Errors.

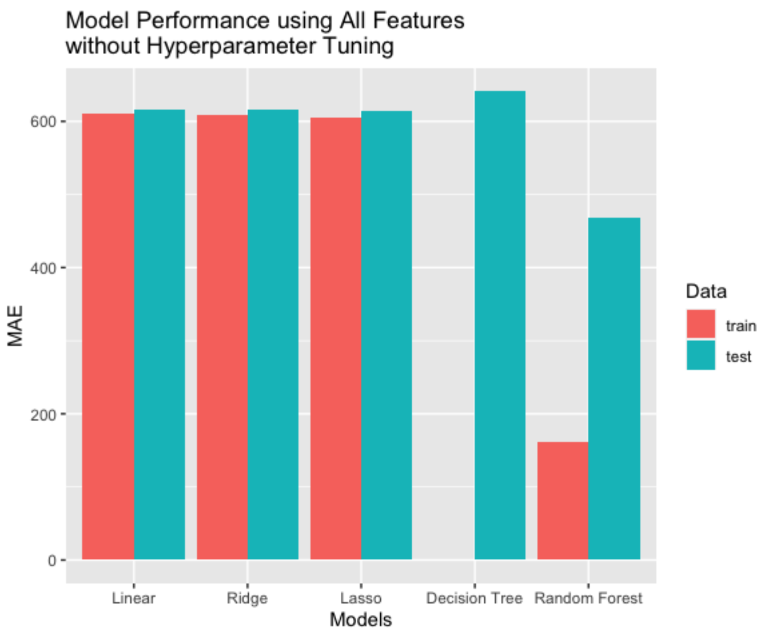

Figure 5.1: Model Performance measured by MAE using All Features without Hyperparameter Tuning

It is also worth noticing that the performance of the Linear Models including Linear Regression, Ridge Regression and LASSO is consistent in the two data sets with the MAE ranging from 605 to 615. However, since the difference between the number is greatly insignificant, lower than 0.1%, we cannot actually draw any conclusion about which Linear Models should be the most suitable. Therefore, we could instead use the SIA metric to observe clearly the model performance

A glance at Figure 5.2 suggests that while Decision Tree outperforms others in the train data, the other Tree-based model Random Forest stands at the greatest SIA among the machine learning models in the test data with the SIA of exactly 84%. It is also noticeable that LASSO is the greatest model among the Linear Models that work approximately the same in MAE metrics with the SIA of train and test being 58% and 57% in that order.

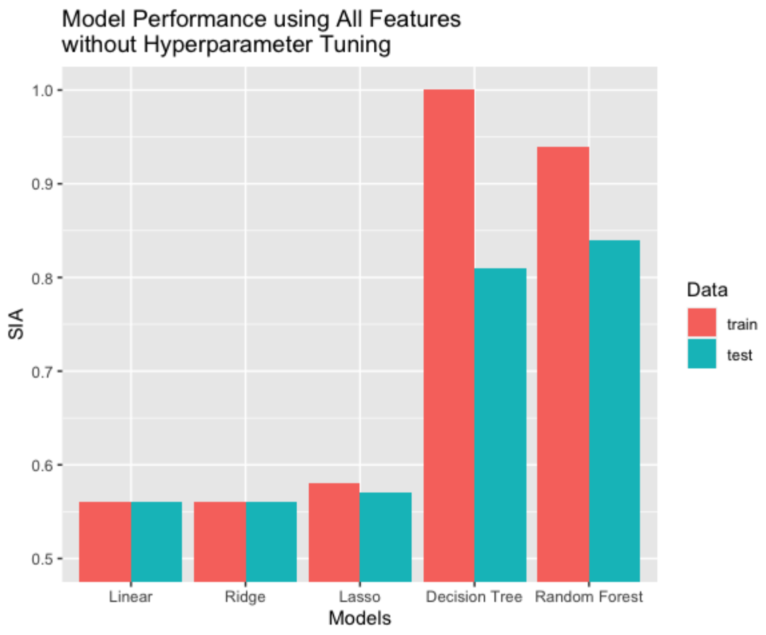

Figure 5.2: Model Performance measured by SIA using All Features without Hyperparameter Tuning

The next investigation would be comparing the performance of those models in all features and those of reduced features. These features are obtained in the LASSO where some of the coefficients of the dependent variable return zero leaving other significant dependent variables.

