Master Thesis

M.Sc. Business Administration

Hamburg University
Faculty for Business Administration
Chair for Statistics
Prof. Dr. Martin Spindler

DEMAND ESTIMATION WITH MACHINE LEARNING

Examiner: Prof. Dr. Martin Spindler

Abstract

This paper describes a comparison of classical and machine learning methods for the prediction of demand of breakfast cereals. A literature review on predicting demand with discrete choice models and machine learning methods is provided. A utility based linear demand model serves as theoretical basis. The applied methods are explained, then data is simulated to allow for the assignment of coefficients and variation of features. Data pre-processing is done before fitting the models. Results show that machine learning methods outperform classical methods, when working with large number of features.

submitted by:

Georgi Z. Zhelev

Matriculation #: 7087423

Krohnskamp 8

22301 Hamburg, Germany

+49 157 70455910

georg.zhelev@gmail.com

supervised from:

Jannis M. Kueck

Submisson Date, Place:

14.08.2020 Hamburg, Germany

Contents

List of Figures

List of Tables

List of Abbreviations and Definitions

ML methods	machine learning methods
standard econometric, or classical methods	non-machine learning methods i.e. linear regression and logit
BLP Model	Berry, Levinson & Pakes Mixed Logit Model
CV	Cross Validation
SNR	Signal-to-Noise Ratio
outcome, alternative, choice & class	are used interchangeably to refer to the alternative selected by the decision maker
consumer, decision maker & observation	are used interchangeably and refer to the decision maker
features, predictors & regressors	are all used to refer to the X-variables of the model
response & y-variable	are used to refer to the Y-variable of the model

List of Symbols

U	utility or net benefit
ϵ	error term or unobserved factors
β	coefficient of x
x	undefined feature of an alternative, also referred to as the x variable
n	index for an observation in the data set
p	index for a product characteristic
j	index for an alternative
V	representative utility or observed factors

List of Equations

1 Introduction

To enable technological services such as financial market transactions, mobile telephone calls and the storage of medical records, data must be recorded stored and analyzed [29]. Because there is so much data which is generated and stored by corporations every day, not all of it is analyzed. Corporations, for example are interested in data in order to adapt their products and services to better meet customer needs, optimize operations and infrastructure and find new sources of revenue. Machine Learning (ML) is an area of new and innovative methods to analyze big amounts of data [29].

ML is particularly well suited to applications in retail, health care, energy, finance or for web based businesses, where large amounts of data are available to help make better decisions and understand customer behavior [12]. Large data sets also allow for predicting more complex relationships. The availability of large data sets has raised interest in modeling consumer behavior and predicting demand for products with the goal of expanding revenue streams [1].

1.1 Demand Estimation with Machine Learning

One specific area of modeling consumer behavior is demand estimation. Business are interested in knowing what the demand for their product be change in the future. Demand estimation is the means to predict how consumers will react to price changes of the product, a change in their income or other factors [38]. Using empirical data, can the quantity demanded or the market share of a specific product be predicted in the future. Therefore demand prediction helps managers make better business decisions. The advantage of machine learning to standard economic models is the ability to estimate demand with large numbers of observations and features, or otherwise the so called *big data* [29].

However big data is difficult to estimate with traditional statistical methods i.e. regression or logistic regression. The reason is the high dimensionality of big data. For example, scanner panel data from supermarkets with product and store level fixed effects could have many hundreds of explanatory variables. A standard regression problem, with so many variables, would cause the parameters to be poorly estimated [1]. The traditional methods (linear and logistic regression) work well when only the most important variables are used for the prediction. To find the most important variables, variable selection is used. Variable selection using the classic models is done by the practitioner himself, and is based on deep industry knowledge, which could be very difficult to acquire, for example when the practitioner isn't an industry expert. Sometimes the selection criteria for variables is too arbitrary and often a subject of discussion, between industry experts.

For this reason a more abstract mechanism is sought to help with variable selection. Machine learning techniques offer this possibility and are often applied to demand estimation [1]. Examples of these are LASSO, decision tree, random forest and gradient boosting. The advantages of these methods are that they are easy to use, provide a better goodness of fit, are scalable to large amounts of data and applicable in open source software such as R and Python [1].

1.2 Goal of Research

This paper describes the empirical application of classical and machine learning methods in the open source software R [28] for the purpose of predicting demand. Product data, such as prices and product characteristics for breakfast cereals are simulated. The response variable is of categorical nature and represents the chosen alternative. The applied classical methods are logit and linear regression, while the applied ML methods are LASSO, a decision tree, random forest and gradient boosting. The methods were adapted to the data and their performance was measured using confusion tables and accuracy. The application focuses mainly on comparing the prediction accuracy of classical and ML methods under the variation of the number of features. When the method allows, estimated coefficients are compared to the assigned coefficients. Model interpretability measures for the tree-based methods such as variable importance are shown. Pre-processing of the simulated data is done for the fitting of the logit, linear regression and LASSO models.

1.3 Overview

The rest of this paper is organized as follows: Section 2 offers a literature review of demand estimation with discrete choice models and with machine learning methods. Section 3 offers the theoretical framework. It begins with the microeconomics of predicting demand. Then presents the utility based demand model, which serves as the theoretical basis for the data simulation. Further the maximum likelihood estimation procedure is explained. Lastly the section summarizes the applied in this paper classical and machine learning methods. Section 4 describes the data simulation and the fitting and tuning of the models. Section 5 presents the results. Section 6 discusses the results and their validity. Section 8 provides a short outlook of open problems, questions or ideas about further investigations. Appendix A displays the used code. An executable version of the code is provided with a data storage device.

2 Literature Review

2.1 Demand Estimation with Discrete Choice Models

Nevo (2000) [24] explains the early hurtles to demand estimation that give insight into the development of the discrete choice model literature. The first attempts of estimating demand involved specifying a system of demand equations for each product. The demand of that product was a function of its own price, the price of other products and other variables. Because there are many and differentiated products in the market, that resulted in a large number of parameters to be estimated. Further, idiosyncratic consumer tastes important for the estimation are not observed and therefore not included in the model. To solve this issue it was simply assumed that an average consumer exists, which made the model extremely restrictive. The logit demand model from McFadden (1974) [22] solved the dimentionality problem and was easy to estimate, however made strong assumptions such as the irrelevant alternatives property, that caused relative probabilities of existing alternatives to be unchanged by the introduction of new ones [9]. Extensions such as the nested logit model, allowed for more flexibility by allowing for preference correlation only inside the nests [9]. The most flexible model in terms of preference correlation was the BLP model developed by Berry Levinson and Pakes (1995) [2]. In this model the substitution among products is a function of the consumer characteristics instead of nesting specification [9].

A good summary of the BLP model is provided by Train [36]. The model focuses on the estimation of consumer demand based on market level data. Individual-level demand is aggregated over consumers to obtain market-level demand functions. Market equilibrium prices are determined from the interaction between demand and supply functions, therefore both need to be modeled. The observed demand is made up of continuous variables (market shares) rather than discrete variables. Because demand is aggregated on a market level, prices are endogenously determined by the interaction of demand and supply. In contrast prices on an individual level are not affected by an individual consumers choice. To account for the endogeneity of prices in this aggregated model, instruments are necessary. But because the data is aggregated, unobserved factors enter the demand function non-linearly, which complicates the use of instrumental variables. According to Train the contribution of Berry Levinson and Pakes is the transformation of the market share data to allow for the unobserved factors to enter the demand function linearly (p. 9).

Nevo (2001) [25] applied the BLP model to breakfast cereals. The dependent variable was given by observed market share of the product and explanatory variables were given by product characteristics such as price, sugar content and mushiness. Nevo considered also demographic characteristics of the cereal buyers, such as income, age and if a child is present in the family. Finally

Nevo also applied a set of instruments to account for the endogeneity of prices, which results due to aggregation. The motivation for using breakfast cereals for the application in this paper comes from Nevo, however in the case of this paper, a basic logit model instead of a BLP model is used.

2.2 Demand Estimation with Machine Learning

This subsection describes relevant to the topic of demand estimation literature, in which classical and machine learning (ML) methods were compared. The subsection will conclude with specifying, which of the presented literature papers this paper is most similar to.

Bajari et al. (2015) [1] compared six machine learning methods to two classical methods. The motivation was to help econometricians find practical tools to estimate demand with large numbers of observations and covariates. The authors used a scanner panel on salty snacks from one grocery store chain for six years. They allowed for product and store level fixed effects, which effectively multiplied the number of explanatory variables, and made standard regression unusable. They found that the six machine learning models predicted demand out-of-sample much more accurately than a panel data or a logistic model. A further step they took was the combination of the prediction methods into an ensemble model and so increasing the prediction accuracy further.

Paredes et al. (2017) [27] investigated how machine learning methods can outperform discrete choice models for prediction of car ownership. The authors used transportation household survey data from Singapore. They find that when using discrete choice features, the machine learning methods are inferior to a logit model. However after engineering these features to be more appropriate for machine learning, they are superior. According to the authors the findings highlighted both the costs of applying machine learning models in economic contexts and the opportunities for improved prediction.

Chernozhukov et al. (2018) [4] mention the interest of econometricians to use machine learning methods originally designed for prediction to help with the model selection process, while retaining desirable inference properties for causal parameters. Because estimation of counter-factual outcomes is a key aspect of causal analysis, but often there is an absence of explicit exogenous variation, economists must do the model selection themselves, which is subjective and labor intensive. Hence the authors reiterate the use of machine learning (ML) methods to automate the model selection process. This is a case where machine learning methods could be applied beyond just prediction.

In addition to discussing the benefits for economists of applying ML methods, Green and Richards (2016) [12] compare several ML methods to standard models of demand estimation, using a scanner panel data on cold cereals for a set of grocery stores. The ML methods they use to reduce dimensionality are stepwise regression, stagewise regression, LASSO, support vector machines, bagging and random forests. The authors apply an approach to simulate coefficients in order to interpret the results of the ML methods, but because these methods don't have standard normal asymptotics, they leave the interpretations open to criticism. They show that ML methods can predict better than standard econometric methods.

Varian (2014) [37] compares ML methods to standard econometric methods and shows that depending on the underlying relationship in the data, ML or standard methods could perform better. For example when the underlying relationship is linear, a normal regression performs better, but when the underlying relationship is non-linear, a regression tree performs better. He gives also examples such as the Titanic data, where a tree model doesn't necessarily perform better, but helps reveal aspects of the data that are not apparent from the traditional linear approach. Similar to the Chernozhukov paper, Varian uses ML methods beyond their original purpose for prediction.

2.3 Justification of Research

The application described in this paper resembles the work of Paredes et al. [27] the most, due to the presence of a categorical response variable and hence the same performance measures: confusion matrix and accuracy. Paredes et al. predict if a car is sold or not, while this paper determines if a breakfast cereal is chosen. Paredes et al. perform feature engineering to adopt features to the prediction methods. Similarly this paper transforms the original data for the purpose of fitting the logit, linear regression and LASSO models. The second most simular paper is the one from Bajari et al. [1]. In contrast to this paper, Bajari et al. work with quantity sold of salty snacks, a categorical variable, which enables them to observe mean squared error (MSE) as performance measure. Both Bajari et al. and Paredes et al. use real world data, while data in this paper is simulated with the purpose of assigning *true* coefficients and measuring deviations to estimated coefficients. A second reason for simulating the data is that splitting of a real world data set into training and testing samples could without the application of any advanced sampling methods risk making the samples imbalanced.

3 Theoretical Framework

3.1 Microeconomics of Demand

This subsection will look into the economics theory and motivate the estimation demand equations. It draws from the textbook *Managerial Economics* by Hill (1989) [15] (p. 100-125). Related examples to breakfast cereals are made, since these products are the focus of the empirical work in this paper.

Production and Pricing Decisions

According to Hill, the success of a business organization depends on it producing goods and services that consumers want. For example in the breakfast cereal market there are goods with different characteristics. Some consumers prefer their cereal crisp, others like it sweet, while others prefer their cereal be made up of whole grains and have less added sugar. Offering multiple different products to satisfy diverse consumer preferences will further the profitability of a firm, because it is catering to a large group of consumers (p. 100).

Goods and services can be sold successfully when a firm offers them at a price consumers are prepared to pay. For example, if a firm offers a breakfast cereal for a price higher than most parents are ready to pay, the cereal will not sell well even if it hast the right combination of sweetness, crispiness or enticing packaging. Or if that same cereal is sold for a very cheap price, then many consumers will buy the product, however the firm may not profit from the extra sales. In this case the firm failed to determine what the highest price is that the consumer is ready to pay for the cereal and forwent measurable profits (p. 100). Therefore it is important for a firm to sell its goods or services at the price consumers are prepared to pay.

A Firms Demand Condition

To find the price consumers are prepared to pay, knowing the firms demand condition is essential. Price and product attributes have influences on the demand of a product. The principal tool to analysing demand is the demand function

$$Da = f(X_1...X_n),$$

where Da is the demand for a good a, f is a general (unspecified) function, and $X_1...X_n$ are the variables that influence demand (p. 101). To determine the demand for product a, the next step would be to identify the particular variables that influence demand. These may be as above mentioned price, quality, popularity, sweetness, packaging etc. After identifying the variables, the next step is to transform the general function f into a specific one.

In this project f will take the form of a logit probability, then a linear function in the case of regression and LASSO, but also a not-linear function in the case of the tree-based methods. When the function is specified, it will be then determined to what extent a variable affects demand. For example, in the marketing world of cereals, the sweetness of a cereal and the enticement of its packaging play an important role. As seen in any grocery store, cereals have their own shapes, colors and even mascots, which serve to differentiate them from the competition.

Variables that influence Demand

To what extent an enticing packaging has on the demand of a cereal will be determined by the estimated coefficient (β) for that variable. Given the demand function, the influence of a particular variable can be measured by the partial derivative

$$\frac{dD}{dXi},$$

where dD is the change in demand for a specific product and dX_i is the change in a specific product attribute. A change in a product attribute in the denominator will cause a corresponding change of the demand in the numerator (p. 101). If the partial derivative is positive, an increase of Xi will cause a increase in demand and if negative, a decrease in demand. If for example the product attribute is price, which tends to be the most important product characteristic for the average consumer, plotting the relationship between quantity demanded and price, yields the graphic in Figure 1.

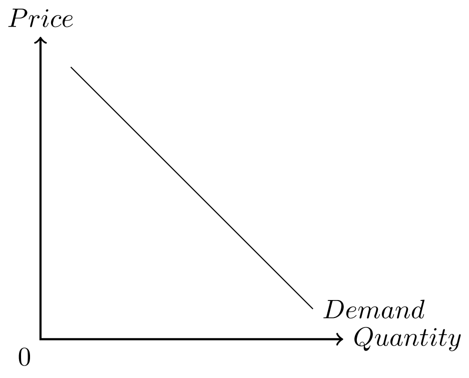


Figure 1: Demand Curve

Figure 1 shows an inverse relationship between quantity demanded and price. This means that the estimated coefficient of price will always be negative and should be modeled as such when simulating data. Other variables such as quality or popularity of the product also affect its demand.

In this project, the demand for cereals, given different simulated characteristics, will be analyzed.

The type of product, also has an influence on its demand. Cereals are consumer and non-durable goods, which means their purchase is repetitive. For cereals the brand is probably their most important aspect next to price. People may buy the same type of cereal over and over because they become loyal to the brand. According to Foeller [9], this effect has an influence on the elasticity of a product and can cause consumers to stick with products despite alternatives being available at a lower price.

Elasticity is the marginal effect on demand of a unit change in one variable and it is measured by the partial derivative as mentioned above. The most common example is the price elasticity of demand, which measures the responsiveness of demand to a change in price. According to Hill there is a unique relationship between price elasticity and the effect of price changes upon the total revenue. This relationship is summarized in Table 1.

Table 1: Price Elasticity and Total Revenue

		Price	
		Decrease	*Increase*
Demand is price	*Elastic*	Total revenue increases	Total revenue decreases
	Inelastic	Total revenue decreases	Total revenue increases

adapted from Hill [15] (p. 111)

Table 1 shows that raising the price will only be profitable under certain conditions, for example when the good is inelastic. When a certain product has not many substitutes, such as gasoline, and consumers depend on it for their transportation, then its price is inelastic. Even if prices of gasoline rise, people will still need to buy it. Similarly when a customer is loyal to a cereal brand, they are likely to buy it even if the price has increased. Price changes are not simulated in the empirical application in this paper, therefore elasticities are not analyzed, however they are an important part of demand estimation in the real world.

The Identification Problem

Often when predicting demand it is difficult to differentiate the demand from the supply curve. Sales of a product, rather than demand are observed at different prices. Economic theory states that sales & prices are determined by the interaction of supply & demand. According to Hill, because of this interaction, it is difficult to determine if the estimated equation is the demand or the supply curve, or more likely a combination of both, such as in Figure 2.

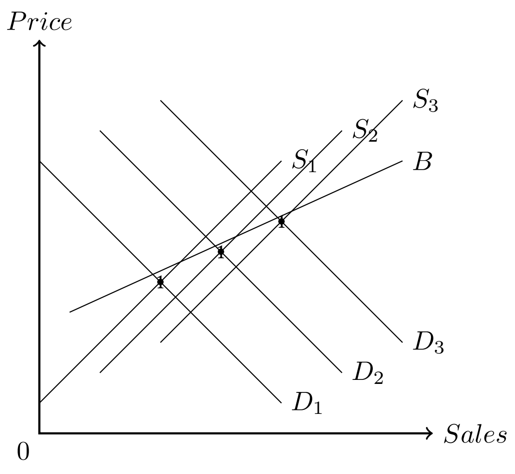


Figure 2: Mongrel Curve
adapted from Hill [15] (p. 125)

This project will only focus on the estimation of demand curves that tend to have price and product characteristics as variables. In this case, the demand equation looks like

$$Demand = \beta_1 * price + \beta_2 * characteristic.$$

Supply will not be taken into account, which tends to have variables such as price and cost of raw materials and looks like

$$Supply = \beta_1 * price + \beta_2 * raw\ material\ costs.$$

According to Nevo (2002) [24], interaction between supply and demand leads to an endogeneity problem in the estimation of coefficients. Demand, which is a response variable on the left side of the equation tends to be influenced by the same factors, which influence price, a variable on the right side of the equation.

Having covered the most important aspects of the microeconomics of predicting demand, the next sub-sections are as follows: first the demand model, which determines the data generating process is presented, then the logit model and its estimation are described. Lastly, the theoretical foundation of the remaining classical and machine learning methods is summarized.

3.2 Demand Model

This section sourced from Train (2002) [36] (p. 1-59), describes the utility based demand model originally proposed by McFadden (1974) [22]. It begins with the basics of a discrete choice model and the utility framework behind it. Then, the section elaborates on the logit model and its limitations. Finally it describes the maximum likelihood estimation procedure for the coefficients. When relevant, references are made to the data simulation, which is described in its entirety in Section 4.

According to Train, in order to derive a model which would be appropriate for the analysis of demand, one has to begin at the individual choice level . A customer faces a buying decision among a set of competing alternatives. The outcome of a decision is denoted as y. The outcome variable is discrete. The goal is to understand the behavioral process which leads to the choice of the consumer. It is reasonable to assume that the buying decision in a market is of a discrete nature. The consumer will choose to buy one product or no product. In the case that the consumer buys two products, the discrete choice assumption still holds, because only one product at a time can be consumed (p. 3).

In order to fit a discrete choice model, the choice probabilities need to have certain characteristics. They have to be mutually exclusive, exhaustive and finite. Mutually exclusive means that choosing one alternative implies not choosing another. Exhaustive means that all possible alternatives are included in the choice set. An option for choosing none of them could also included. Finite means that the number of possible outcomes is not infinite. These features differentiate discrete choice models from continuous models (p. 16).

Decision analysis is classically done through utility, which means that a person obtains a net benefit from choosing a product. It is assumed that the decision maker maximizes utility. The factors, which contribute to utility are the observed x and unobserved ϵ. They flow into utility in the following manner:

$$U_{nj} = \beta * x_{nj} + \epsilon_{nj};\ \forall j, \tag{3.1}$$

where U is the net utility and β is a coefficient of how observed factors x contribute to U. The index n is for a decision-maker that faces a choice between j alternatives. The different alternatives, j, are labeled with the following sequence: $j = (1, ..., J)$. The ϵ term captures the factors that are included in U, but are not part of the model. The decision maker will choose the alternative that provides the highest utility. For example, when $U_{ni} > U_{nl}$ the choice is alternative i (p. 19).

The researcher observes the attributes of the product x_{nj}. They are labeled with a subscript n, because the attributes depend on the perception of the decision maker. Representative utility is only made up of the observable factors x as

$$V_{nj} = \beta' * x_{nj}. \tag{3.2}$$

Therefore representative utility is the deterministic part of the equation and belongs to the total utility equation in the following manner

$$U_{nj} = V_{nj} + \epsilon_{nj},$$

where the attributes which the researcher does not observe are labeled ϵ_{nj}. The unobserved attributes are product and consumer specific attributes, which the researcher can't capture. Therefore he treats them as random. For the sake of estimation, one has to transform V_{nj} which can take any real value so that it can be interpreted as a probability (p. 20).

According to Train, because the decision maker chooses one alternative, indexed i from all available alternatives indexed j, the choice could be expressed as

$$Prob(U_{ni} > U_{nj} \;\; \forall j \neq i).$$

Now substituting for the utility formula results in

$$Prob(V_{ni} + \epsilon_{ni} > V_{nj} + \epsilon_{nj} \;\; \forall j \neq i),$$

and rearranging the terms gives us

$$Prob(\epsilon_{ni} - \epsilon_{nj} < V_{ni} - V_{nj} \;\; \forall j \neq i).$$

Since the values of ϵ are unknown, a probability could be calculated. This probability is equal to the integral over all possible values of the unobserved factors or

$$\int_{\epsilon} I(\epsilon_{nj} - \epsilon_{ni} < V_{ni} - V_{nj} \;\; \forall j \neq i) f(\epsilon_n) d(\epsilon_n).$$

Since ϵ is a random vector $\epsilon_n = (\epsilon_{n1}, ..., \epsilon_{nJ})$ with a joint density $f(\epsilon_n)$, the choice cannot be predicted exactly. Instead taking the integral over the density of the unobserved portion of utility $f(\epsilon_n)$, would calculate the probability of a particular outcome (p. 20).

The Logit Model

In order to evaluate the integral, one has to make assumptions about the distribution of the error term ϵ. The logit model, is derived by assuming the distribution of the unobserved portion of utility, $f(\epsilon)$, is independently identically extreme value distributed (p. 42). The unobserved factors ϵ are assumed to be uncorrelated over alternatives and to have the same variance over all alternatives. Then ϵ has density

$$f(\epsilon)_{nj} = e^{-\epsilon_{nj}} e^{-e^{-\epsilon_{nj}}} \tag{3.3}$$

and a cumulative distribution

$$F(\epsilon)_{nj} = e^{-e^{-\epsilon_{nj}}}.$$

Because the ϵ's are independent, the cumulative distribution is the product of the individual cumulative distributions (p. 44),

$$P_{ni} \mid \epsilon_{ni} = \prod_{j \neq i} e^{-e^{(\epsilon_{ni}+V_{ni}-V_{nj})}}.$$

Because ϵ_{ni} is not given, the probability of that choice is the integral over all values of ϵ_{ni} weighted by its density (3.3):

$$P_{ni} = \int (\prod_{j \neq i} e^{-e^{(-\epsilon_{ni}+V_{ni}-V_{nj})}}) e^{-\epsilon_{ni}} e^{-e^{-\epsilon_{ni}}} d\epsilon_{ni}.$$

After manipulating the integral, the closed-form expression results in

$$P_{ni} = \frac{e^{V_{ni}}}{\sum_j e^{V_{nj}}}, \tag{3.4}$$

which is the logit choice probability. Expressed in terms of the linear parameters, the equation becomes

$$P_{ni} = \frac{e^{\beta' X_{ni}}}{\sum_j e^{\beta' X_{nj}}}.$$

Properties of Logit Choice Probabilities

The resulting logit-probabilities have the following properties. **First**, the probability P is between 0 and 1. When the representative utility of alternative i rises, reflecting an improvement in its observed attributes X_{ni}, the exponential in the numerator of (3.4) will increase. While representative utilities of other alternatives, in the denominator, remain constant, P_{ni} increases and approaches one. Same goes for a decrease in V_{ni}, which will decrease the exponential in the numerator and the probability P_{ni} will approach zero (p. 45).

Second, choice probabilities for all alternatives j have to sum to one. This means that the decision-maker has to choose one alternative. The denominator of (3.4) is the sum of the representative utilities over all alternatives (p. 45). During the data simulation, described in Section 4, the obtained logit probabilities are proofed if their probabilities sum to 1.

The **third** property of a logit choice probability is that it is related to the representative utility in an S-shaped fashion, such as shown in Figure 3. In the figure P_{ni} is the probability of choosing alternative i by decision maker n with a representative utility V.

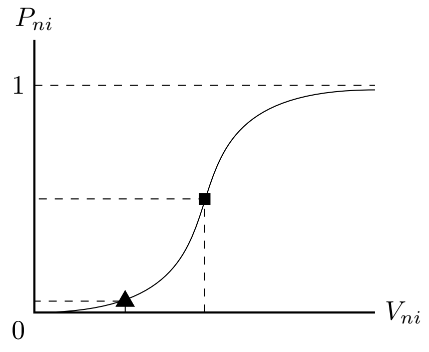


Figure 3: Binomial probability plot from simulating two choice alternatives
adapted from Train (p. 46)

The S-shape describes the impact of changes in explanatory variables onto the probability of choosing the alternative. For example, if the representative utility of an alternative is very low (triangle-point in Figure 3) compared to that of another (square-point), a small increase in the utility of that alternative has little impact on the probability of it being chosen. If an alternative has already a moderate representative utility (square-point), only a small increase in one of its attributes would increase the probability more than a larger utility increase by the triangle-point. Train makes the example that a large improvement in the bus service of an area, where the service is already poor, that a few travelers take the bus, has a much smaller impact on total bus travel, compared to a small improvement of bus service in an area where bus service is widely spread (p. 45).

Limitations of Logit

Logit has certain limitations which have to be considered when applying it. The first one is that β's or tastes can not vary with unobserved variables or randomly. Tastes have to vary systematically in order to be incorporated in logit. If tastes were to vary based on unobserved variables or purely randomly, they become part of the error term, which will no longer be independently and identically distributed (p. 50).

The second limitation of logit is the proportional substitution pattern derived from the independence of irrelevant alternatives (IIA) property. This property implies that an improvement in one alternative draws proportionately from other alternatives. If for example the probability of one alternative rises by ten percent, then the probabilities of all other alternatives muss fall by ten percent. Therefore the ratio of probabilities of two alternatives have stay constant, when an attribute of a third alternative changes (p. 54),

$$\frac{P_{ni}}{P_{nl}} = \frac{e^{V_{ni}}}{e^{V_{nl}}}.$$

Train mentions that proportionate substitution can be realistic for some situations, but in many cases more flexible patterns of substitution are to be expected.

To explain proportionate substitution, Train makes the famous blue bus, red bus example: A traveler can choose between taking a blue bus or a car to work. Assumed that the choice probabilities between the two alternatives are the same 1/2, their ratio is 1. A new alternative, a red bus, that is exactly the same as the blue bus is introduced. According to the logit model, the new probabilities would be 1/3 for each of the three alternatives. In real life it is expected that the probability of taking the car remains by 1/2 and the probability of each of the two buses drops to 1/4, since the traveler is indifferent between the two buses. The ratio of probabilities between two alternatives should actually change with the introduction of a third instead of staying constant. Therefore the logit model overestimates the probability of taking either of the buses and underestimates the probability of taking the car to work (p. 54).

According to Croissant [6], if the IIA property is violated, because of an unobserved variable, such as the inability of the model to recognize that the blue and red bus are the same, the error term will no longer be independent. Relaxing the IIA assumption leads to an extension of the logit model called the mixed logit model. It maintains the Gumbel distribution and the errors are allowed to be correlated and heteroskedastic, meaning they don't all have the same variance.

Data for this paper was simulated so that the representative utilities of the three products are roughly the same such that their choice probabilities are similar, making it appropriate for a basic logit model. Since the focus of this paper is the comparison of a discrete choice model to machine learning methods, a mixed logit model remains a point of further study. The next section will cover the estimation procedure of the logit model.

3.3 Estimation

This section draws further from Train's book (2002) [36] (p. 70-78) on discrete choice methods.

According to Train, the most basic type of estimation happens when the sample is random, all alternatives are used and the explanatory variables are exogenous, meaning that the variables entering representative utility are independent of the unobserved components. A sample of N decision makers is obtained. The logit probabilities P_{ni} are then calculated, given the parameters and the explanatory variables. The probability that one individual n made the choice he was actually observed to make is

$$\prod_i (P_{ni})^{y_{ni}},$$

where y_{ni} is a dummy variable which is equal to 1 if individual n made choice i and otherwise 0. Assuming each decision maker's choice is independent of that of other decision makers, the probability of each person in the sample choosing the alternative that he was observed to chose is simply the product of the probabilities associated with every observation

$$L(\beta) = \prod_{n=1}^{N} \prod_i (P_{ni})^{y_{ni}},$$

where β are the parameters of the model. Taking the log of both sides, leads to the log-likeyhood function

$$LL(\beta) = \sum_{n=1}^{N} \sum_i y_{ni} ln(P_{ni}) \tag{3.5}$$

and the coefficient β is the value that maximizes the function (p. 70).

Interpretation of the Log-likelihood Function

Rearranging Equation (3.5) shows how the process of estimation works. At the maximum of the likelihood function, its derivative with respect to each of the parameters is zero

$$\frac{dLL(\beta)}{d\beta} = 0.$$

When representative utility (3.2) is linear in parameters, using the log-likelihood function and the formula for the logit probabilities, Train shows that the first order condition becomes

$$\sum_n \sum_i (y_{ni} - P_{ni}) x_{ni} = 0. \tag{3.6}$$

Then rearranging both sides and dividing by N results in

$$\frac{1}{N}\sum_n\sum_i y_{ni}x_{ni} = \frac{1}{N}\sum_n\sum_i P_{ni}x_{ni}. \tag{3.7}$$

The left hand side of (3.7) is the observed average of x in the sample and the right hand side the predicted average. These two averages equal each other at the maximum likelihood estimates, β. That is: the maximum likelihood estimates β are those that make the predicted average of each explanatory variable equal to the observed average in the sample (p. 71-72).

Equation (3.6) shows that the difference between a person's actual choice, y_{ni} and the probability of that choice, P_{ni} is the modeling error. Further, the left hand side of Equation (3.6) is the sample-covariance of the residuals with the explanatory variable. The maximum likelihood estimates are therefore the values of β that make the covariance zero, that is, make the residuals uncorrelated with the explanatory variables. This condition for obtaining unbiased estimates is the same as the one that applies to linear regression.

Closed-Form Solution

The estimation procedure discussed in this section has an explicit solution. This means that the log-likelihood function is globally concave and there is a unique optimum that is the global maximum. An explicit solution is present when the parameters enter utility linearly. Then the estimation can be aided using common statistical packages (p. 4). In this paper the utility formula is designed so that the parameters enter the equation in a linear manner, therefore no numerical approximation is necessary for the estimation of the coefficients.

This section concludes the presentation of the logit model and its estimation. The next section will present the theory behind the remaining statistical methods used for the prediction.

3.4 Summary of Applied Methods

This section presents the theoretical basis for the remaining methods applied in the prediction. The sourced theory for this section comes from the lecture notes of Spindler (2018) [30] and the books from James et al. (2013) [17] and Hastie et al. (2008) [14].

Figure 4 presents a diagram of all the applied models. When referring in this paper to *classical methods*, Logit and Linear Regression are meant. When referring to *machine learning methods*, LASSO, Decision Tree, Random Forest and Boosting are intended. Linear Regression and LASSO are both *linear* methods, however the former is a classical method, while the latter a machine learning method.

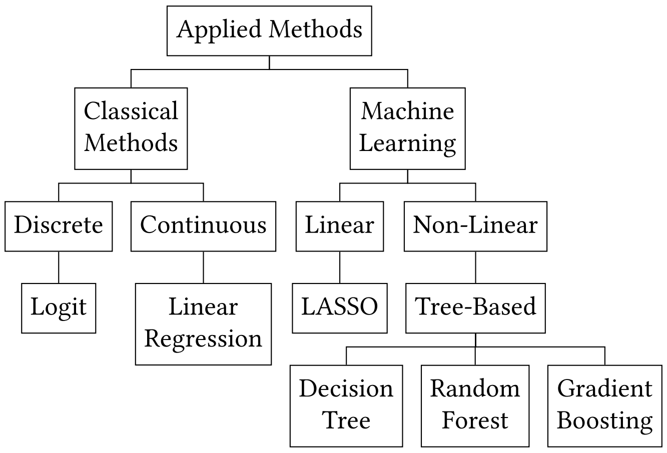


Figure 4: Applied Models

The simulated data for the prediction requires a classification, therefore it is important to keep in mind which of the presented methods is suitable and if eventually data needs to be pre-processed to enable fitting. Throughout this section and paper the assumed data generating process is that the parameters enter utility linearly according to the utility model (3.1).

Linear Regression

Adapted from Train [36] (p. 72), for a regression model $y_n = \beta' x_n + \epsilon_n$, the ordinary least squares estimates are the values of β that set $\sum_n (y_n - \beta' x_n) x_n = 0$. However instead of maximizing the likelihood function, such as in the case of logit, the estimates, β, minimize the sum of the squared residuals between the observed and predicted values, $S = min \sum_{i=1}^{n} r_i^2$. This is achieved by solving for β: $\beta = (\sum_n x_n x_n')^{-1} (\sum_n x_n y_n)$, which is the formula for the ordinary leas squares estimator. Since $y_n - \beta' x_n$ is the residual in the regression model, the estimates make the residuals uncorrelated with the explanatory variable.

Considering the applicability of linear regression to a three-level classification problem, Hastie et al. [14], explain why this method fails for classification of more than two alternatives, namely the approximated probability can be under 0 or greater than 1. Therefore classes can be masked such as in Figure 5. That means that the class made of triangle-points in the figure is not separated from the other two classes.

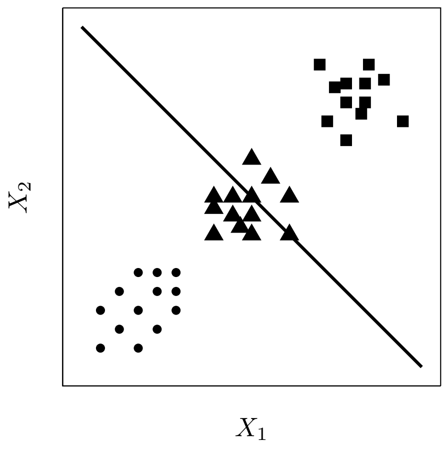


Figure 5: Linear Regression for Classification
source: Hastie et al. [14](p.105)

Further, according to James et al. [17], there is no natural way to convert a qualitative response variable with more than two levels into a continuous response variable.

A technique that allows using regression to go beyond binary classification is the *one-vs.rest* approach explained by Mueller [23] (p. 65). It essentially splits the triple-classification problem into multiple binary classification problems. Through this approach a regression is learned for each class, yielding an indicator variable 1, when the class in question is chosen or 0 when it is not. For each class is a separate regression fitted. Each fit yields its own vector of coefficients β. When making a prediction, all the regressions are run on a test point and the class of the regression that has the highest output is the predicted class. This approach was applied for both linear and LASSO regressions as they were fitted.

Estimating linear regressions when p is close to n. According to Spindler [30], a potential problem of applying linear regression is the inaccurate estimation of coefficients when the features-vector p is large and close to the size of observations n. The estimation of linear regression depends on the product-characteristics as such:

$$\sum_{j=1}^{p} = \beta_j X_j = \beta' X, \;\; for \;\; \beta = (\beta_1, ..., \beta_p)'.$$

Because there are multiple product characteristics X, to estimate each parameter β, many observations per parameter are needed. In the estimation of Bajari et al. [1], where fixed product and store effects were considered, this added many variables to the right hand side. According to Spindler [30], when the regressors p are much more than the observations n, can linear regression no longer estimate the parameters well. The goal of the empirical application in this paper is to show exactly that by increasing the number of features and comparing prediction performance. Because data has been simulated the real β's are known and deviation from the true betas can easily be determined.

LASSO Regression

As stated above, if there are many right hand variables p and not enough observations n, then the coefficients will be poorly estimated by linear regression. However if only a small amount of regressors capture most of the variation, what is known as *sparsity*, then a penalized form of linear regression,

$$\underset{\beta \in \mathbb{R}}{min} \sum_i (Y_i - \beta' X_i)^2 + \lambda * \sum_{j=1}^{p} |\beta_j|, \tag{3.8}$$

can be applied [30] (p. 24). Equation (3.8) is made up of the least squares algorithm plus a penalty term, with $\hat{\beta}$ as the solution to it. The added term penalizes the size of coefficients by their absolute values times the penalty level λ. A theoretically justified penalty level is given by

$$\lambda = \sqrt{E\epsilon^2} 2 \sqrt{2nlog(pn)}.$$

The penalty level ensures that the lasso predictor $\hat{\beta}' * X$ does not overfit the data and delivers good predictive performance under approximate sparsity. The penalization term also helps with variable selection, by choosing the variables with high coefficients. According to Spindler [30] (p. 33), having approximate sparsity, makes it possible to perform estimation with high-dimensional data. Therefore to test this quality of LASSO, sparsity was built into the simulated data.