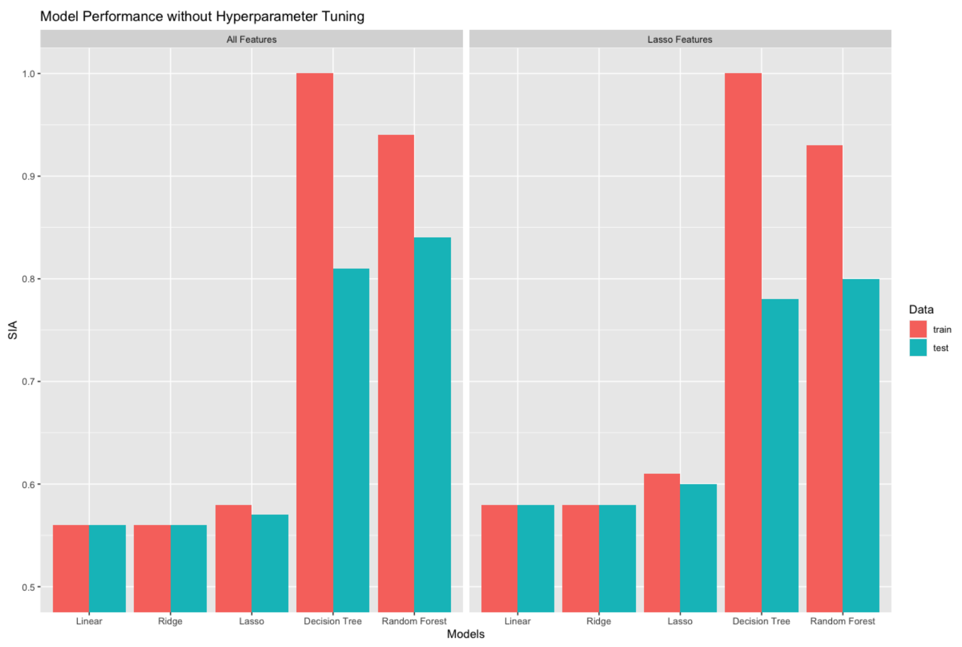

Figure 5.3: Model Performance measured by SIA using All Features and LASSO Features without Hyperparameter Tuning

A closer look at Figure 5.3 indicates the increase in the performance of Linear Models compared to the Tree-based methods where SIA of LASSO surpasses 60% in both train and test data. The opposite is true in the case of Tree-based models where the performance of Random Forest slightly decreases by about 5% in the test data.

5.2 Models with Hyperparameter tuning

model	**train**	**validation**	**test**
random_forest	162	411	462
decision_tree	277	483	580
ridge	603	576	611
LASSO	606	579	613
linear	610	583	616

Table 5.2: Comparing MAE of models in train, validation and test sets

Table 5.2 suggests that the difference between the train, validation and test data set is not so significant, implying overfitting is not a problem in the cases of Linear Models. In the Tree-based methods, hyper-parameter tuning has solved the Decision Tree issue of overfitting with MAE in the train, validation and test data at 277, 483 and 580 respectively. Additionally, although decreasing the MAE of the test set due to the loss of information resulting from dimensionality reduction, Random Forest is still the best model with a minimum MAE of 462.