Decision Tree

According to Bajari et al. [1] tree-based methods partition the characteristic space into a series of regions, and fit a value to each partition also known as a node. The tree uses an algorithm that determines the best variables, according to which it divides the characteristics space. For example, the decision tree fitted by James et. al. [17] (p. 304-305) shows that two variables have a particular influence on determining the salary of a baseball player. That is the number of *Hits* the player made last year and the number of *Years* the player is in the league. This partitioning is pictured in Figure 6.

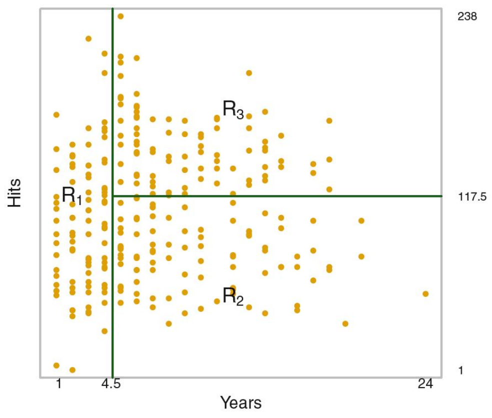


Figure 6: Partitioning of the Characteristic Space with a Tree
source: James et al. [17] (p.305)

Such a partitioning could help predict the future salary of a player. One simply needs to determine in which quadrant of the space the player falls.

Classification Tree for Prediction. Mathematically expressed, the procedure of fitting a tree starts with dividing the predictor space $X_1, X_2, ..., X_p$ into J distinct and non-overlapping regions, $R_1, R_2, ..., R_j$. This is called recursive binary splitting. For a regression tree the predicted response for an observation is given by the mean response of the training observations that belong to that region [17] (p. 307). In this paper a categorical response variable is used, therefore the prediction procedure needs to be adapted for classification. For classification, each to be predicted observation gets assigned the most commonly occurring class of training observations in the region to which it belongs (p. 312).

Similarly to growing a regression tree, recursive binary splitting is used for a classification tree [17] (p. 307). Unlike the regression tree, where selecting a predictor and a cut off point that minimize the mean squared error, classification minimizes the total variance across all classes. Therefore it divides the characteristics space so that each class is separated from other classes. One measure of this is the so called *classification error.* Classification error is the fraction of the training observations that do not belong to the most common class in a specific region and is expressed by

$$E = 1 - \underset{k}{max}(\hat{p}_{mk}).$$

where $\hat{p}_{mk}$ represents the proportion of training observations in the *m*th region that are from the *k*th class [17] (p. 312). This value is maximized and subtracted from 1, it yields the *impurity* or the proportion of training observations in the *m*th region that do not belong to the *k*th class.

The *Gini Index* is another measure of node impurity. It is defined as the total variance across the K lasses. Mathematically

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

where G takes on small values if all the $\hat{p}_{mk}$'s are close to 1 or 0. A small value indicates that the node contains predominantly observations of the same class. Cross-entropy is another splitting rule, which acts similarly to the Gini Index. The *rpart* package [34] in R uses the Gini index as the default criterion for splitting.

After applying these splitting rules the tree is grown by dividing the characteristics space more and more so an upside down tree as in Figure 7 results. The tree splits on those variables, which contribute the most to the total decrease in node impurities. In case of the baseball players the algorithm chose a subset of all available variables to split on. One can see their names and criteria in the figure. For example the first split is based on *years<4.5*. The left side includes the salary of players that played less than 4.5 years while the right side of those who played more than 4.5 years, and so on.

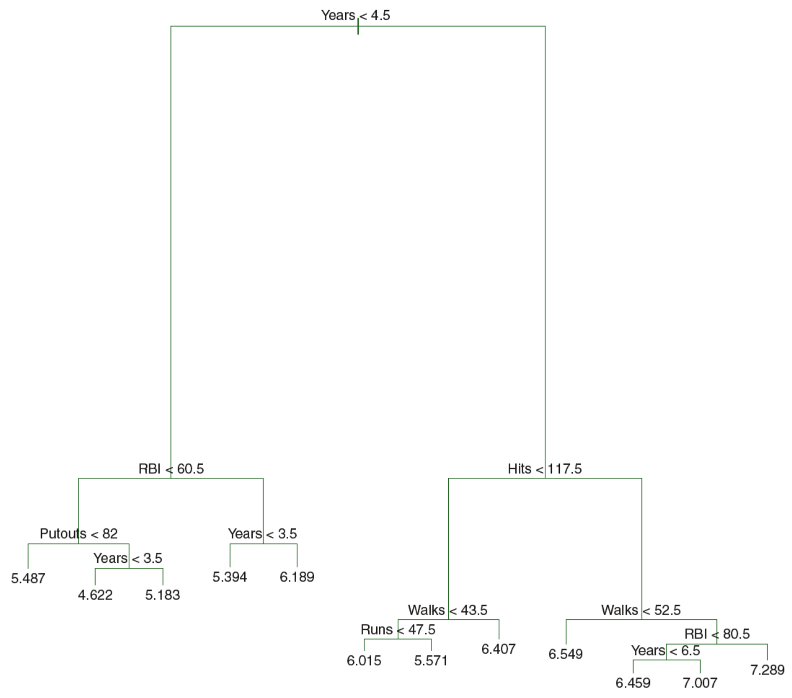


Figure 7: Baseball Players Tree Split

source: James et al. [17] (p.304)

Underlying Relationship of the Data. According to James et al. [17] (p. 314) in comparison to a regression model which assumed a linear model of the form

$$f(X) = f_0 + \sum_{j=1}^{p} X_j \beta_j + \epsilon_j,$$

regression trees assume a non-linear model of the form

$$f(X) = \sum_{m=1}^{M} \hat{\beta}_m 1(X \in R_m),$$

where $R_1, ...R_M$ are the partitioned spaces [31] (p. 3). Which model is better depends on the underlying relationship between the features and the response. If the underling relationship is linear, a linear model will perform better in predicting it. If there is a complex non-linear relationship between the features and the response, a decision tree may outperform a classical linear model. In the case of this paper the data generating process is linear.

Random Forest

According to James et al. [17] (p. 316), individual trees have high variances. If for example the training data is split into two parts and a decision tree is fit on each part, the splits will be different from one another. When two regressions were fit to these same data parts, they would yield similar results. A solution to the variance problem by trees is averaging them to reduce their variance using a sampling method called *bootstrap aggregation*. The idea is that building multiple prediction models from multiple training sets and averaging them, will produce a single low-variance model. But because only one training set is present, it is cleverer to take many samples from it with replacement. If the samples are random, some points will be chosen and others won't, but if done enough times, the average model will have the lowest variance. This is called *bagging*, which is another way of saying averaging and mathematically looks like:

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

where B is the number of total regression trees using the same number of bootstrapped training sets. The trees have low bias, because they are grown deep, but have high variance. For classification the predicted class is simply the class which was most commonly chosen by the individual trees.

By *bagging*, each time a single tree is fitted, the best feature is chosen to split on. That could be problematic, because the same feature would be chosen over all the samples. Therefore averaging wouldn't reduce the variance of the trees, because they are all correlated. *Random forest* improves the splitting procedure by choosing a random subset of features to split on, instead of the full set of features (James et al., p. 317). Usually this is the square root of the full sample, $\sqrt{p}$. Due to randomness, trees with different splits will be build, what is called *decorelating* the trees and makes the final average less variable and more reliable (James et al., p. 321).

Gradient Boosting

According to James et al. (p. 321), boosting is another model combination technique like random forest, but instead of averaging like the random forest, the technique combines sequentially built correlated trees. The next tree learns upon the previous. That means that the previous tree should be of sufficient quality.

The algorithm, adapted from James et al. [17], begins by setting the ensemble model $\hat{f}(x) = 0$ and the residuals to the actual outcome $y_i = r_i$ for all i in the training set. Then repeating the following procedure for all sequentially fitted trees, $b = 1, 2, ..., B$:

1. Fit a tree $\hat{f}^b$ with d splits to the training data (X, r).

2. Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$

3. Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$$

4. Finally output the boosted model,

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

In comparison to bagging, this is a summation of sequentially fitted models, where the current tree is fit on the residuals r_i rather than on the outcome y_i. The decision tree is then added to the fitted function using a small learning rate λ. The residuals are then updated. The fitted trees are not grown deep, such as in random forest, instead they are kept small using the tuning parameter d, which signifies the number of splits per tree. By fitting small trees, the algorithm learns slower. That is important, because the learning is sequential. To achieve good performance however a sufficient number of trees B are necessary (James et al., p. 323). The learning rate λ, the number of trees and number of splits can be chosen using a technique called *cross-validation.* Cross-Validation is applied for the tuning of random forest and gradient boosting in Section 4.

Model Tuning Using Cross-Validation

Cross-validation is used for the tuning of the parameters of random forest and gradient boosting later. This is done automatically using a designated R-package, therefore it is important to cover this concept, because the package leaves the impression of a black box.

Cross-validation is repeatedly drawing samples from a training set and refitting the model on each sample in order to remove variability due to randomness in the data [17]. Therefore it is useful to fit a model more than one time on different samples of the training set. Further is cross-validation used to tune machine learning methods by trying different tuning parameters and choosing the best combination.

The *train* function part of the *caret* package [18] helps automatically evaluate the effect of different tuning parameters on performance, using resampling, and chooses the best model [19]. To be tuned features include number of trees, complexity of the tree and learning rate among others. For random forest and gradient boosting; the *train* function creates a grid of tuning parameters.

Many models are trained on different combinations of the tuning parameters and sub-samples of the training set. The performance of the hold out samples is calculated across each fold. The best combination of parameters is chosen and the entire training set is used to fit the final model.

Accuracy as a Performance Measure

The performance of predictions in this paper will be assessed using accuracy, which according to Gatto [20] is the standard performance measure for classification exercises. Accuracy is the number of correct predictions over the total number of predictions. In the two-alternative case it is expressed by

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{3.9}$$

where the TP is the number of true positives, TN the number of true negatives, FP the number of false positives and FN the number of false negatives. These numbers can be expressed in a four by four confusion matrix. However the classification exercise in this paper is between three alternatives, therefore the confusion matrix takes on more dimensions, which is shown in Table 2.

		Observed Class		
		TF	HB	NC
Predict Class	TF	TP_{TF}	E_{HBTF}	E_{NCTF}
	HB	E_{TFHB}	TP_{HB}	E_{NCHB}
	NC	E_{TFNC}	E_{HBNC}	TP_{NC}

Table 2: Example of a Confusion Matrix with three Cereal Alternatives

For three alternatives (TF, HB and NC) accuracy is determined by adding up the values on the diagonal and dividing by the sum of all values to determine the percent correctly predicted. All values not on the diagonal are miss-classifications.

This section covered the theoretical framework of the discrete, linear and tree-based methods and explained the accuracy-performance-measure and how random forest and gradient boosting are tuned with the help of the *caret* package. Section 4 will next cover the methodology of the data simulation and fitting and tuning of the models.

4 Methodology

The methodology section is probably the most important section of this paper. It explains, step by step, how the data was simulated based on the demand model presented in the theory section and how the classical and machine learning methods were tuned and fitted. The method is implemented in the open source statistical programming software R [28], however it is not R-specific and can be implemented in other information processing software, for example in Python. The statistical methods, which were fitted include a discrete choice model: logit; two linear models: regression and LASSO; and three tree-based methods: decision tree, random forest and gradient boosting. The *Fitting and Tuning* part of the section explains step by step how the methods were implemented and what the outputs were. The code for the data simulation and the fitting of the models can be found in Appendix A. The results are then presented in Section 5.

Data

The goal of this exercise is to simulate data, assign *true* parameters, then fit a discrete, linear and machine learning models on the data and see how good the models estimate the true parameters. A second goal is to be able to manipulate the data by increasing or decreasing the number of features to confirm or reject the hypothesis that classical methods overfit the data when the number of parameters is large. This implies a comparison of the prediction accuracies between all models.

Simulation of data is the preferred method, because it allows the assignment of true coefficients. Another reason why data simulation was done, instead of relying on real world data is that a data split would be required for the training and testing of most machine learning methods. However available real world data is cross-sectional in nature. A data split could in this case not guarantee that both sets have similar underlying parameters, i.e. are balanced. A third reason for simulating data instead of using real world data is having the freedom to determine the data generating process and i.e. the relationship between the features and the response. In the case of this paper, the relationship is linear and is based on the demand model framework presented in Section 2.

Note on Simulating Utility

Utility is not observable in the real world by the researcher. Compared to the real world data collected by Bajari et al. [1] on salty snacks and Paredes et al. [27] on car ownership, the cereals data in this project was simulated. The data simulation, which is discussed next, calculated utilities using hypothetical product characteristics. Therefore it was possible to generate probabilities, which are not observed in the real world. In the real world, only the person, who makes the choice to either buy or not buy a product has knowledge of their utility. Further, the researcher only observes the choice that the consumer made, but doesn't know if the consumer hesitated between two products or if the choice was clear, i.e. probabilities are also unobserved.

4.1 Data Simulation

The following section explains the specific steps performed to simulate the data. The data simulation below is presented with $p = 10$ features and $n = 100$ observations. An example of the typed code is found in Appendix A. In the appendix-code the observations are set to $n = 300$, otherwise the execution is exactly the same.

Assign Indexes

First, the number of observations n is generated, where

$$n = 1, ..., N$$

and $N = 100$. Then the number of product characteristics p is set, where

$$p = 1, ..., P$$

and $P = 10$. Only 3 of these features p will have non-zero coefficients. By design individuals face a choice situation between 3 breakfast cereal alternatives j, where

$$j = 1, 2, 3\,.$$

This index is only for presentation purposes and it is not actually used in the code, since the data for each alternative, j, is calculated in a separate formula (see Appendix A). This was done with the purpose of avoiding the building of three-dimensional matrices that have n, p and j as dimensions. Instead two dimensional matrices were build with dimensions n and p, which were simply repeated three times.

Assign Coefficients

The coefficients for the *three relevant* features: price, quality and popularity are set to -5, 3 and 5. The relevant features will always be denoted $X1$, $X2$ and $X3$, while the non-relevant features are denoted $X4, X5, ...XP$. Price ($X1$) has a negative coefficient because it reduces utility. Quality-Score ($X2$) and Popularity-Score ($X3$) both increase utility, therefore have positive coefficients. By setting the true coefficients, one can later determine how far model estimates deviate from the true coefficients.

A vector called *sparce* is created, which is a count of the features that are assigned zero coefficients. The vector is made to vary with the number of features p. If for example the number of features p is increased from 10 to 20, then the number of features which receive zero coefficients are no longer 7 as in the case of $p = 10$ and instead 17, etc. The features with non-zero coefficients stay the same three features, regardless if the model has 10 or 90 features. The generated coefficients are combined in a one-dimensional beta-vector,

$$\begin{bmatrix} -5 & 3 & 5 & 0 & 0 & \dots & \beta_p \end{bmatrix}.$$

Generate Product Characteristics (X-Variables)

By design individuals face a choice situation between three fictional cereal-alternatives with three relevant product characteristics $\tilde{p}$. Therefore, three vectors of product characteristic-means (μ) are next created (one pro alternative). The idea with the means is that the products don't all cost the same in each store, therefore vary around a mean value. Each *relevant* characteristics-vector is $\tilde{p}$ long and is assigned the values in Table 3.

Table 3: Product Characteristics Table

Alternative Index	Alternative Name	μ Price	μ Quality Score	μ Popularity Score
j=1	TF (Tiger Flakes)	€3.90	5	2
j=2	HB (Honey Bits)	€3,50	1	4
j=3	NC (Nougat Crisps)	€1.50	1	2

A quality score of 5 contributes more to utility than a quality score of 1. The same ranking applies to popularity. For example, for the first alternative, the vector of means is

$$\begin{bmatrix} 3.9 & 5.0 & 2.0 & 0 & 0 & \dots & \mu_p \end{bmatrix},$$

where the first three characteristic-means take the values from the table and the rest are assigned zero. *Quality* is a product-specific variable. *Popularity* is also a product-specific variable reflecting the reality that national brands are more recognized than store brands. According to Table 3, *Tiger Flakes* is the most expensive alternative. *Honey Bits* is a bit cheaper and compensates for lower quality with the highest popularity. *Nougat Crisps*, which models a store brand cereal, lacks on both quality and popularity, but compensates with a very low price.

Generate Covariance Structure of X-Variable. Then the covariance structure of the X's is created through a diagonal matrix. On the diagonal the variance of each of the variables is set to 0.2. The variance is not arbitrary and it has implications on the signal-to-noise ratio explained later. All elements not on the diagonal are zero, meaning the X's are independent from each other.

Then the X's are generated using the *rmvnorm* function from the *mvtnorm* library [11] setting μ to the generated means-vectors and σ to the newly created covariance matrix. The product characteristics vectors are n-observations long and p-characteristics wide. The vector of X-s for the first alternative ($j = 1$) is

$$\begin{bmatrix} x_{1,1}^{(1)} & x_{1,2}^{(1)} & \dots & x_{1,p}^{(1)} \\ x_{2,1}^{(1)} & x_{2,2}^{(1)} & \dots & x_{2,p}^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n,1}^{(1)} & x_{n,2}^{(1)} & \dots & x_{n,p}^{(1)} \end{bmatrix},$$

where the rows are the observations n and the columns are the features p. Since there are 3 alternatives j, three such matricies are generated. The column of prices is $p = 1$, the column of qualities, $p = 2$ and the column of popularities is $p = 3$. The remaining columns of generated X's vary around 0, since their means were assigned to be zero.

Name the Columns. In the next step, the columns of the generated matrices are assigned names. For example by $p = 10$, the columns of the *Tiger Flakes* alternative are called $1.TF$, $2.TF$, ..., $10.TF$ and so on for the other two alternatives. The naming is important especially for the logit and linear models, which allow for interpretation of the coefficients. The naming is coded so it adapts to the variation of the number of features p.

Calculate Representative Utility

Now that the features (X's) and coefficients (β's) have been generated, the representative utility for the three alternatives is calculated using matrix multiplication. The model for calculating representative utility (3.2) serves as a basis, however with a slight modification of the index, to avoid a three-dimensional matrix. The multiplication

$$V_n = \beta_p * t(X_{n,p}),$$

is performed three times (one for each alternative). The resulting three vectors are combined into a single vector V, which because of the combination takes on a new dimension j or $V_{n,j}$. The rows represent the observations n and the columns represent the three alternatives as such:

$$\begin{bmatrix} v_{1,1} & v_{1,2} & v_{1,3} \\ v_{2,1} & v_{2,2} & v_{2,3} \\ \vdots & \vdots & \vdots \\ v_{n,1} & v_{n,2} & v_{n,3} \end{bmatrix}.$$

Table 4 provides the summary statistics of the generated representative utilities. It can be seen from the table that the means of the representative utilities of the three alternatives are close to each other. That is because the goal of the simulation is that all three alternatives are incorporated in the response, instead of just only one or two alternatives. Therefore they have to yield a comparable amount of utility.

Table 4: Summary Statistics of Generated Utilities

V1	V2	V3
Min. :-6.176	Min. :-3.674	Min. :-3.395
Median : 5.508	Median : 6.087	Median : 5.802
Mean : 5.759	Mean : 5.718	Mean : 5.749
Max. :15.053	Max. :16.076	Max. :13.755

Generating the Random Term ϵ

The random term ϵ serves to model product or consumer specific characteristics that are not observed by the researcher. It is generated by drawing $n * j$ random numbers from a Gumbel distribution with a location 0 and scale 1.72,

$$\epsilon \sim \text{Gumbel}(0, 1.72).$$

This is done using the *rgumbel* command part of the *evd* [33] package. The random draws are ordered into a $n * j$ matrix which is then added to the combined representative utility according to the utility model (3.1). The matrix addition looks like

$$\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} \\ u_{2,1} & u_{2,2} & u_{2,3} \\ \vdots & \vdots & \vdots \\ u_{n,1} & u_{n,2} & u_{n,3} \end{bmatrix} = \begin{bmatrix} v_{1,1} & v_{1,2} & v_{1,3} \\ v_{2,1} & v_{2,2} & v_{2,3} \\ \vdots & \vdots & \vdots \\ v_{n,1} & v_{n,2} & v_{n,3} \end{bmatrix} + \begin{bmatrix} \epsilon_{1,1} & \epsilon_{1,2} & \epsilon_{1,3} \\ \epsilon_{2,1} & \epsilon_{2,2} & \epsilon_{2,3} \\ \vdots & \vdots & \vdots \\ \epsilon_{n,1} & \epsilon_{n,2} & \epsilon_{n,3} \end{bmatrix}.$$

Because the three products are so designed, that there is no clear winner based on their representative utility. The decision will ultimately be tipped in favor of the alternative that is added a large positive random error.

Signal-to-Noise Ratio

Now that the utilities are calculated, the model is adjusted to the logit model assumptions. According to Train [36] (p. 48), the variance of the unobserved factors ϵ should be $(\pi^2)/6$ or 1.64 in order to satisfy the extreme value distribution-assumption of the logit model. Therefore, in the next step the mean variance of the error terms, known as the *noise*, is calculated. The mean variance of the representative utilities, known as the *signal* is also calculated. Their ratio is the *signal-to-noise ratio*. The *scale* parameter part of the *rgumbel* command is adjusted so the resulting noise is 1.64 and the resulting signal to noise ratio is around 2.5.

Calculating Logit Probabilities

The next step is to convert the calculated utilities into logit probabilities, as was discussed in Section 2. The generated utility is fed through the logit choice probability (3.4), however utility U, is here used in the formula instead of representative utility V. The formula is $P_{ni} = \frac{e^{U_{ni}}}{\sum_j e^{U_{nj}}}$. This is done in order to account for the added random error, ϵ. A generated summary of the probabilities P_{nj}, shows that

$$0 \leq P_{n,j} \leq 1.$$

Using the *apply* command and the *sum* function it was also confirmed that the choice probabilities of all alternatives, within an observation are

$$\sum_{i=1}^{J} P_{ni} = \sum_{i=1} \frac{e^{U_{ni}}}{\sum_j e^{U_{nj}}} = 1\,.$$

Generate Response Variable y

Now that the logit probabilities are correctly generated, they are next transformed into multinomial draws. Using the *apply* command, the *rmultinom* function is applied on in the previous step generated probability matrix to compute the multinational draws. One random vector is drawn for each row of the probability matrix. Using the generated probabilities, one object is chosen. As seen in the output, where the probability is highest, is the chance higher for that column to take on the value of 1.

$$\begin{bmatrix} 0.05 & 0.95 & 0.00 \\ 0.01 & 0.00 & 0.98 \\ 0.23 & 0.32 & 0.45 \\ \vdots & \vdots & \vdots \end{bmatrix} \to \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ \vdots & \vdots & \vdots \end{bmatrix}.$$

Again using the *apply* function the probabilities equal to 1 are assigned their respective column indicators (1, 2 or 3). Column one corresponds to the alternative *Tiger Flakes*, column two to *Honey Bits* and column three to *Nougat Crisps*. This yields the vector

$$\begin{bmatrix} 2 \\ 1 \\ 3 \\ 3 \\ \vdots \\ y_n \end{bmatrix},$$

which takes on the values $y_n = j, where\, j = 1, 2$ or 3. Hereby is the categorical response variable y generated.

In one last step, the y-categorical variable is transformed into a factor, with 3 levels, named *choice*. In order of column appearance, *TF* (*Tiger Flakes*) is assigned to level 1, *HB* (*Honey Bits*) to level 2 and *NC* (*Nougat Chrisps*) to level 3, as such:

$$\begin{bmatrix} 2 \\ 1 \\ 3 \\ 3 \\ \vdots \\ y_n \end{bmatrix} \to \begin{bmatrix} HB \\ TF \\ NC \\ NC \\ \vdots \\ y_n \end{bmatrix}.$$

Tabulating the vector of the y-s shows the response choices in Table 5. An almost equal distribution of chosen alternatives resulted. This is because the three alternatives were designed to have price, quality and popularity characteristics that result in nearly the same representative utilities. Only the random error term is responsible for tipping the scale in favor of one or in favor of another alternative.

Table 5: Tabulation of
Response Variable (n=100)

TF	HB	NC
31	34	35

Safe Data to a Data Frame

Finally, the generated y's and generated x's are combined into a data frame which has $p * j + 1$ variables and n observations. In the so far presented case of $p = 10$ features and $j = 3$ alternatives, there are 10*3+1 columns in the data frame. The 31st variable is the response variable $choice$. The saved data frame is called *cereals* in the r-data file and looks like

$$\begin{bmatrix} HB & x1.TF_1 & \dots & x10.TF_1 & x1.HB_1 & \dots & x10.HB_1 & x1.NC_1 & \dots & x10.NC_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ TF & x1.TF_n & \dots & x10.TF_n & x1.HB_n & \dots & x10.HB_n & x1.NC_n & \dots & x10.NC_n \end{bmatrix},$$

where each observation has one row. It can be seen from the matrix, that each alternative has its very own features (*x.TF, x.HB* and *x.NC*). The naming of the levels is important, so the data transformation for the logit model could be executed.

A second data frame is created for the purpose of fitting linear models. This data set includes the elements of the first data frame and additionally the continuous non-transformed logit probabilities, from which the categorical variable was calculated. Each alternative has its own probability vector, therefore that adds three additional columns to the data frame. The data frame is named *cereals.lin* in R and looks like

$$\begin{bmatrix} HB & p_{1,1} & p_{1,2} & p_{1,3} & x1.TF_1 & \dots & x10.NC_1 \\ \vdots & \vdots & \dots & \vdots & & & \\ TF & p_{n,1} & p_{n,2} & p_{n,3} & x1.TF_n & \dots & x10.NC_n \end{bmatrix}.$$

Including the logit probabilities in the data frame is important, because the linear models (linear regression and LASSO) only work with continuous response variables.

Because the logit probabilities were used for the calculation of the y-variable, the prediction performance of models fitted on both data sets *cereals* and *cereals.lin* can be compared.

4.2 Fitting and Tuning the Models

Logit Model

Data Split. The data is split in half: 50% for the sample on which the model will be trained and 50% for the sample on which the performance of the fitted model will be assessed. A general rule for data splitting is 70-30, however because the data is randomly simulated, and there is enough of it generated, $n = 300$[1], it does not make a difference what the split is. The training sample is named *cereals_fit* and the testing sample, *cereals_test.*

Data Preparation. The fitting of the logit model will be done using the R package *mlogit* [7]. Using R's own *glm* (generalized linear model) command, of the family *binomial*, only works for a discrete situation with two alternatives. In this case the choice is between three alternatives.

Before a model could be fit, the data is transformed into a format for the multinomial logit model using the *mlogit.data* command. The following settings are specified:

1. The *cereals_fit* sample is selected.
2. All columns except the response column are selected as *varying*.
3. The *shape* is chosen as "wide", because the data is structured so, that each line hast its own observation.
4. The response variable is selected for the parameter *choice.*

The mlogit.data command transforms the training data frame, *cereals_fit* into the mlogit.data frame named *Cereal_fit* and looks like

$$\begin{bmatrix} FALSE & HB_1 & x1_1 & \dots & x10_1 \\ TRUE & NC_1 & x1_1 & \dots & x10_1 \\ FALSE & TF_1 & x1_1 & \dots & x10_1 \\ \vdots & \vdots & \vdots & \vdots & \\ TRUE & HB_{\frac{n}{2}} & x1_{\frac{n}{2}} & \dots & x10_{\frac{n}{2}} \\ FALSE & NC_{\frac{n}{2}} & x1_{\frac{n}{2}} & \dots & x10_{\frac{n}{2}} \\ FALSE & TF_{\frac{n}{2}} & x1_{\frac{n}{2}} & \dots & x10_{\frac{n}{2}} \end{bmatrix},$$

where each observation has three rows, for the three alternatives, with the chosen alternative labeled as *TRUE* and the rest as *FALSE.* The *mlogit.data* transformation enables the logit model to

[1]Initially the number of observations n, was set to 100, however it was determined that after the 50% split, there were too few observations left for each class. Therefore the number of observations was increased to 300. This way, after the data-split, 150 observations remain to be divided under the three classes, which make 50 observations per class.

sort which features belong to which alternatives. The sorting is done based on the names of the features. This enables the listed features to fit into p columns. In contrast the non-transformed data frame had $p * j$ columns. The length of the columns is the total number of observations n divided by 2, because of the 50:50 data split.

Model Formula. A flexible formula is created that adjusts to the variation of features and looks like

$$Cereal\_fit\$choice = \beta_1 * X1 + \beta_2 * X2 + ... + \beta_{10} * X10.$$

The linear utility model (3.1) serves as basis. *Choice* is the response variable, and the features are the (by *mlogit.data*) transformed columns: $X1$ to $X10$. In this specific case, $p = 10$. The intercept is removed.

Fitting the Model. The model is fit using the *mlogit* command from the *mlogit*-library using the model formula, the train-data-sample and a seed of 1.

Making Predictions. In-sample-fit is obtained by using the *fitted* command. This allows the model to predict the response values on which it was fitted on. The resulting predictions are in the form of logit probabilities and follow the properties of the logit model: add up to 1 and are between 0 and 1. The out-of-sample fit is obtained by using the *predict* command, the fitted model, and the test-sample.

Assessing Performance Using the *apply* command the highest predicted probability is sought and its level name is assigned to a new vector called *y_hat*. This vector is a character vector, which consists of the alternative abbreviations (*TF, HB* and *NC*). The vector is then compared to the true response vector y or in R called *cereals_fit$choice*. This comparison looks like

$$\begin{bmatrix} NC \\ NC \\ HB \\ HB \\ \vdots \\ \hat{y}_{\frac{n}{2}} \end{bmatrix} == \begin{bmatrix} NC \\ NC \\ HB \\ TF \\ \vdots \\ y_{\frac{n}{2}} \end{bmatrix}.$$

In the case of in-sample fit the *fitted* $\hat{y}$ is compared to the train-sample-y, while in the case of out-of-sample fit, the predicted $\hat{y}$ is compared to the test-sample-y. Using the *confusionMatrix* command from the *caret* package [18] a confusion matrix is build and the accuracy is calculated in accordance with 3.9. The resulting in-and out-of-sample accuracies are bound in a vector, which is stored for presentation purposes. The results will be presented in Section 5.

Linear Regression

Data Split. The data split is the same 50:50 split as by logit, however the data frame *cereals.lin*, meant for linear models, is used. As a reminder this data frame has three additional columns, made up of the logit probabilities, to serve as continuous response variables.

Partition the Data for Linear Regression. As described in Section 2, linear models can predict categorical responses, when the *one-vs. rest* approach is applied. This approach includes fitting a linear model for each class. Therefore the training sample is divided into three training samples, named *cereals.TF, cereals.HB* and *cereals.NC*. Each sample is made up of the respective logit probability as a response variable, and the list of features pertaining to that class. For example, the matrix

$$\begin{bmatrix} p_1 & x1.TF_1 & \dots & x10.TF_1 \\ \vdots & \vdots & \dots & \vdots \\ p_{\frac{n}{2}} & x1.TF_{\frac{n}{2}} & \dots & x10.TF_{\frac{n}{2}} \end{bmatrix}$$

shows the training sample for the the *Tiger Flakes* class with $p = 10$ features. The same partition method is applied to the remaining classes (HB and NC) in the train and test samples.

Model Formula. Because there are three train samples, three flexible formulas are coded (one for each class) so that the regression formula can adjust to variations of the number of features p. For example, the formula for the first class, *Tiger Flakes* with $p = 10$ features, is

$$log(p_1^{TF}) = \beta_1^{TF} * X1.TF + \beta_2^{TF} * X2.TF + ... + \beta_{10}^{TF} * X10.TF\,.$$

The superscript *TF* is there to indicate that each regression yields its own vector of coefficients. The intercept is removed.

Fitting the Model. Three linear regression models are fit using the respective training samples and formulas. The estimated coefficients for $p = 10$ features are presented in Section 5.

Making Predictions. Using the fitted models, similar to logit the *fitted* and *predict* functions are applied. Predictions are made using the also partitioned test samples.

Assessing Performance The fitted and predicted log-probabilities from the three classes are combined into a single vector. From this vector the highest log-probability per observation is chosen using the *apply* command with the function *which.max(x)*. The column indicator of that probability is displayed in a new vector reflecting the same level-assignment as in logit. As a reminder: *TF* (*Tiger Flakes*) is assigned to level 1, *HB* (*Honey Bits*) to level 2 and *NC* (*Nougat Chrisps*) to level 3. This new vector is named $\hat{y}$.

A confusion matrix is build using the fitted-vector $\hat{y}$ and the response y from the training sample (in-sample-fit). Then a second confusion matrix is build using the predicted-vector $\hat{y}$ and the response y from the testing sample (out-of-sample fit). Adding up the diagonal of the confusion matrix and dividing through the sum of all of its components yields the accuracy. The in-and out-of-sample accuracies are saved in a vector for presentation purposes in Section 5.

The generated y-test partitions for the regression classes are not used to assess performance, but only for the prediction. Performance is assessed by comparing the categorical response variable *cereals$choice* to the predicted alternatives. The alternative that yielded the highest utility is chosen as the predicted value.

LASSO Regression

Data Split The fitting of the LASSO regression assumes the same data split as by the linear regression, made from the data frame called *cereals.lin*. Because LASSO is also a linear method, the *one-vs. rest* approach and its respective 3-class partition of the training-data applies. Hence, LASSO uses the same partitioned training data (*cereals.TF, cereals.HB* and *cereals.NC*) as linear regression.

Prepare Data for LASSO with Cross-validation For the purposes of cross-validated LASSO the training sample is divided into X-Train and Y-Train data. This is done for each alternative class. The response variable is a vector of the log-probability and the features vector is the list of product features. An example for the class *TF* (j=1) is displayed in Figure 8.

$$\begin{bmatrix} log(pr_{1,1}) \\ log(pr_{2,1}) \\ \vdots \\ log(pr_{\frac{n}{2},1}) \end{bmatrix}$$

Y-Train Vector for *TF*

$$\begin{bmatrix} X1.TF_1 & \dots & X10.TF_1 \\ \vdots & \dots & \vdots \\ X1.TF_{\frac{n}{2}} & \dots & X10.TF_{\frac{n}{2}} \end{bmatrix}$$

X-Train Matrix for *TF*

Figure 8: Train Data for LASSO

The X-Test data is sourced from the testing sample for the purpose of assessing the performance of the prediction.

Cross Validation for LASSO. During the model tuning, first a lasso without cross-validation was fitted using the *rlasso* function from the *hdm* package [5]. It was found that LASSO with cross validation (CV) yields better and more stable results. Through the *cv.glmnet* function of the *glmnet* package [10], CV is applied using the newly generated Y-Train and X-Train data. The *family* parameter is set to *gaussian*, because a continuous y-variable is present, in the form of a logit probability. All other settings of the command *cv.glmnet* including the number of folds (10) were left to their defaults.

Model Formula. No model formula is necessery by LASSO, because the X-and Y-Train variables are specified in the *glmnet* command.

Fitting the Model. LASSO is fitted using the *glmnet* command using the same parameters as for the cross-validation, but this time taking the *minimum lambda* obtained through the cross-validation. The *selected* coefficients of the fitted LASSO are saved and are then run through the *lm* command to generate Post-LASSO coefficients, including significance levels and standard errors. These coefficients are saved in a vector for presentation purposes in Section 5.

Making Predictions. In-sample-predictions are made using the minimum lambda that was determined during the cross-validation. Out of sample predictions are made with the cross-validated LASSO model. For details, please see the R Markdown code in Appendix A.

Assessing Performance. The in-and out-of-sample predictions, composed of one vector per alternative, are in the form of utilities. They are combined into a single vector and the largest utility receives the column index of its respective column. This vector is then compared to the numeric of the response vector of the testing sample, displayed in Figure 9.

As a reminder, the response variable in the testing sample is a character variable with the levels (TF, HB and NC). Because the prediction outputs of LASSO and linear regression are in the form of column indicators instead of character vectors, such as by the logit predictions, the response variable needs to be transformed to a numeric for the purpose of comparison. Transforming the response variable to a numeric drops the characters and the levels, which are assigned the same as the column indicators, remain. Level 1 remains the alternative TF, level 2, HB and level 3, NC. The comparison is presented in Figure 9.

A confusion matrix such as in Table 2 is built using the *table* function. Then accuracy is calculated according to Equation 3.9 by summing the diagonal values of the confusion matrix and dividing

$$\begin{bmatrix} 1 \\ 1 \\ 2 \\ \vdots \\ \hat{y}_n \end{bmatrix} == \begin{bmatrix} 1 \\ 1 \\ 3 \\ \vdots \\ y_n \end{bmatrix}.$$

Figure 9: LASSO Prediction Comparison

them through the sum of all of its components. The fraction of correctly predicted choices is saved in a vector for presentation purposes.

The Y-Test data is not sourced for the prediction comparison between predicted and actual responses. This comparison is made using the unpartitioned categorical data, as was the case for linear regression. Please see the detailed code in Appendix A for details.

Decision Tree

Data Split. The data split used for the logit model, is also used for the decision tree.

Prepare data. Extra data pre-processing as in the case of logit, regression or lasso is not required, because the tree algorithm can alone determine the best split.

Model Formula. The formula for the example case of $p = 10$ is

$$\begin{aligned} cereals\_fit\$choice = \beta_1 * X1.TF + ... + \beta_{11} * X1.HB + ... \\ +\beta_{21} * X1.NC ... + \beta_{31} * X10.NC \end{aligned} \tag{4.1}$$

or simply the y-categorical variable *choice* is dependent on all remaining variables. According to the formula, the tree-based method has $p*j$ features to split on, in this case 10*3. Logit for example was able to order the features to the respective responses, only having to estimate p features. For regression and LASSO, the assignment was done manually by partitioning the data by class. Therefore logit and the linear methods have only p features to estimate, while the decision tree and the subsequent tree-based methods have $p * j$ features to split on.