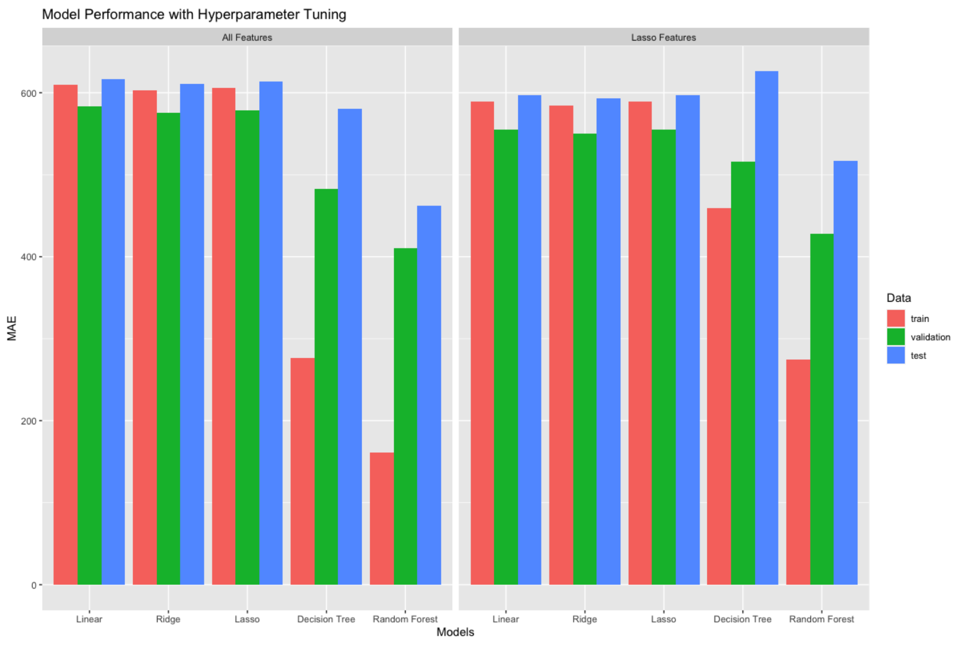

Figure 5.4: Model Performance measured by MAE using All Features and LASSO Features with Hyperparameter Tuning

Visualizing the table by bar plot, it is quite clear to see that while Linear Models slightly improved by reducing the features, the other methods show significant changes, the most noticeable of which is the small gap between MAE and SIA in the train, validation and test data.

Lastly, SIA could be considered a good measure when it comes to visualizing the models. The action of reducing the features truly improved the Linear Models in both efficiency and performance but the number is still greatly outnumbered by that of the results of Decision Tree and Random Forest.

model	**train**	**validation**	**test**
random_forest	0.94	0.84	0.84
decision_tree	0.91	0.82	0.81
ridge	0.58	0.57	0.58
LASSO	0.58	0.57	0.57
linear	0.56	0.55	0.56

Table 5.3: Comparing SIA of models in train, validation and test sets

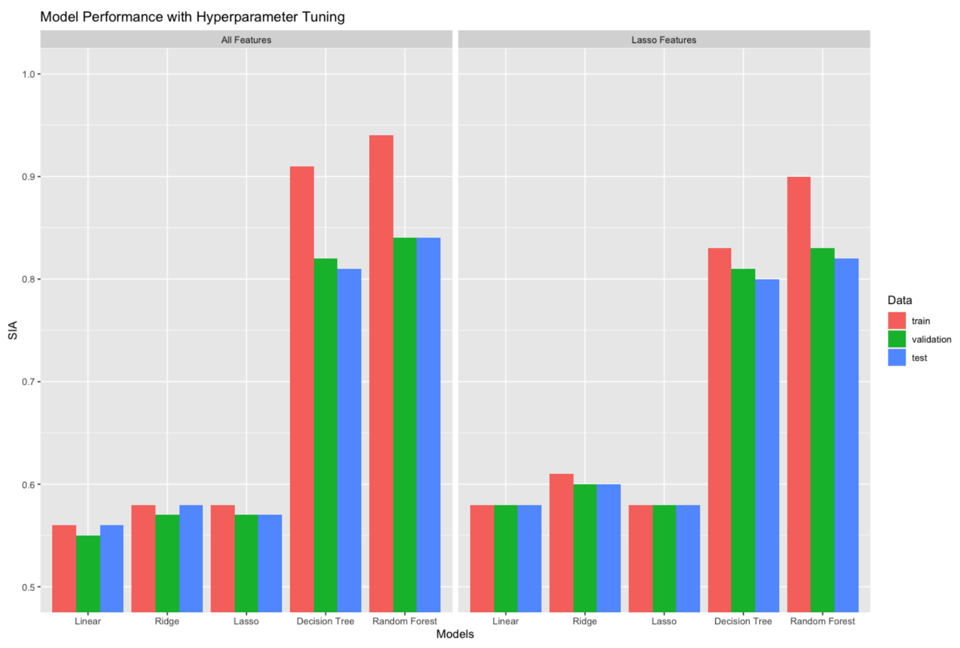

Figure 5.5: Model Performance measured by SIA using All Features and LASSO Features with Hyperparameter Tuning

Chapter 6

DISCUSSION

The amount data needed in order to have a good chance of getting significant results requires power analysis, which requires all the above information plus a measure of effect size, but the validity of the analysis depends much more on the quality of the data than on the quantity. The investigated dataset is good in terms of legitimacy since it is uploaded in Shoppe, so it could be rather accurate. Another reason why this is high-quality data is that it certainly does not have missing values since the data on the website is tidied with every product element. We also believe that it is not worth answering how much data is enough because all the businesses are different and all the data, there can be measured in different ways.
Although acknowledged the scatter plot describing the correlation of the dependent variables, we do not actually use any hypothesis testing to significantly reject any violation of Classical Linear Regression. In fact, the author does not support this classical model but is in favor of the modern Tree-based Methods. Therefore, the resulting MAE and SIA metrics have been seen as the key factor to show that Decision Tree and Random Forest have outperformed these classical models of Linear Regression, Ridge Regression and LASSO.
The data could be further analyzed using XG Boost. Random Forest is a bagging technique that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. XGBoost is a good option for unbalanced datasets when we cannot trust random forests in these types of cases. The classes will be almost certainly imbalanced where the product sold in Ho Chi Minh and Ha Noi will be huge when compared with unauthentic transactions. In XGBoost, when the model fails to predict the anomaly for the first time, it gives more preferences and weightage to it in the upcoming iterations thereby

increasing its ability to predict the class.
This thesis strongly measures the Mean Absolute Error but does not mention Mean Squared Error. One of the top reasons for this is the outliers have already been investigated with another metric Soft Interval Accuracy. With these metrics, we are interested in the amount of data fall with thin the absolute difference of a predicted observation compared to that of the data. We could go further to apply the same working pattern to see how the numbers behave in the case of Mean Squared Error and the corresponding Soft Interval Accuracy.

Chapter 7

CONCLUSION

The goal of this thesis is achieved, having investigated how to improve sales prediction accuracy through trial and error using various machine learning models and to explain why the highest accuracy models register as the best. This thesis described and analyzed five major machine learning algorithms in order to determine how effective each technique is at predicting future performance. Linear Regression, Ridge Regression, LASSO Regression Decision Trees, and Random Forest are the algorithms investigated. We investigated the accuracy of different machine learning models' sales prediction accuracy through trial and error. The prediction of total sales of shops using machine learning techniques used two approaches: least square regression and tree-based methods. Random forests produced the best results when evaluated using two metrics, Mean Absolute Errors (MAE) and Soft Interval Accuracy (SIA). We demonstrated that regression shrinkage models can produce faster results at the expense of slightly higher errors. While linear regression is a well-known statistical technique for predicting programming performance, it is limited by underlying assumptions such as normal distribution and linear relationship requirements. The data seem to not meet some of the major assumptions so subsequent work was triggered, resulting in interpretation issues and other difficulties.

Because there are numerous distinct machine learning approaches to use when building a sales prediction model, the authors may have included additional strategies in the research. The thesis work could also look into a larger dataset that allows the model to be fully generalized to all cities in Vietnam.

One expectation is that forecasting inaccuracies are not always due to the forecasting technique, but can also be the result of an unorganized forecasting process and inefficient information data. Another expected outcome is that it is critical not only to review the machine learning models within the data, but also to change the model hyperparameters rather than the default values so that data scientists can improve the accuracy of a forecast.This research lays the groundwork for future research on the application of artificial intelligence techniques to this problem and promotes computer science.