Fitting the Model. The tree is fit using the training data and the *rpart* command from the *rpart* package [34]. The tree is tuned with the *method* parameter *class* for classification and the complexity parameter *cp*, which is used to control the size of the decision tree by deciding the number of splits.

The parameter $cp = 0.001$ was used for the initial fit. The tree is then pruned in order to avoid overfitting the data. Looking at the *xerror* column printed by *printcp()*, the complexity parameter that yields the minimum cross-validated error is chosen. Taking the optimal cp according to this mechanism for the last run yields a poor classifcation (33%). In this case the tree even chooses only two alternatives and leaves the third one completely out. Therefore a cp level that yields gut average results over all runs was chosen $cp = 0.06451613$.

For this reason the optimal cp displyed by the *printcp()* function for all runs does not match the chosen average cp. The difference in out-of-sample performance however is minimal (49% compared 51%) for run 1. The advantage is that the last run now shows a good out-of-sample classification (47% with the average cp, compared to 33% with the minimal cp). Tree diagrams are included in Section 5 show on which variables the split was chosen.

Making Predictions. The in-sample-prediction is implemented using the fitted tree-model, the *predict* function and the train-data set. The out-of-sample prediction repeats this last step, however with the testing-data set. The predictions in both cases are in the form of probabilities, which are all between 0 and 1 and sum to 1.

Assessing Performance The probabilities vectors are run through an *apply* function with the goal of outputting the column indicator that has the highest probability. The vector $\hat{y} = 1,\ 2$ or 3 is outputted. The predicted vector is then compared to the response variable y from the testing sample, which in turn is transformed to a numeric in order to replace the characters with level indicators ($TF = 1, HB = 2$ and $NC = 3$). The two vectors are tabulated into a confusion matrix using the *table* command and accuracy is calculated by summing up the diagonal of the matrix and dividing it through the sum of all of its components.

Random Forest

Data Split. Random forest used the data split already used by the other classifiers (logit and decision tree).

Prepare Data. Because the random forest chooses the best variables to split on, not data preparation is done.

Model Formula. The model formula is the same as for the decision tree (4.1).

Fitting and Tuning the Model: Manually. First the random forest was fit using the *randomForest* package [21]. Different tuning parameters such as adjusting *mtry* (the learning rate), *ntree* (the number of trees) and the note size were tested and the in-and out-of-sample fit was observed. According to Breiman's Manual [3] on *Setting up, Using and Understanding Random Forests*, $mtry$ should be the square root of p to yield optimal prediction results. This provided indeed the best fit. The parameter *ntree=500* was found to be optimal, because the reduction of error, at that number of trees, was found to be the greatest.

Fitting and Tuning the Model: Automatically. A second method for automatic fitting and tuning the random forest with the caret package [18] was implemented to see if out of sample performance can be improved. Using the *trainControl* function, 5 folds of cross validation were assigned. Then using the *train* function, the model formula, the train sample, *rf* as the *method* and the *trainControl* parameter from the previous step, the random forest was trained. Due to the implemented cross-validation, the variance and bias of the random forest were reduced. The method with cross-validation proved to yield better out-of-sample performance than the manual method, and was adapted for the final results. Variable importance was also ascertained to see if the random forest is finding the three relevant variables to split on. Figures with the findings are presented in Section 5.

Making Predictions. As predictions, the random forest outputted the character-names of the alternatives, instead of outputing their probabilities, such as in the case of linear regression, lasso or tree. Therefore it wasn't necessary to determine which probability is the highest. Therefore a direct comparison with the training and testing sample was possible.

Assessing Performance. Performance was assessed the usual way: by tabulating the predictions and the true responses in a confusion matrix and calculating the accuracy.

Gradient Boosting

Data Split. The data split is the same as the one used for the other classifiers (logit, decision tree and random forest).

Data Preparation. Similarly to the tree and random forest, no data preparation was necessary.

Model Formula. The same model formula 4.1 from the decision tree and random forest was used for gradient boosting.

Fitting and Tuning the Model: Manually. First, the model was fit manually using the *gbm* package [13] and the *gbm* function. The *distribution* was set to *multinomial*, since there are more than three categories. The default parameters of the *gbm* function, listed below, proved to yield the best out-of-sample results.

1. *n.tress* was set to 100 by default
2. *interaction.depth* was set to 1 by default
3. *shrinkage* or the learning rate was set to 0.1 by default

Fitting and Tuning the Model: Automatically. Similarly to random forest, boosting was trained a second time, but with the caret package [18], using the same number of cross-validation folds (5) as the random forest. This time *gbm* was selected as the *method*-parameter. Variable importance was assessed and is presented in Section 5. The training and tuning through caret proved to yield better performance than the manual training, therefore was the chosen method. The manual training is commented-out in the code, but can be reactivated for a closer look into the various tuning parameters.

Making Predictions. To obtain in-and out-of-sample predictions the *predict* command and the fitted model were used, first with the *cereals_fit* data, and then with the *cereals_test* data.

Assessing Performance. By the automatic fit (with caret) the predictions are in the form of the alternative labels (TF, HB and NC), therefore were directly compared to the test data.

By the manual fit, the predictions were in the form of probabilities. In this case the alternative with the highest probability was assigned its alternative label (level) and then the data was compared to the numeric of the testing sample.

In both cases a confusion matrix was build, from which the accuracies were calculated and saved in a vector for presentation purposes. The next section presents a comparison of the prediction results.

5 Results

In this section the prediction performances of all fitted models are presented. The models were fitted according to the specifics of the previous section.

First, the model characteristics are provided in Table 6, so the results could be reproduced. Second, the signal-to-noise ratios are provided for each run in Table 7. Third, a comparison of the prediction accuracies is provided in Table 8. Fourth, a tabulation of the categorical outcomes of the train and test samples is provided in Table 9. Next, confusion matrices that show misclassifications per class are shown for the first ($p = 10$) and last ($p = 290$) runs in Tables 10 and 11. The estimated *relevant* coefficients, for the models that allow coefficient-estimation, are shown in Table 12, while Tables 13, 14 and 15 show *all* estimated coefficients for the first run (p=10). Finally, variable importance of the tree-based methods is presented in Figures 10, 11, 12. The meaning and implications of the results is discussed in Section 6.

Table 6: Model Characteristics for Reproducibility

R version	3.6.0 (2019-04-26)
observations	$n = 300$
features	$p = (10, 50, 150, 290)$
assigned coefficients	$\beta_1 = -5$, $\beta_2 = 3$, $\beta_3 = 5$, $\beta_{rest} = 0$
cv-folds	5
total run time	2 Minutes
seed	1

Table 7: Signal-to-Noise Ratio

	Signal-to-Noise	Noise
p=10	2.24	1.64
p=50	3.02	1.64
p=150	2.43	1.64
p=290	2.84	1.64

Table 8: Prediction Accuracy of all Models (in %)

	In-Sample	Out-of-Sample
Logit	76	78
Linear Regression	76	71
LASSO with CV	72	77
Decision Tree	67	49
Random Forest	100	63
Gradient Boosting	88	61

p=10

	In-Sample	Out-of-Sample
Logit	77	62
Linear Regression	72	45
LASSO with CV	62	54
Decision Tree	58	41
Random Forest	100	45
Gradient Boosting	100	45

p=50

	In-Sample	Out-of-Sample
Logit	100	41
Linear Regression	80	33
LASSO with CV	70	65
Decision Tree	69	47
Random Forest	100	51
Gradient Boosting	100	49

p=150

	In-Sample	Out-of-Sample
Logit	100	36
Linear Regression	84	29
LASSO with CV	74	64
Decision Tree	65	47
Random Forest	100	50
Gradient Boosting	100	47

p=290

Table 9: Tabulation of Observed Choices

	TF	HB	NC
p=10	51	57	42
p=50	51	57	42
p=150	49	62	39
p=290	51	57	42

cereals_fit

	TF	HB	NC
p=10	52	52	46
p=50	44	52	54
p=150	52	57	41
p=290	48	53	49

cereals_test

	TF	HB	NC
p=10	51	57	42
p=50	51	57	42
p=150	49	62	39
p=290	51	57	42

cereals_fit_lin

	TF	HB	NC
p=10	52	52	46
p=50	44	52	54
p=150	52	57	41
p=290	48	53	49

cereals_test_lin

Confusion Matricies[2]

Table 10: Out-of-Sample Confusion Matrices, p=10 (run 1)

	TF	HB	NC
TF	40	4	3
HB	8	38	4
NC	4	10	39

Logit

	TF	HB	NC
TF	39	2	4
HB	11	37	11
NC	2	13	31

Linear Regression

	TF	HB	NC
TF	41	4	1
HB	9	40	10
NC	2	8	35

LASSO

	TF	HB	NC
TF	28	9	8
HB	19	33	25
NC	5	10	13

Decison Tree

	TF	HB	NC
TF	31	6	9
HB	19	41	14
NC	2	5	23

Random Forest

	TF	HB	NC
TF	29	4	9
HB	17	33	8
NC	6	15	29

Boosting

Table 11: Out-of-Sample Confusion Matrices, p=290 (run 4)

	TF	HB	NC
TF	17	13	18
HB	14	22	16
NC	17	18	15

Logit

	TF	HB	NC
TF	20	27	24
HB	16	10	11
NC	12	16	14

Linear Regression

	TF	HB	NC
TF	34	13	12
HB	11	32	7
NC	3	8	30

LASSO

	TF	HB	NC
TF	31	26	13
HB	6	15	11
NC	11	12	25

Decison Tree

	TF	HB	NC
TF	16	9	8
HB	24	36	18
NC	8	8	23

Random Forest

	TF	HB	NC
TF	23	18	11
HB	17	27	17
NC	8	8	21

Boosting

[2]The three fictional classes are *HB* for Honey Bits, *TF* for Tiger Flakes and *NC* for Nougat Crisps. The matrices on the page are not labeled with the goal to save space. The labeling is assumed from Figure 2

Relevant Coefficients[3]

Table 12: Relevant Coefficients

	p=10	p=50	p=150	p=290
X1	-3.10	-3.35	-28.90	-12.35
X2	1.85	1.92	16.58	6.58
X3	3.21	3.31	31.08	11.22

Logit

	p=10	p=50	p=150	p=290
X1.TF	-3.91	-4.36	-10.06	-677.76
X2.TF	1.45	1.54	4.60	634.97
X3.TF	2.19	2.92	6.15	-386.86
X1.HB	-4.35	-2.73	5.96	-41.00
X2.HB	1.09	1.37	-8.14	-49.80
X3.HB	2.69	1.11	-4.57	43.41
X1.NC	-4.86	-4.53	4.29	-3.29
X2.NC	1.49	1.07	-6.53	-7.96
X3.NC	0.95	0.78	-10.66	4.20

Linear Regression

	p=10	p=50	p=150	p=290
X1.TF	-3.90	-3.87	-4.20	-3.52
X2.TF	1.46	0.79	1.47	0.78
X3.TF	2.15	3.67	2.71	3.09
X1.HB	-4.35	-2.73	-3.57	-3.21
X2.HB	1.18	0.86	0.95	1.61
X3.HB	2.66	1.32	2.03	1.64
X1.NC	-4.87	-4.00	-5.98	-4.21
X2.NC	1.49	1.06	1.00	0.86
X3.NC	0.97	1.24	2.12	0.82

LASSO

[3]The coefficients X1, X2 and X3 refer to the features *price*, *quality* and *popularity*. For the model logit, these coefficients are grouped for the three classes together, while for regression and LASSO, they are separately estimated for each class.

Table 13: Estimated Logit Coefficients, p=10

	Dependent variable:
	choice
X1	-3.096^{***}
	(0.414)
X2	1.852^{***}
	(0.248)
X3	3.213^{***}
	(0.420)
X4	-0.004
	(0.302)
X5	-0.026
	(0.290)
X6	-0.333
	(0.310)
X7	0.001
	(0.304)
X8	0.225
	(0.296)
X9	-0.060
	(0.309)
X10	0.282
	(0.325)
Observations	150
Log Likelihood	-91.375
Note:	*p$<$0.1; **p$<$0.05; ***p$<$0.01

Table 14: Estimated Regression Coefficients, p=10

	Response: $log(pr_{TF})$
X1.TF	-3.907^{***}
	(0.437)
X2.TF	1.451^{***}
	(0.331)
X3.TF	2.192^{***}
	(0.529)
X4.TF	-0.181
	(0.554)
X5.TF	0.037
	(0.481)
X6.TF	0.068
	(0.505)
X7.TF	0.224
	(0.558)
X8.TF	1.181^{**}
	(0.556)
X9.TF	0.201
	(0.527)
X10.TF	-0.081
	(0.584)
Obs.	150
R^2	0.667
Adj. R^2	0.644
RSE	2.911
	(df = 140)
F Stat.	28.102^{***}
	(df = 10; 140)
Note:	*p<0.1; **p<0.05; ***p<0.01

	Response: $log(pr_{HB})$
X1.HB	-4.351^{***}
	(0.437)
X2.HB	1.089^{**}
	(0.548)
X3.HB	2.688^{***}
	(0.376)
X4.HB	-0.026
	(0.542)
X5.HB	0.462
	(0.566)
X6.HB	0.462
	(0.553)
X7.HB	0.334
	(0.568)
X8.HB	0.008
	(0.549)
X9.HB	-0.022
	(0.563)
X10.HB	0.204
	(0.618)
Obs.	150
R^2	0.644
Adj. R^2	0.619
RSE.	3.077
	(df = 140)
F Stat.	25.363^{***}
	(df = 10; 140)
Note:	*p<0.1; **p<0.05; ***p<0.01

	Response: $log(pr_{NC})$
X1.NC	-4.860^{***}
	(0.456)
X2.NC	1.494^{***}
	(0.464)
X3.NC	0.954^{**}
	(0.367)
X4.NC	-0.283
	(0.579)
X5.NC	-0.049
	(0.524)
X6.NC	0.304
	(0.514)
X7.NC	0.545
	(0.490)
X8.NC	-0.349
	(0.477)
X9.NC	0.076
	(0.534)
X10.NC	-0.769
	(0.518)
Obs.	150
R^2	0.720
Adj. R^2	0.700
RSE.	2.803
	(df = 140)
F Stat.	36.010^{***}
	(df = 10; 140)
Note:	*p<0.1; **p<0.05; ***p<0.01

Table 15: Estimated Post LASSO Coefficients, p=10

	Response:
	$log(pr_{TF})$
X1.TF	-3.901^{***}
	(0.427)
X2.TF	1.465^{***}
	(0.321)
X3.TF	2.150^{***}
	(0.511)
X7.TF	0.199
	(0.537)
X8.TF	1.163^{**}
	(0.536)
Obs.	150
R^2	0.667
Adj. R^2	0.655
RSE	2.863
	(df = 145)
F Stat.	58.035^{***}
	(df = 5; 145)
Note:	*p<0.1; **p<0.05; ***p<0.01

	Response:
	$log(pr_{HB})$
X1.HB	-4.348^{***}
	(0.422)
X2.HB	1.181^{**}
	(0.525)
X3.HB	2.658^{***}
	(0.358)
X5.HB	0.470
	(0.553)
Obs.	150
R^2	0.641
Adj. R^2	0.632
RSE	3.026
	(df = 146)
F Stat.	65.274^{***}
	(df = 4; 146)
Note:	*p<0.1; **p<0.05; ***p<0.01

	Response:
	$log(pr_{NC})$
X1.NC	-4.865^{***}
	(0.435)
X2.NC	1.494^{***}
	(0.452)
X3.NC	0.965^{***}
	(0.351)
X7.NC	0.545
	(0.474)
X10.NC	-0.819
	(0.496)
Obs.	150
R^2	0.718
Adj. R^2	0.708
RSE	2.764
	(df = 145)
F Stat.	73.855^{***}
	(df = 5; 145)
Note:	*p<0.1; **p<0.05; ***p<0.01

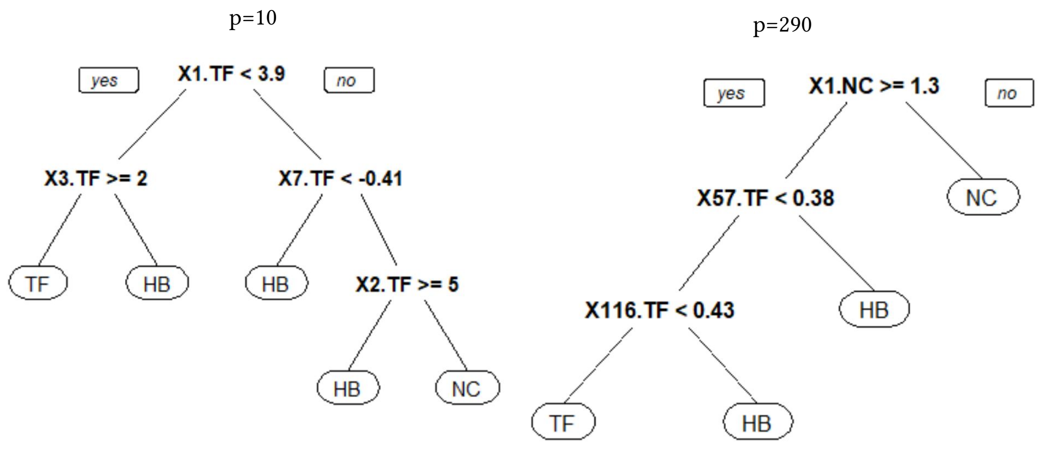


Figure 10: Decision Tree Split Results

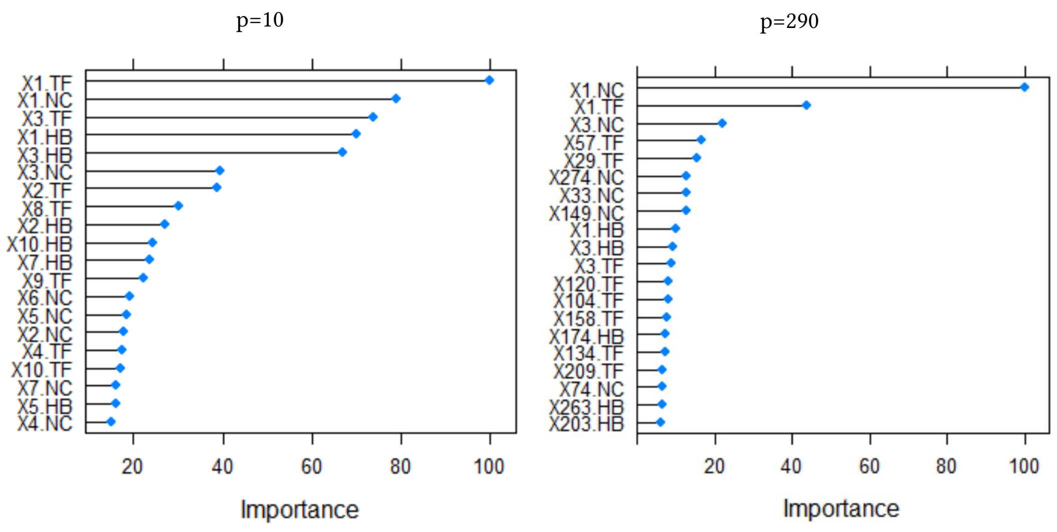


Figure 11: Random Forest Variable Importance

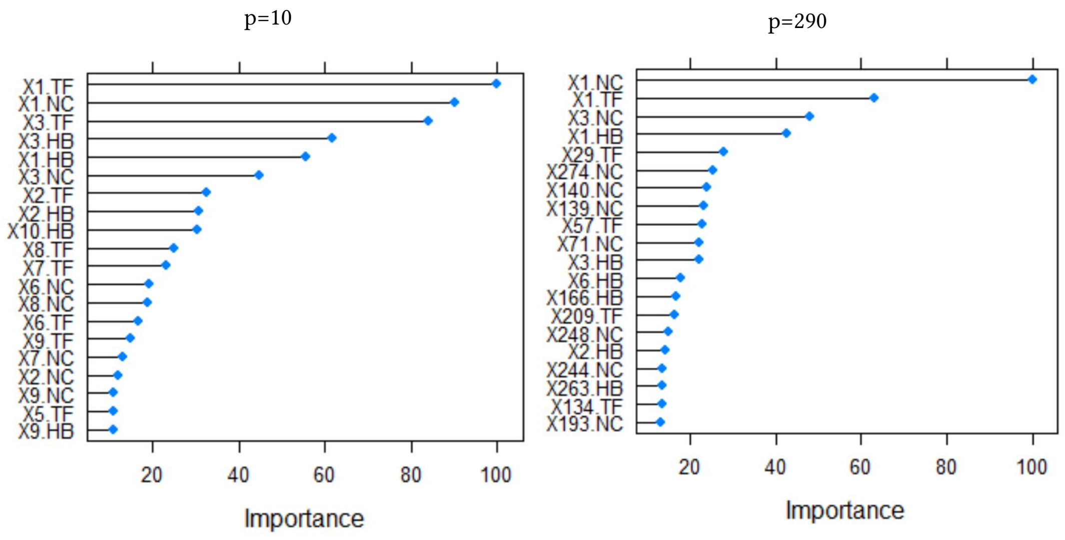


Figure 12: Gradient Boosting Variable Importance

6 Discussion

This section discusses the meaning and implications of the results. First, the hypotheses in relation to the research question are presented. Next, the performance of the fitted models is discussed.