Appendix

```
import numpy as np

import tensorflow as tf

import sklearn

import pandas as pd

import os

from sklearn.preprocessing import MinMaxScaler, StandardScaler

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression, Ridge, LASSO

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import RandomizedSearchCV

import warnings

warnings.filterwarnings("ignore")

normalizer = StandardScaler()
```

```
item_df = pd.read_csv("https://raw.githubusercontent.com/cliex159/
                                   Shopee/main/items.csv",
                      encoding = 'utf-8', delimiter = "\t")
item_df.info()

search_df = pd.read_csv("https://raw.githubusercontent.com/cliex159/
                                   Shopee/main/search_15k6.csv",
                        encoding = 'utf-8', delimiter = "\t")
search_df.info()

# Convert shop_rate_feedback and shop_time_feedback's None values to
                                   zeros
item_df[['shop_rate_feedback']] = item_df[['shop_rate_feedback']].
                                   replace("None", 0).astype(int)
item_df[['shop_time_feedback']] = item_df[['shop_time_feedback']].
                                   replace("None", 0).astype(int)


full_df = search_df.merge(item_df, how = 'inner', on = ['id'],
                          suffixes = (None, '_2'))
full_df

y = full_df['n_sold'].to_numpy() #get the label
```

```
y

category = pd.get_dummies(full_df['category'], drop_first=True) #
                                   Convert category to one hot
                                   encoding

price = full_df[['price']]

shop_address = pd.get_dummies(full_df['shop_address'], drop_first=True
                                   )

avg_rating = full_df[['avg_rating']]

n_reviews = full_df[['n_reviews']]

n_loved = full_df[['n_loved']]

shop_n_review = full_df[['shop_n_review']]

shop_n_product = full_df[['shop_n_product']]

shop_rate_feedback = full_df[['shop_rate_feedback']]

shop_time_feedback = full_df[['shop_time_feedback']]

shop_age = full_df[['shop_age']]

shop_follower = full_df[['shop_follower']]


feature_list = [shop_address,

                category,
                price,

                avg_rating,

                n_reviews,

                n_loved,

                shop_n_review,

                shop_n_product,

                shop_rate_feedback,

                shop_time_feedback,

                shop_age,

                shop_follower]


class Metrics():



    @staticmethod

    def computeMAE(y_true, y_pred):

        return np.round(np.mean(np.abs(y_true-y_pred)),2)



    @staticmethod
    def computeSIA(y_true, y_pred, eps = 500): # soft interval
                                                     accuracy

        error = np.abs(y_true - y_pred)

        res = np.round(np.mean(((error - eps) < 0 ) & (y_pred >= 0)),2
                                                        )

        return res


def run_model(y, model_name, feature_list, tuning = False):

  X = np.hstack(feature_list)

  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
                                                   = 0.2,

                                                                random_state = 1



  X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,

                                                              test_size = 0.2,

                                                              random_state = 1)




  if model_name != 'tree' and model_name != 'forest':

    for i in [X_train, X_val, X_test]:

      i = normalizer.fit_transform(i)
```

```
if model_name == 'linear':
  model = LinearRegression()
  model.fit(X_train, y_train)
elif tuning == False:
  if model_name == 'ridge':
    model = Ridge()
  elif model_name == 'LASSO':
    model = LASSO()
  elif model_name == 'tree':
    model = DecisionTreeRegressor()
  else:
    model = RandomForestRegressor()

  model.fit(X_train, y_train)
else:
  alpha = np.linspace(0.001, 100, num = 100)
  n_estimators = [int(x) for x in np.linspace(10, 50, 10)]
```

```
    max_features = ['auto', 'sqrt']

    max_depth = n_estimators

    max_depth.append(None)

    min_samples_split = [2, 5, 10]

    min_samples_leaf = [1, 2, 4]

    if model_name == 'ridge':

      random_grid = {'alpha': alpha}

      estimator = Ridge()

    elif model_name == 'LASSO':

      random_grid = {'alpha': alpha}

      estimator = LASSO()

    elif model_name == 'tree':

      random_grid = {'max_depth': max_depth,

                     'min_samples_split': min_samples_split,

                     'min_samples_leaf': min_samples_leaf}

      estimator = DecisionTreeRegressor()

    else:

      random_grid = {'n_estimators': n_estimators,
                     'max_features': max_features,
                     'max_depth': max_depth,
                     'min_samples_split': min_samples_split,
                     'min_samples_leaf': min_samples_leaf}
      estimator = RandomForestRegressor()
    search_cv = RandomizedSearchCV(estimator = estimator,
                                   param_distributions = random_grid,
                                   n_iter = 100, n_jobs = -1)
    search_cv.fit(X_train, y_train)
    model = search_cv.best_estimator_

  mae_train = Metrics.computeMAE(y_train, model.predict(X_train))
  mae_val = Metrics.computeMAE(y_val, model.predict(X_val))
  mae_test = Metrics.computeMAE(y_test, model.predict(X_test))
  print('MAE:', mae_train, mae_val, mae_test)

  for i in [200, 500, 1000]:
    SIA_train = Metrics.computeSIA(y_train, model.predict(X_train), i)
    SIA_val = Metrics.computeSIA(y_val, model.predict(X_val), i)

    SIA_test = Metrics.computeSIA(y_test, model.predict(X_test), i)

    print('SIA', i, end = '')

    print(':', SIA_train, SIA_val, SIA_test)



  if model_name == 'tree' or model_name == 'forest':

    return [model, None]

  return [model, model.coef_]


def tuning_run(tuning = False):

  print('All Features:\n')

  print('\nDecision Tree:\n')

  run_model(y, 'tree', feature_list, tuning = tuning)

  print('\nRandom Forest:\n')

  run_model(y, 'forest', feature_list, tuning = tuning)

  print('\nLinear Regression:\n')

  run_model(y, 'linear', feature_list, tuning = tuning)

  print('\nRidge Regression:\n')

  run_model(y, 'ridge', feature_list, tuning = tuning)
```

```
print('\nLASSO Regression:\n')

LASSO_result = run_model(y, 'LASSO', feature_list, tuning = tuning)

indices = np.nonzero(np.around(LASSO_result[1]))[0]

col = []

for i in range(0, len(feature_list)):

  col.extend(list(feature_list[i].columns))

reduced_features = [col[i] for i in indices]

reduced_shop_address = shop_address[shop_address.columns.
                                    intersection(reduced_features)]

reduced_category = category[category.columns.intersection(
                                    reduced_features)]

reduced_feature_list = [reduced_shop_address, reduced_category,
                                    n_reviews,

                        shop_rate_feedback]

print('\n\nReduced Features:\n')

print('\nDecision Tree:\n')

run_model(y, 'tree', reduced_feature_list, tuning = tuning)

print('\nRandom Forest:\n')

run_model(y, 'forest', reduced_feature_list, tuning = tuning)
```

```
  print('\nLinear Regression:\n')

  run_model(y, 'linear', reduced_feature_list, tuning = tuning)

  print('\nRidge Regression:\n')

  run_model(y, 'ridge', reduced_feature_list, tuning = tuning)

  print('\nLASSO Regression:\n')

  run_model(y, 'LASSO', reduced_feature_list, tuning = tuning)


tuning_run()


tuning_run(tuning = True)


linear_result = run_model(y, 'linear', feature_list)


random_forest_result = run_model(y, 'forest', feature_list)


ridge_result = run_model(y, 'ridge', feature_list)


LASSO_result = run_model(y, 'LASSO', feature_list)


indices = np.nonzero(np.around(LASSO_result[1]))[0]

indices
```

```
col = []

for i in range(0, len(feature_list)):

  col.extend(list(feature_list[i].columns))

reduced_features = [col[i] for i in indices]

reduced_features



reduced_shop_address = shop_address[shop_address.columns.intersection(
                                   reduced_features)]

reduced_category = category[category.columns.intersection(
                                   reduced_features)]

reduced_feature_list = [reduced_shop_address, reduced_category,
                                   n_reviews,

                        shop_rate_feedback]



linear_result = run_model(y, 'linear', reduced_feature_list)



random_forest_result = run_model(y, 'forest', reduced_feature_list)



ridge_result = run_model(y, 'ridge', reduced_feature_list)



LASSO_result = run_model(y, 'LASSO', reduced_feature_list)
```

```
random_forest_result = run_model(y, 'forest', feature_list, tuning =
                                                True)
```