Hypotheses

The **first** hypothesis, which considers the dimentionality problem, discussed in Section 1 is that a classic discrete choice model is good for a small number of features p, but starts to over fit as p nears the number of observations n. Due to overfitting, it is expected that for a p, which nears n, the prediction performance of the discrete model falls. The **second** hypothesis is that ML methods perform better than classical methods, when the number of features p nears the number of observations n, as suggested in Section 1. The **third** hypotheses is that linear models will have a better prediction accuracy as non-linear models, because the data generating process is linear in nature. The applied linear models in this exercise are multinomial logit, linear regression and LASSO. Applied non-linear models are decision tree, random forest and gradient boosting. A **sub-hypothesis** to the third hypothesis is the presumption that a linear model, which practices variable selection will outperform a linear model which doesn't, because the simulated data is sparse.

Model Characteristics for Reproducibility

The predictions were run with the characteristics presented in Table 6. Running the predictions with different characteristics will yield different results.

Signal-to-Noise Ratio

The signal-to-noise ratio (SNR) for all runs is presented in Table 7. The table shows that each run hat a sufficient signal-to-noise ($SNR > 2$) and conforms to the assumptions of the logit model (noise of $\pi^2/6$ or 1.64) as perscribed by Train [36] (p. 48).

Model Performance

In Table 8, the in-and out-of-sample prediction accuracies for the six models over four runs are presented. Here the focus is on out-of-sample prediction performance, which is measured in percent. Starting with the **first** run, where $p = 10$, the logit model has the best out-of-sample fit. The best explanation for this, is that the data simulation was designed for a discrete choice model that corresponds to the assumptions of the logit model. Therefore, it should be no surprise that the logit model performs the best in the first run. LASSO is close second and linear regression is third.

Lasso, which practices variable selection, performs better than linear regression. As a reminder, the *relevant* coefficients are three, where in the first run the number of features p is set to 10. That means that sparsity is present in the data. Because linear regression fits a model with all available variables, its fit will be worse than LASSO under sparsity. Next, the tree-based methods perform substantially worse, with random forest being the best of the three. The decision tree tends to perform, throughout all runs, worse than the random forest, because of its high variance. The in-sample fit of the random forest and boosting is near 100, which indicates overfitting. All attempts to reduce this in-sample-overfitting resulted in reducing the out-of-sample fit, therefore it was left as is. In contrast, the linear models show similar in magnitude in-and out-of-sample fits, which indicates a good fit.

Continuing with the **second** run, where $p = 50$, the multinomial logit model continues its leadership in prediction performance, but we see a substantial drop in the performance of all other models. LASSO remains the second best model. Linear regression, random forest and boosting all have the same performance. By 50 features the linear regression is starting to overfit and has low performance. A possible reason for the poor performance of the tree-based methods is that they have de facto $p * 3$ features to split on, while regression, logit and LASSO only estimate p features per run.

The **third** run, with $p = 150$ features, observes a massive decrease in the performance of the logit model and linear regression. LASSO is the best predictor, which is due to its linear nature and variable selection quality. Random forest and gradient boosting, which, like lasso, also make predictions based on the most important variables are second and third best. Logit, which is clearly overfitting, because of its now perfect in-sample performance, takes second to last place. At this point a decision tree is doing better than logit. Linear regression rounds out the end.

The final and **fourth** run, increases the number of features p to 290, while the number of observations n stays by 300. This is an example of what is called *wide data* [32]. Here overfitting is present by logit and linear regression, due to the big differences between their in and out-of-sample performances. The best fitted model is LASSO, clearly profiting from its variable selection quality. Its out-of-sample performance is also very close to its in-sample-performance. The next best model is random forest, which tends to perform around as good as gradient boosting throughout all the runs. Tree comes close after random forest, which is to be expected, due to its high variance. At this point all tree-based methods are outperforming the logit model and therefore validating the second hypothesis. The overfitting by the classical models (logit and linear regression) seen in runs 3 and 4 confirms the first hypothesis. The fact that LASSO, a linear model, outperforms the non-linear tree-based models, confirms the third hypothesis.

Tabulation of Observed Choices

Table 9 shows the response variable tabulated for the test and train samples. As a reminder, linear regression and LASSO were fitted using a data frame called *cereals.lin*, while the remaining methods using a data frame called *cereals.* These data frames were split in the middle to form train and test samples. The tabulation in Table 9 serves to prove that the two data frames, on which the methods were fit and then compared, are the same. This allows for a comparison of the regression methods linear regression and LASSO, with the classification methods logit, decision tree, random forest and gradient boosting.

Further, the tabulation shows that all classes were about equally chosen. That is not surprising, because their representative utilities were modeled to have similar means.

The tabulation in Table 9 also shows that the train and test sets are balanced and have similar tendencies. If one looks closely, it could be seen that the *HB* class is observed slightly more than the *TF* class, and that the *NC* class is observed the least of the three. This finding is consistent for both the test and train samples. The consistency arises from the fact that data was simulated according to the same process. Therefore a simple data-splitting method such as the 50:50 Train/Test split resulted in balanced samples.

Confusion Matrices

The confusion matrices presented in Tables 10 and 11 offer a deeper look into the out-of-sample classification performance of the six models. Specifically it was examined if any of the models have an inclination to predict a particular class better or worse, than the others.

Under perfect classification, the confusion matrices would only show values on the diagonal. These values would match the values expressed in the tabulation of observed choices in Table 9. Because the classification is not perfect however, there are values outside of the diagonal. Each classification is based on $n = 150$ observations, which is the size of the training sample. That allows for around 50 observations per class, since there are 3 classes.

Table 10, where $p = 10$ features, shows more values on the diagonal than Table 11 ($p = 290$ features), which has more miss-classifications. This is not surprising, because adding more features tends to confuse the prediction models. Overall there were no large disparities in the miss-classification between the classes. The models have no inclination toward predicting a particular class better or worse.

Estimated *Relevant* Coefficients

Table 12 shows the estimated *relevant* coefficients over all runs for the models that allow for the interpretation of coefficients: logit, linear regression and LASSO. The relevant coefficients are: *X1 = price*, *X2 = quality* and *X3 = popularity* which were set to $X1 = -5$, $X2 = 3$ and $X3 = 5$ in the data simulation.

The first sub-table shows the coefficients for logit. Because the logit model predicts the categorical response, only one coefficient for all three classes is estimated. In contrast linear regression and LASSO estimate the relevant coefficients separately for each class.

In the first two runs, the logit model predicts the coefficients with a disparity around 2 from the true coefficients. The last two runs show a bigger disparity, especially for run 3. This is due to overfitting. The regression, similarly to logit, predicts the first two runs with a small difference to the true coefficients. By run 3, overfitting is present and run 4 shows an extreme difference from the true coefficients. LASSO in contrast is able to keep its good predicting power through run 3 and 4.

The signs for all predicted coefficients are correct. However regression and LASSO are having difficulty predicting the coefficients of the class *NC* correctly, because the discrepancy by that coefficient is higher than by the other two. This could be because *NC* was slightly less chosen from the three alternatives and there were not enough observations to estimate the coefficients on.

All Estimated Coefficients, p=10

Tables 13, 14 and 15 show *all* estimated coefficients for $p = 10$ features. Because it would be difficult to show the coefficients for $p = 50$, $p = 150$ or $p = 290$, only the coefficients of the first run are presented. Details such as significance levels and residual standard errors are shown. Overall deviate the coefficients slightly from the assigned coefficients. Tree-based methods don't estimate coefficients, therefore are not shown.

In Table 13, the logit model finds the first three coefficients correctly as significant and the remaining as insignificant. Signs are also predicted correctly.

The regression coefficients presented in Table 14 result from the regressions executed for each alternative (TF, HB and NC). At $p = 10$ features the regression has no difficulty to estimate the relevant coefficients for all classes. The observations n are 150 and adjusted R^2 varies from 65% to 70%. All coefficients are significant to the 1% level except for X2.HB and X3.NC, which are significant to the 5% level. The regression erroneously estimated X8.TF as significant to the 5% level although that coefficient was set to zero in the simulation.

Table 15 shows the estimated Post-LASSO coefficients, which is a linear regression fitted after the variable selection step by LASSO. LASSO was in turn fitted just like linear regression, separately on each class. The variable selection chose correctly the relevant coefficients for all three classes, which were also significant in Post-LASSO. The variable $X1$ tends to be better predicted than the variable $X3$, due to its smaller difference from the true coefficient.

Variable Importance of Tree-Based Methods

To provide model interpretation of the tree-based methods, plots of the most important variables are presented in Figures 10, 11 and 12 for the first ($p = 10$) and last ($p = 290$) out-of-sample runs.

The decision tree plot for $p = 10$ in Figure 10 shows that the split was done overwhelmingly on relevant variables. These are the variables that have X1 to X3 in their names. In the first split (p=10) are the variables X1.TF, X2.TF and X3.Tf present. The $p = 290$ split shows that one of the relevant variables was included as the very first split (X1.NC). Since the trees are not grown deep, no further splits are observed.

The random forest variable-importance-plot in Figure 11 shows splitting on which variables contributes the most to the total decrease in node impurities averaged over all trees [21]. For classification, the node impurity is measured by the Gini-Index. In the $p = 10$ sub-figure, one can see that variables that carry the designation X1 to X3 are all included almost consecutively in the top 10. That means that by $p = 10$ features, the random forest recognizes the best variables to split on. For the $p = 290$ features sub-figure only six out of the nine relevant variables are included in the top 20 plot. In this case even the random forest is having a hard time finding the relevant variables to split on. Similar findings for the first and last runs are observed for gradient boosting in Figure 12.

7 Conclusion

This section presents the conclusion. First, important findings are discussed in relation to the research question raised in Section 1. Second, the validity and applicability of the results is discussed.

Important Findings

The results were overall satisfactory and confirmed the hypotheses. Further, they confirmed the conclusion from the literature that, by high dimensional data, applying ML methods for prediction outperforms classical methods. The logit model performs well with small p, however when p is increased starts to overfitt the data. Here come the machine learning methods in use, which tend to perform well when p is close to n. Specifically LASSO proved to be the best linear prediction method. Therefore an important finding is that linear data can be best predicted with a linear model.

A possible reason for the poorer than LASSO performance of the tree-based methods is that they had $p * j$ features to split on, while linear regression, logit and LASSO only p features to estimate. The model formula for the tree-based methods simply piled all the variables together. Figuring out which variable belongs to which alternative was left to the tree-based method to determine. In contrast the logit model transformed the data, structuring the features to the individual alternatives. For regression and LASSO the structuring was done manually by dividing the training sample for each alternative class and fitting the model on it. When coding the formula for the tree-based methods, it was assumed that the they could alone figure out which feature belongs to which alternative, therefore all variables were simply included on the right side of the equation. Therefore data pre-processing may be required. Section 8 discusses recommendations for further study.

Another explanation why random forest and gradient boosting were not better than LASSO as originally expected is that the data generating process is linear in nature. Tree-based methods perform especially well on data that have a non-liner underlying relationship.

Validity and Applicability of Results

The results are valid for the simulated data on which they were trained and tested. There are serious doubts that the simulated data resembles real world data considering its complexities. However other papers based on real world data such as the ones mentioned in Section 2, show substantial benefits of using machine learning methods for prediction. Therefore there is a case that the results are also valid in real world applications.

8 Outlook

In this section an outlook is given that covers collected ideas and recommendations for further research. The section also discusses how the findings contribute to knowledge in the field and why the conducted research is important.

Recommendations for Research and Practice

An obvious area for further research is to simulate data with a non-linear data generating process, then fit the same models on it. The expectation would be that, non-linear tree-based models would perform better than the linear LASSO model.

Another potential extension of the simulated data would be varying the number of relevant variables with the number of features. The simulation in this paper assumed only three relevant variables. Letting the number of relevant variables increase with the number of features would reduce sparsity and increase the performance of methods that don't practice variable selection.

Yet a further extension of the six models would be creating an combined ensemble model from the above fitted methods with weights, such was done by Bajari [1]. The goal would be to improve the prediction performance more than the best model did individually.

Another path for further research could be to simulate data that has qualities, which go beyond the estimation possibilities of a basic logit model. This would include simulating random taste variation, unrestricted substitution patterns and correlation between unobserved factors [36] (p. 50). The BLP-Literature [2] [25] used real world data, but it would be interesting to see if data with these qualities could be simulated. Such an exercise of making the data more suitable to a mixed logit model would potentially reduce the applicability of machine learning methods.

To test the validity of the results in this paper, the same models can be trained on real world data to see if the results are consistent.

In Section 5 it was observed that pre-processing of data could potentially lead to a better fit with the tree-based methods. For example re-structuring the data so tree-based methods are fit respectively on the features of each class, rather than on all features, would potentially ease prediction. This conclusion is similar to the conclusion of Paredes et. al. [27], who also mention that machine learning methods could perform better under the condition of pre-processing their inputs. To what extent this conclusion also applies in the case of the simulated data in this paper remains to be seen.

Another area of improvement considers the coding syntax of the project. Further work would allow the automation of the code through application of custom functions. Currently as the code is, there is a lot of repetition of the same tasks such as in-and out-of-sample predictions, confusion matrix building and accuracy calculation. These commands could be automated, which in turn would increase the clarity and replicability of the code.

How findings contribute to knowledge in the field

The findings underpin the importance of machine learning methods for prediction, which are already popular in academia and business. This paper adds to the body of knowledge of how to fit, tune and assess the performance of classical statistical and machine learning methods. It further shows pre-processing of data for some of the models. The research is important, because it practices modern techniques which are in high demand. The detailed and accessibly explained methodology makes the field to newcomers more reachable and the provided code serves as a reference for similar prediction exercises.

Appendix A: R Code

The next page presents the code used for the simulation of the data, fitting of the models and compiling of the results.

Code: Demand Estimation with Machine Learning

Georgi Zhelev

The code is presented and explained. Outputs are shown where appropriate. To save space and maintain clarity, selected results are shown at the end. For further details please see the executable r-data file, which is provided with the data storage device.

1 Model Assumptions

The first section of the code represents the model assumptions.

1.1 Model Characteristics

This line of code determines the number of cross-validation folds and could be changed to increase computational speed. The results were generated with 5 folds, which amounted to 2 minutes of run time. Cross-Validation is used for the tuning of ML-Methods.

```r
cv.folds<-5
```

The model assumptions are 10, 50, 150 and 290 features, p. The number of observations n is kept constant at 300. The parameter *cvfolds* was already assigned to 5.

```r
p<-c(10, 50, 150, 290)
n=300
cvfolds <- cv.folds
```

1.2 Labels

Next, the names of the four runs and alternatives are set up, which will later be used to label the rows and columns of the results-data-frames.

```r
name<-c(p[1], p[2], p[3], p[4])
names<-c("p=", "p=", "p=", "p=")
index<-paste(names, name, sep = "", collapse = NULL)
alt.names <- c("TF", "HB", "NC")
run.names <- c("run1", "run2","run3", "run4")
```

1.3 Create Storage for Loop Results

Storage data frames in the form of lists are created for the storage of results such as signal-to-noise ratios, accuracies, coefficients, confusion matrices, variable importance and tabulations of the y-variable.

```r
details <- list()
coeff.logit <- list()
details1 <- list()
details3 <- list()
details4 <- list()

coeff.reg <- list()
coeff.reg.TF<- list()
coeff.reg.HB<- list()
coeff.reg.NC<- list()

coeff.lasso <- list()
coeff.lasso.TF <- list()
coeff.lasso.HB <- list()
coeff.lasso.NC <- list()

cm.logit <- list()
cm.reg <- list()
cm.lasso <- list()
cm.tree <- list()
cm.rf <- list()
cm.boost <- list()

rf_plot <- list()
gbm_plot <- list()

summ_fit_sample <- list()
summ_test_sample <- list()
summ_fit_lin_sample <- list()
summ_test_lin_sample <- list()
```

1.4 Loop

Finally a for-loop is created to vary based on the assigned p's.

```
for (i in 1:length(p)) {
  p[i]<-p[i]
```

2 Simulation

2.1 Set Relevant Coefficients

Next the relevant coefficients are set to -5, 3 and 5. A vector *sparce* is created to count the number of non-relevant features. In the case of the first run *p=10*, which will serve as an example, are the non-relevant coefficients 7.

```
pr<--5    #price
qual<-3 #quanlity
pop<-5  #popularity
sparce<-p-(length(pr)+length(qual)+length(pop)) #count 0-betas for sparcity
sparce
```

```
## [1] 7
```

Then a beta vector is created with the 3 relevant coefficients, while the rest are set to 0.

```
set.seed(1)
beta2<- c(pr, qual, pop, rep(0, sparce))
beta2
```

```
##  [1] -5  3  5  0  0  0  0  0  0  0
```

2.2 Generate X-Variables

In the next step the price, quality and popularity averages are assigned to the three alternatives (TF, HB and NC). The alternatives are modeled to be dissimilar, therefore each has a different magnitude of price, quality and popularity. Below are the coefficient-means of the first alternative shown.

```
prices<-c(3.90, 3.50, 1.50)

mu1=c(rep(prices[1] , length(pr)), rep(5 , length(qual)), rep(2 , length(pop)),
      rep(0, sparce)) #TF

mu2=c(rep(prices[2] , length(pr)), rep(1 , length(qual)), rep(4 , length(pop)),
      rep(0, sparce)) #HB
mu3=c(rep(prices[3] , length(pr)), rep(1 , length(qual)), rep(2 , length(pop)),
      rep(0, sparce)) #NC

round(mu1, 1)
```

```
##  [1] 3.9 5.0 2.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

Next the covariance structure of the x's is created, which is based on a diagonal matrix. Therefore the x's are not dependent on each other.

```
set.seed(1)
covar=diag(0.2, p, p)
head(covar)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]  0.2  0.0  0.0  0.0  0.0  0.0    0    0    0     0
## [2,]  0.0  0.2  0.0  0.0  0.0  0.0    0    0    0     0
## [3,]  0.0  0.0  0.2  0.0  0.0  0.0    0    0    0     0
## [4,]  0.0  0.0  0.0  0.2  0.0  0.0    0    0    0     0
## [5,]  0.0  0.0  0.0  0.0  0.2  0.0    0    0    0     0
## [6,]  0.0  0.0  0.0  0.0  0.0  0.2    0    0    0     0
```

After that the x's are created using the covariance structure and the means.

```
X1=rmvnorm(n, mean=mu1, sigma=covar)
X2=rmvnorm(n, mean=mu2, sigma=covar)
X3=rmvnorm(n, mean=mu3, sigma=covar)
str(X1)
```

```
##  num [1:300, 1:10] 3.62 4.58 4.31 4.51 3.83 ...
```

The created X-matrices are 300 observations long and 10 observations wide.

Finally the x's are packed into a matrix for each alternative and are labeled.

```
x1<-matrix(X1, nrow = n, ncol = p)
colnames(x1)<-paste(as.character(1:p), rep(".TF", p), sep = "")
x2<-matrix(X2, nrow = n, ncol = p)
colnames(x2)<-paste(as.character(1:p), rep(".HB", p), sep = "")
x3<-matrix(X3, nrow = n, ncol = p)
colnames(x3)<-paste(as.character(1:p), rep(".NC", p), sep = "")
```

2.3 Calculate Utility

To obtain representative utility, matrix multiplication is done and the X-matrices are transposed.

```
V1<-beta2%*%t(x1)     #1st product
V2<-beta2%*%t(x2)     #2nd product
V3<-beta2%*%t(x3)     #3rd product
```

The resulting representative utilities are bound into a single matrix.

```
V<-cbind(t(V1), t(V2), t(V3))
summary(V)
```

```
##        V1                V2                V3
##  Min.   :-6.176   Min.   :-3.674   Min.   :-3.395
##  1st Qu.: 3.288   1st Qu.: 3.197   1st Qu.: 3.805
##  Median : 5.508   Median : 6.087   Median : 5.802
##  Mean   : 5.759   Mean   : 5.718   Mean   : 5.749
##  3rd Qu.: 8.054   3rd Qu.: 7.852   3rd Qu.: 7.943
##  Max.   :15.053   Max.   :16.076   Max.   :13.755
```

```
str(V)
```

```
##  num [1:300, 1:3] 5.28 1.25 4.66 3.19 7.09 ...
```

2.4 Generate Error Term

Next, the error term is generated using the package for generating extreme values.

```
#install.packages("evd")
library(evd)
set.seed(1)
e1<-rgumbel(n*ncol(V), loc=0, scale=1.72)
e1<-matrix(e1,n,ncol(V),byrow=FALSE)
str(e1)
```

```
##  num [1:300, 1:3] 0.483 -0.287 3.313 3.384 1.428 ...
```

The model is completed by adding the unobserved (random) values of utility.

```
z1<-V+e1
```

2.5 Investigate Signal-to-Noise Ratio

```
signal<-mean(var(V))
noise<-mean(var(e1))
noise
```

```
## [1] 1.64465
```

The noise is according to logit-model assumptions.

```
snr<-mean(var(V))/mean(var(e1))
snr
```

```
## [1] 2.236826
```

The signal-to-noise is above 2.

2.6 Calculate Logit Probabilities

```
pr1 = exp(z1)/apply(exp(z1), drop=F, 1, sum)
```

The probabilities have the necessary qualities. They are between 0 and 1.

```
specify_decimal <- function(x, k) trimws(format(round(x, k), nsmall=k))
head(specify_decimal(pr1, 2))
```

```
##      [,1]   [,2]   [,3]  
## [1,] "0.05" "0.95" "0.00"
## [2,] "0.01" "0.00" "0.98"
## [3,] "0.23" "0.32" "0.45"
## [4,] "0.40" "0.00" "0.60"
## [5,] "0.95" "0.05" "0.00"
## [6,] "0.00" "0.96" "0.04"
```

They sum to 1.

```
head(apply(pr1, drop=F, 1, sum))
```

```
## [1] 1 1 1 1 1 1
```

Next using the *apply* command, the *rmultinom* function is applied on in the previous step generated probability matrix. One random vector is drawn for each row of the probability matrix. Using the generated probabilities one object is chosen. As seen in the output, where the probability is highest, is the chance higher for that column to take on the value of 1.

```
mChoices1 = t(apply(pr1, 1, rmultinom, n = 1, size = 1))
head(mChoices1)
```

```
##      [,1] [,2] [,3]
## [1,]    0    1    0
## [2,]    0    0    1
## [3,]    0    1    0
## [4,]    0    0    1
## [5,]    1    0    0
## [6,]    0    1    0
```

2.7 Generate Y-Variable

By replacing 1-s with respective column labels, a categorical ranking results with the options *y=1,2 or 3*.

```
y1 = apply(mChoices1, 1, function(x) which(x==1))
head(y1)
```

```
## [1] 2 3 2 3 1 2
```

```
length(y1)
```

```
## [1] 300
```

```
tabulate(y1)
```

```
## [1] 103 109  88
```

2.8 Safe Data

A data frame for not linear Methods: logit, tree, random forest and boosting is saved.

```
cereals<-data.frame(y1,x1,x2,x3)
str(cereals)
```

```
## 'data.frame':    300 obs. of  31 variables:
##  $ y1    : int  2 3 2 3 1 2 2 3 1 1 ...
##  $ X1.TF : num  3.62 4.58 4.31 4.51 3.83 ...
##  $ X2.TF : num  5.08 5.17 5.35 4.95 4.89 ...
##  $ X3.TF : num  1.63 1.72 2.03 2.17 2.31 ...
##  $ X4.TF : num  0.7134 -0.9904 -0.8897 -0.0241 0.2489 ...
##  $ X5.TF : num  0.147 0.503 0.277 -0.616 -0.308 ...
##  $ X6.TF : num  -0.3669 -0.0201 -0.0251 -0.1856 -0.3164 ...
##  $ X7.TF : num  0.21798 -0.00724 -0.06967 -0.17633 0.16305 ...
##  $ X8.TF : num  0.3302 0.4221 -0.6577 -0.0265 0.3437 ...
##  $ X9.TF : num  0.2575 0.3673 -0.2138 0.4919 -0.0502 ...
##  $ X10.TF: num  -0.137 0.266 0.187 0.341 0.394 ...
##  $ X1.HB : num  3.83 3.72 3.52 4.1 3.7 ...
##  $ X2.HB : num  1.173 1.192 1.69 0.206 0.804 ...
##  $ X3.HB : num  4.58 3.43 4.43 4.17 4.54 ...
##  $ X4.HB : num  -0.3594 1.0001 -0.8194 0.0188 -0.6364 ...
##  $ X5.HB : num  -0.7167 0.1485 0.0595 -0.0594 -0.0263 ...
##  $ X6.HB : num  0.41736 -0.06229 -0.46215 -0.00624 0.09111 ...
##  $ X7.HB : num  0.808 -0.329 -0.773 -0.164 0.332 ...
##  $ X8.HB : num  -0.0253 -1.2422 -0.5187 -0.047 -0.1689 ...
##  $ X9.HB : num  0.843 -0.144 -0.623 0.742 1.064 ...
##  $ X10.HB: num  0.706 -0.463 -0.448 0.695 -0.397 ...
##  $ X1.NC : num  1.22 1.04 1.52 1.12 2.18 ...
##  $ X2.NC : num  0.504 0.837 1.444 0.163 1.186 ...
##  $ X3.NC : num  1.03 1.38 1.26 2.33 1.67 ...
##  $ X4.NC : num  -0.014 -0.2395 0.0635 -0.1548 -0.7202 ...
##  $ X5.NC : num  -0.116 0.123 -0.301 -0.623 -0.232 ...
##  $ X6.NC : num  0.239 0.588 -0.645 -0.349 0.529 ...
##  $ X7.NC : num  -0.2502 -0.0762 0.3721 0.2759 0.0937 ...
##  $ X8.NC : num  0.7193 0.6471 0.1148 -0.1233 -0.0928 ...
##  $ X9.NC : num  0.249 0.737 -0.307 0.716 -0.498 ...
##  $ X10.NC: num  0.08301 0.45325 -0.00687 -0.06702 0.82171 ...
```

for linear Methods: regression and lasso, a second data frame is saved. The second data frame is the same as the first, except for that it has three additional probability variables (X1, X2 and X3).

```
cereals.lin<-data.frame(y1,pr1,x1,x2,x3)
names(cereals.lin)
```

```
##  [1] "y1"     "X1"     "X2"     "X3"     "X1.TF"  "X2.TF"  "X3.TF"
##  [8] "X4.TF"  "X5.TF"  "X6.TF"  "X7.TF"  "X8.TF"  "X9.TF"  "X10.TF"
## [15] "X1.HB"  "X2.HB"  "X3.HB"  "X4.HB"  "X5.HB"  "X6.HB"  "X7.HB"
## [22] "X8.HB"  "X9.HB"  "X10.HB" "X1.NC"  "X2.NC"  "X3.NC"  "X4.NC"
## [29] "X5.NC"  "X6.NC"  "X7.NC"  "X8.NC"  "X9.NC"  "X10.NC"
```

2.9 Assign Levels

Format the y-column in the data frame as a factor and call it “choice”.

```
choice<-factor(cereals$y1)
head(choice)
```

```
## [1] 2 3 2 3 1 2
## Levels: 1 2 3
```

Name the levels of the as-factor-formatted choice-column. 1 for Tiger Flakes (TF), 2 for Honey Bits (HB) and 3 for Nougat Crisps (NC).

```
levels<-c("TF", "HB", "NC")
levels(choice)<-levels
head(choice)
```

```
## [1] HB NC HB NC TF HB
## Levels: TF HB NC
```

Replace the “y” column with the “choice” column in the data frames.

```
cereals<-cbind(choice, cereals[,-1])
cereals.lin<-cbind(choice, cereals.lin[,-1])
```

3 Model Fitting and Tuning

The fitting of the six classical and machine learning models is presented next.

3.1 Fit Logit

3.1.1 Split Data

Find split point in the data.

```
r<-nrow(cereals)/2
r
```

```
## [1] 150
```

```
cereals_fit<-cereals[1:r,]
cereals_test<-cereals[-(1:r),]
nrow(cereals_fit)
```

```
## [1] 150
```

3.1.1.1 Means of Samples

means show that only relevant features (X1 to X3) have means higher than 0.

```
round(sapply(cereals_fit[,-1], mean), 2)
```

```
##   X1.TF  X2.TF  X3.TF  X4.TF  X5.TF  X6.TF  X7.TF  X8.TF  X9.TF X10.TF
##    3.86   5.08   2.01  -0.07   0.06  -0.03  -0.02  -0.03  -0.03   0.03
##   X1.HB  X2.HB  X3.HB  X4.HB  X5.HB  X6.HB  X7.HB  X8.HB  X9.HB X10.HB
##    3.49   1.00   4.05   0.00   0.01  -0.02  -0.02   0.08  -0.01  -0.05
##   X1.NC  X2.NC  X3.NC  X4.NC  X5.NC  X6.NC  X7.NC  X8.NC  X9.NC X10.NC
##    1.46   1.07   1.94  -0.02  -0.06  -0.02  -0.01   0.00   0.09  -0.01
```

Below is a tabulation of the selected alternatives in the train sample.

```
summ_cereals_fit<-summary(cereals_fit$choice)
summ_cereals_fit
```

```
## TF HB NC
## 51 57 42
```

```
summ_cereals_test<-summary(cereals_test$choice)
```

3.1.2 Transform Data into Mlogit Data Object

```r
library(mlogit)
Cereal_fit <- mlogit.data(cereals_fit, varying = c(2:ncol(cereals_fit)),
                          shape = "wide", choice = "choice")
Cereal_test <- mlogit.data(cereals_test, varying = c(2:ncol(cereals_test)),
                           shape = "wide", choice = "choice")
```

3.1.3 Set up Flexible Formula for Logit

```r
last<-ncol(Cereal_fit)-2
k<-colnames(Cereal_fit)[-1]
k<-k[-1]
k<-k[-last]
factors <- k
factors
```

```
##  [1] "X1"  "X2"  "X3"  "X4"  "X5"  "X6"  "X7"  "X8"  "X9"  "X10"
```

```r
X<-as.formula(paste("choice~", paste(factors, collapse="+")))
X
```

```
## choice ~ X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10
```

```r
f1 <- mFormula(X)
```

3.1.4 Remove Intercept

```r
f0 <- update(f1, .~.-1)
```

3.1.5 Fit Logit Model

```r
m<-mlogit(f0, data=Cereal_fit, seed = 1)
```

3.1.6 In-Sample Performance

Make predictions. The logit predictions are in the form of probabilities.

```r
library("caret")
set.seed(1)
logit.pred_in<-fitted(m, outcome = FALSE)
head(logit.pred_in)
```

```
##           HB         NC          TF
## 1 0.80802081 0.00895389 0.183025301
## 2 0.15875477 0.71489982 0.126345407
## 3 0.95266582 0.01520610 0.032128083
## 4 0.03575981 0.85312309 0.111117101
## 5 0.36271514 0.01108054 0.626204322
## 6 0.70555287 0.29111293 0.003334203
```

The column names are used to label the alternatives.

```r
y_hat = colnames(logit.pred_in)[apply(logit.pred_in, 1, which.max)]
head(y_hat)
```

```
## [1] "HB" "NC" "HB" "NC" "TF" "HB"
```

They are then compared to the observed alternatives from the train sample.

```
logit_in = confusionMatrix(cereals_fit$choice, as.factor(y_hat))
```

The accuracy is recorded the generated confusion matrix output.

```
logit_in[2]
```

```
## $table
##           Reference
## Prediction HB NC TF
##         HB 41  6 10
##         NC  9 31  2
##         TF  6  3 42
```

```
logit_in<-logit_in$overall[1]*100
logit_in
```

```
## Accuracy
##       76
```

```
logit_in<-logit_in[[1]]
logit_in
```

```
## [1] 76
```

3.1.7 Out-of-Sample Performance

Make predictions

```
logit.pred_out<-predict(m, newdata=Cereal_test)
```

3.1.7.1 Data-Check

Specify a decimal format for the predicted probabilities and reformat the predicted probabilities into a readable format.

```
specify_decimal <- function(x, k) trimws(format(round(x, k), nsmall=k))
head(specify_decimal(logit.pred_out, 2))
```

```
##   HB     NC     TF
## 1 "0.01" "0.62" "0.38"
## 2 "0.02" "0.43" "0.55"
## 3 "0.00" "0.00" "1.00"
## 4 "0.87" "0.11" "0.02"
## 5 "0.04" "0.89" "0.06"
## 6 "0.26" "0.03" "0.71"
```

Verify predicted probabilities add to 1

```
head(apply(logit.pred_out, drop=F, 1, sum))
```

```
## 1 2 3 4 5 6
## 1 1 1 1 1 1
```

Record accuracy.

```
y_hat = colnames(logit.pred_out)[apply(logit.pred_out, 1, which.max)]
head(y_hat)
```

```
## [1] "NC" "TF" "TF" "HB" "NC" "TF"
```

```
logit_out_cm = confusionMatrix(as.factor(y_hat), cereals_test$choice)
logit_out<-logit_out_cm$overall[1]*100
logit_out<-logit_out[[1]]
```

3.1.8 Save Results

```
logit<-round(cbind(logit_in, logit_out), 0)
logit
```

```
##      logit_in logit_out
## [1,]       76        78
```

3.2 Fit Linear Regression

3.2.1 Split Data

find split point in the data and split it.

```
r<-nrow(cereals.lin)/2
cereals_fit<-cereals.lin[1:r,]
cereals_test<-cereals.lin[-(1:r),]
names(cereals_fit)
```

```
##  [1] "choice" "X1"     "X2"     "X3"     "X1.TF"  "X2.TF"  "X3.TF"
##  [8] "X4.TF"  "X5.TF"  "X6.TF"  "X7.TF"  "X8.TF"  "X9.TF"  "X10.TF"
## [15] "X1.HB"  "X2.HB"  "X3.HB"  "X4.HB"  "X5.HB"  "X6.HB"  "X7.HB"
## [22] "X8.HB"  "X9.HB"  "X10.HB" "X1.NC"  "X2.NC"  "X3.NC"  "X4.NC"
## [29] "X5.NC"  "X6.NC"  "X7.NC"  "X8.NC"  "X9.NC"  "X10.NC"
```

Test sample includes logit probabilities.

record summary statistics of samples for later

```
summ_cereals_fit_lin<-summary(cereals_fit$choice)
summ_cereals_test_lin<-summary(cereals_test$choice)
```

3.2.2 Pre-Process Data

3.2.2.1 Separate Data for Regressions

Data frames are build using the logit probability and features per alternative.

```
cereals.TF<-cbind(cereals_fit$X1, cereals_fit[(1+4):(5+p-1)])
str(cereals.TF)
```

```
## 'data.frame':    150 obs. of  11 variables:
##  $ cereals_fit$X1: num  0.0477 0.0122 0.2296 0.4004 0.9462 ...
##  $ X1.TF         : num  3.62 4.58 4.31 4.51 3.83 ...
##  $ X2.TF         : num  5.08 5.17 5.35 4.95 4.89 ...
##  $ X3.TF         : num  1.63 1.72 2.03 2.17 2.31 ...
##  $ X4.TF         : num  0.7134 -0.9904 -0.8897 -0.0241 0.2489 ...
##  $ X5.TF         : num  0.147 0.503 0.277 -0.616 -0.308 ...
##  $ X6.TF         : num  -0.3669 -0.0201 -0.0251 -0.1856 -0.3164 ...
##  $ X7.TF         : num  0.21798 -0.00724 -0.06967 -0.17633 0.16305 ...
##  $ X8.TF         : num  0.3302 0.4221 -0.6577 -0.0265 0.3437 ...
##  $ X9.TF         : num  0.2575 0.3673 -0.2138 0.4919 -0.0502 ...
##  $ X10.TF        : num  -0.137 0.266 0.187 0.341 0.394 ...
```

```
cereals.HB<-cbind(cereals_fit$X2, cereals_fit[(5+p-1+1):(5+2*p-1)])
cereals.NC<-cbind(cereals_fit$X3, cereals_fit[(5+2*p-1+1):ncol(cereals_fit)])
```

Test Data has the same structure as the train data.