Bibliography

[1] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R.* Springer, 2013. [Online]. Available: https://faculty.marshall.usc.edu/gareth-james/ISL/

[2] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *Journal of the Royal Statistical Society Series B*, vol. 68, no. 1, pp. 49–67, February 2006. [Online]. Available: https://ideas.repec.org/a/bla/jorssb/v68y2006i1p49-67.html

[3] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996. [Online]. Available: http://www.jstor.org/stable/2346178

[4] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *Journal of the Royal Statistical Society Series B*, vol. 68, pp. 49–67, 02 2006.

[5] L. Wendlinger, E. Berndl, and M. Granitzer, "Methods for automatic machine-learning workflow analysis," in *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track*, Y. Dong, N. Kourtellis, B. Hammer, and J. A. Lozano, Eds. Cham: Springer International Publishing, 2021, pp. 52–67.

[6] A.-M. Legendre, *Nouvelles methodes pour la determination des orbites des cometes.* Paris: F. Didot, 1805.

[7] C. F. Gauss, *Theoria motus corporum coelestium in sectionibus conicis solem ambientium.* Hamburg: F. Perthes & I. H. Besser, 1809.

[8] C. Brooks, *Introductory Econometrics for Finance*, 4th ed. Cambridge University Press, 2019.

[9] T. Amemiya, *Advanced Econometrics.* Harvard University Press, 1985.

[10] T. E. Dielman, “A comparison of forecasts from least absolute value and least squares regression,” *Journal of Forecasting*, vol. 5, no. 3, pp. 189–195, 1986. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/for.3980050305

[11] S. Makridakis, S. Wheelright, and R. Hyndman, *Manual of Forecasting: Methods and Applications*, 01 2000.