```
cereals.TF.test<-cbind(cereals_test$X1, cereals_test[(1+4):(5+p-1)])
cereals.HB.test<-cbind(cereals_test$X2, cereals_test[(5+p-1+1):(5+2*p-1)])
cereals.NC.test<-cbind(cereals_test$X3, cereals_test[(5+2*p-1+1):ncol(cereals_fit)])
```

3.2.3 Set up Formulas for 3 Regressions

```
k<-colnames(cereals.TF[2:ncol(cereals.TF)])
TF<-as.formula(paste("log(cereals.TF[,1])~", paste(k, collapse="+")))
TF
```

```
## log(cereals.TF[, 1]) ~ X1.TF + X2.TF + X3.TF + X4.TF + X5.TF +
##     X6.TF + X7.TF + X8.TF + X9.TF + X10.TF
```

```
k<-colnames(cereals.HB[2:ncol(cereals.HB)])
HB<-as.formula(paste("log(cereals.HB[,1])~", paste(k, collapse="+")))
HB
```

```
## log(cereals.HB[, 1]) ~ X1.HB + X2.HB + X3.HB + X4.HB + X5.HB +
##     X6.HB + X7.HB + X8.HB + X9.HB + X10.HB
```

```
k<-colnames(cereals.NC[2:ncol(cereals.NC)])
NC<-as.formula(paste("log(cereals.NC[,1])~", paste(k, collapse="+")))
NC
```

```
## log(cereals.NC[, 1]) ~ X1.NC + X2.NC + X3.NC + X4.NC + X5.NC +
##     X6.NC + X7.NC + X8.NC + X9.NC + X10.NC
```

3.2.3.1 Remove Intercept

```
form.TF <- update(TF, .~.-1)
form.HB <- update(HB, .~.-1)
form.NC <- update(NC, .~.-1)
```

3.2.4 Fit Regression Models

```
ols.TF <- lm(form.TF, data=cereals.TF)
ols.HB <- lm(form.HB, data=cereals.HB)
ols.NC <- lm(form.NC, data=cereals.NC)
```

3.2.5 In-Sample Predictions

The predictions of the three regressions are compared.

```
TF<-fitted(ols.TF, outcome = FALSE)
HB<-fitted(ols.HB, outcome = FALSE)
NC<-fitted(ols.NC, outcome = FALSE)
util<-cbind(TF, HB, NC)
head(util, 4)
```

```
##          TF        HB        NC
## 1 -2.848340 -2.811533 -4.560369
## 2 -5.846930 -5.872507 -2.778482
## 3 -5.301662 -2.086851 -4.101773
## 4 -5.686562 -6.390244 -2.695700
```

The alternative that has the highest utility is chosen.

```
wahl<-apply(util, 1, function(x) which.max(x))
head(wahl)
```

```
## 1 2 3 4 5 6
## 2 3 2 3 1 2
```

Column numbers serve as indicators for the alternatives.

3.2.6 In-Sample Correct

The predicted choices are compared to the numeric of the observed choices. The comparison is done in a confusion matrix.

```
y_hat<-as.numeric(wahl)
cm = table(wahl, cereals_fit$choice)
cm
```

```
##
## wahl TF HB NC
##    1 40  7  3
##    2  6 44  9
##    3  5  6 30
```

Accuracy is calculated

```
reg_in<-sum(diag(cm))/sum(cm)*100
reg_in
```

```
## [1] 76
```

3.2.7 Out-of-Sample Predictions

Make Predictions.

```
TF=predict(ols.TF, newdata =cereals.TF.test)
HB=predict(ols.HB, newdata =cereals.HB.test)
NC=predict(ols.NC, newdata =cereals.NC.test)
util<-cbind(TF, HB, NC)
```

3.2.8 Out-of-Sample Correct

```
wahl<-apply(util, 1, function(x) which.max(x))
reg_out_cm = table(wahl, cereals_test$choice)
rownames(reg_out_cm)<- alt.names
reg_out<-sum(diag(reg_out_cm))/sum(reg_out_cm)*100
```

3.2.9 Store Results

```
lin.reg<-round(cbind(reg_in, reg_out), 0)
lin.reg
```

```
##      reg_in reg_out
## [1,]     76      71
```

3.3 Fit LASSO

LASSO using glmnet package requires coding the y and x variables in separate matrices.

3.3.1 Code train and test samples for lasso.cv

3.3.1.1 Train

```
Y.train.TF = log(cereals.TF[, 1]) # output variable
str(Y.train.TF)
```

```
##  num [1:150] -3.0428 -4.4063 -1.4714 -0.9153 -0.0553 ...
```

```
X.train.TF = as.matrix(cereals.TF[,2:ncol(cereals.TF)]) # regressors
str(X.train.TF)
```

```
##  num [1:150, 1:10] 3.62 4.58 4.31 4.51 3.83 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:150] "1" "2" "3" "4" ...
##   ..$ : chr [1:10] "X1.TF" "X2.TF" "X3.TF" "X4.TF" ...
```

```
Y.train.HB = log(cereals.HB[, 1]) # output variable
X.train.HB = as.matrix(cereals.HB[,2:ncol(cereals.HB)]) # regressors
Y.train.NC = log(cereals.NC[, 1]) # output variable
X.train.NC = as.matrix(cereals.NC[,2:ncol(cereals.NC)]) # regressors
```

3.3.1.2 Test

```
Y.test.TF = log(cereals.TF.test[, 1]) # output variable
X.test.TF = as.matrix(cereals.TF.test[,2:ncol(cereals.TF)]) # regressors
Y.test.HB = log(cereals.HB.test[, 1]) # output variable
X.test.HB = as.matrix(cereals.HB.test[,2:ncol(cereals.HB)]) # regressors
Y.test.NC = log(cereals.NC.test[, 1]) # output variable
X.test.NC = as.matrix(cereals.NC.test[,2:ncol(cereals.NC)]) # regressors
```

3.3.2 Do Cross-Validation

```
library(glmnet)
lasso.cv.TF=cv.glmnet(X.train.TF, Y.train.TF, family="gaussian",alpha=1)
lasso.cv.HB=cv.glmnet(X.train.HB, Y.train.HB, family="gaussian",alpha=1)
lasso.cv.NC=cv.glmnet(X.train.NC, Y.train.NC, family="gaussian",alpha=1)
```

3.3.3 Determine Best Lambda, Fit LASSO, Predict lasso.cv In-Sample

3.3.3.1 TF

```
bestlam.lasso.cv = lasso.cv.TF$lambda.min
reg.la.TF=glmnet(X.train.TF,Y.train.TF, family="gaussian", alpha=1,
                 lambda = bestlam.lasso.cv)
TF = predict(reg.la.TF, s= bestlam.lasso.cv, newx=X.train.TF)
str(TF)
```

```
##  num [1:150, 1] -3.3 -5.6 -4.45 -5.16 -2.76 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:150] "1" "2" "3" "4" ...
##   ..$ : chr "1"
```

Utilities result from the prediction.

3.3.3.2 HB

```
bestlam.lasso.cv = lasso.cv.HB$lambda.min
reg.la.HB=glmnet(X.train.HB,Y.train.HB, family="gaussian", alpha=1,
                 lambda = bestlam.lasso.cv)
HB = predict(reg.la.HB, s= bestlam.lasso.cv, newx=X.train.HB,
             lambda = bestlam.lasso.cv)
```

3.3.3.3 NC

```
bestlam.lasso.cv = lasso.cv.NC$lambda.min
reg.la.NC=glmnet(X.train.NC,Y.train.NC, family="gaussian", alpha=1,
                 lambda = bestlam.lasso.cv)
NC = predict(reg.la.NC, s= bestlam.lasso.cv, newx=X.train.NC)
```

3.3.4 Post Lasso Coefficients

3.3.4.1 Variable Selection and Liner Model

```
W <- as.matrix(coef(reg.la.TF))
W
```

TF

```
##                      s0
## (Intercept) -7.96959386
## X1.TF       -2.89974525
## X2.TF        2.20885371
## X3.TF        2.28251390
## X4.TF        0.00000000
## X5.TF        0.00000000
## X6.TF        0.00000000
## X7.TF        0.05462683
## X8.TF        0.65283385
## X9.TF        0.00000000
## X10.TF       0.00000000
```

```
keep_X <- rownames(W)[W!=0]
keep_X <- keep_X[!keep_X == "(Intercept)"] #remove intercept
keep_X
```

```
## [1] "X1.TF" "X2.TF" "X3.TF" "X7.TF" "X8.TF"
```

```
x<- X.train.TF[,keep_X]
summ.lasso.TF<-lm(Y.train.TF~x -1)
#stargazer(summ.lasso.TF, type = "latex")
```

```
W <- as.matrix(coef(reg.la.HB))
keep_X <- rownames(W)[W!=0]                 #only keep non-zero coefficients
keep_X <- keep_X[!keep_X == "(Intercept)"]  #don't selected intercept
x<- X.train.HB[,keep_X]                     #only use selected features
summ.lasso.HB<-lm(Y.train.HB~x -1)          #remove intercept in new model
```

HB

```
W <- as.matrix(coef(reg.la.NC))
keep_X <- rownames(W)[W!=0]
keep_X <- keep_X[!keep_X == "(Intercept)"]
x<- X.train.NC[,keep_X]
summ.lasso.NC<-lm(Y.train.NC~x -1)
```

NC

3.3.5 Compare In-Sample Predictions and choose largest utility

```
util<-cbind(TF, HB, NC)
colnames(util)<-levels
wahl<-apply(util, 1, function(x) which.max(x))
```

3.3.6 In-Sample Accuracy

```
cm = table(wahl, cereals_fit$choice)
lasso_in<-sum(diag(cm))/sum(cm)*100
```

3.3.7 Out-of-Sample Prediction

```
TF = predict(reg.la.TF, s= bestlam.lasso.cv, newx=X.test.TF)
HB = predict(reg.la.HB, s= bestlam.lasso.cv, newx=X.test.HB)
NC = predict(reg.la.NC, s= bestlam.lasso.cv, newx=X.test.NC)
```

3.3.8 Compare Out-of-Sample Predictions and choose highest

```
util<-cbind(TF, HB, NC)
wahl<-apply(util, 1, function(x) which.max(x))
```

3.3.9 Out-of-Sample Accuracy

```
lasso_out_cm = table(wahl, cereals_test$choice)
rownames(lasso_out_cm)<-alt.names
lasso_out<-sum(diag(lasso_out_cm))/sum(lasso_out_cm)*100
```

3.3.10 Save Results

```
lasso.cv<-round(cbind(lasso_in, lasso_out), 0)
lasso.cv
```

```
##      lasso_in lasso_out
## [1,]       72        77
```

3.4 Decision Tree

3.4.1 Split Data

This is the same split as in logit. But hast to be done again because variable names overlap.

```
r<-nrow(cereals)/2
cereals_fit<-cereals[1:r,]
cereals_test<-cereals[-(1:r),]
```

3.4.2 Tuning

The tree is first fit with the default cp of 0.001.

```
library(rpart)
library(rpart.plot)
tree <- rpart(choice ~ ., data = cereals_fit, method = "class", cp= 0.001)
```

3.4.3 Prune Tree

The optimal *cp* displyed below by the *printcp()* function for the first run **does not** match the chosen *cp* in the code below. The chosen *cp* level however yields good overal results and remedies the poor classification of the fourth run, if the minimal *cp* were to be used.

```
printcp(tree)
```

```
##
## Classification tree:
## rpart(formula = choice ~ ., data = cereals_fit, method = "class",
##     cp = 0.001)
##
## Variables actually used in tree construction:
## [1] X1.TF X2.TF X3.HB X3.NC X3.TF X5.HB X7.TF
##
## Root node error: 93/150 = 0.62
##
## n= 150
##
##         CP nsplit rel error  xerror     xstd
## 1 0.193548      0   1.00000 1.06452 0.062384
## 2 0.107527      1   0.80645 0.93548 0.064998
## 3 0.080645      2   0.69892 0.97849 0.064331
## 4 0.053763      4   0.53763 0.92473 0.065134
## 5 0.032258      5   0.48387 0.84946 0.065753
## 6 0.021505      6   0.45161 0.80645 0.065846
## 7 0.001000      7   0.43011 0.83871 0.065794
```

```
tree <- rpart(choice ~ ., data = cereals_fit, method = "class", cp= 0.06451613)
```

3.4.4 In-Sample Tree Fit

Probabilities for each class are predicted

```
fit.tree<-predict(tree, newdata=cereals_fit)
head(fit.tree, 4)
```

```
##           TF        HB        NC
## 1 0.24324324 0.5135135 0.2432432
## 2 0.20000000 0.5600000 0.2400000
## 3 0.20000000 0.5600000 0.2400000
## 4 0.08695652 0.1304348 0.7826087
```

Choose alternative that has the maximum utility. Compare to observed choices and calculate accuracy from confusion matrix.

```
fit.tree<-apply(fit.tree, 1, function(x) which.max(x))
head(fit.tree)
```

```
## 1 2 3 4 5 6
## 2 2 2 3 1 3
```

```
cm = table(fit.tree, cereals_fit$choice)
tree_in<-sum(diag(cm))/sum(cm)*100
```

3.4.5 Out-of-Sample Tree Fit and Save Results

```
tree.pred=predict(tree, newdata=cereals_test)
fit.tree<-apply(tree.pred, 1, function(x) which.max(x))
tree_out_cm = table(fit.tree, cereals_test$choice)
rownames(tree_out_cm)<-alt.names
tree_out<-sum(diag(tree_out_cm))/sum(tree_out_cm)*100

tree<-round(cbind(tree_in, tree_out), 0)
tree
```

```
##      tree_in tree_out
## [1,]      67       49
```

3.5 Random Forest

3.5.1 Manual Tuning (commented out)

```
library(randomForest)
set.seed(1)
#bag1 = randomForest(choice ~ ., data=cereals_fit, mtry = round(sqrt(p),0),
#ntree=500)
#bag1 = randomForest(choice ~ ., data=cereals_fit, mtry =23, ntree=500,
#importance=TRUE, proximity=TRUE)
#bag1 = randomForest(choice ~ ., data=cereals_fit, importance=TRUE,
#proximity=TRUE)
```

3.5.2 Automatic Tuning

The caret package divides the train data set randomly into folds and trains model on them, then averages the obtained error terms. This type of fitting is better than the manual fitting.

```
library("caret")
tc = trainControl(method = "repeatedcv", number = cvfolds)
bag1 = train(choice ~., data=cereals_fit, method="rf", trControl=tc)
```

3.5.3 Analyse Fit

If enabled, *print(bag1)* shows how the parameters were tuned. It is disabled for better clarity and to reduce space. Please see the provided executable r data file for more details.

```
#print(bag1)
rf_var_plot <- varImp(bag1, top = 20)   #only to be used with bag1 from caret package
```

3.5.4 In-Sample Fit

```
pred.bag1 = predict(bag1, newdata =cereals_fit)
head(pred.bag1)      #predictions are in the form of categories
```

```
## [1] HB NC HB NC TF HB
## Levels: TF HB NC
```

```
cm = table(pred.bag1, cereals_fit$choice)
rf_in<-sum(diag(cm))/sum(cm)*100
```

3.5.5 Out-of-Sample Fit and Save Results

```
pred.bag1 = predict(bag1, newdata =cereals_test)
rf_out_cm = table(pred.bag1, cereals_test$choice)
rf_out<-sum(diag(rf_out_cm))/sum(rf_out_cm)*100

rf<-round(cbind(rf_in, rf_out), 0)
rf
```

```
##      rf_in rf_out
## [1,]   100     63
```

3.6 Gradient Boosting

3.6.1 Manual tuning

The commented out line shows the manual tuning method with the *gbm* package. This method was not used in favor of the automatic tuning.

```
library("gbm")
set.seed(1)
#boost1=gbm(choice ~ ., data=cereals_fit, distribution= "multinomial")
```

3.6.2 Automatic Tuning

The automatic tuning is done using the *caret* package.

```
tc = trainControl(method = "repeatedcv", number = cvfolds)
garbage <- capture.output(boost1 <- train(choice ~., data=cereals_fit, method="gbm",
                                          trControl=tc))
```

3.6.3 Analyse Fit

If enabled, *boost1* shows how the parameters were tuned. It is disabled for better clarity and to reduce space. Please see the provided executable r data file for more details.

```
#boost1
var_imp_gbm<-plot(varImp(boost1))
```

3.6.4 In-Sample Fit

```
pred.boost1=predict(boost1, newdata =cereals_fit)
cm = table(pred.boost1, cereals_fit$choice)
boost_in<-sum(diag(cm))/sum(cm)*100
```

3.6.5 Out-of-Sample Fit

```
pred.boost1=predict(boost1, newdata =cereals_test)
boost_out_cm = table(pred.boost1, cereals_test$choice)
rownames(boost_out_cm) <- alt.names
boost_out<-sum(diag(boost_out_cm))/sum(boost_out_cm)*100
```

3.6.6 Save Results

```
boost<-round(cbind(boost_in, boost_out), 0)
boost
```

```
##      boost_in boost_out
## [1,]       88        61
```

4 Save Estimated Coefficients

##Logit

```
coef<-c(coefficients(m)[1],
        coefficients(m)[2],
        coefficients(m)[3])
```

##Linear Regression

```
coef.reg<-c(coefficients(ols.TF)[1],
            coefficients(ols.TF)[2],
            coefficients(ols.TF)[3],
            coefficients(ols.HB)[1],
            coefficients(ols.HB)[2],
            coefficients(ols.HB)[3],
            coefficients(ols.NC)[1],
            coefficients(ols.NC)[2],
            coefficients(ols.NC)[3])
```

4.1 LASSO

```
coef.lasso<-c(coefficients(summ.lasso.TF)[1],
            coefficients(summ.lasso.TF)[2],
            coefficients(summ.lasso.TF)[3],
            coefficients(summ.lasso.HB)[1],
            coefficients(summ.lasso.HB)[2],
            coefficients(summ.lasso.HB)[3],
            coefficients(summ.lasso.NC)[1],
            coefficients(summ.lasso.NC)[2],
            coefficients(summ.lasso.NC)[3])
```

5 Save Results from Loop

```
details[[i]]<-cbind(coef)
coeff.logit[[i]]<-m

details1[[i]]<-cbind(logit,lin.reg,lasso.cv,tree,rf,boost)
```

```r
details3[[i]]<-cbind(snr, noise)

details4[[i]]<-cbind(sum(rf_in), sum(rf_out))

coeff.reg[[i]] <-cbind(coef.reg)
coeff.reg.TF[[i]] <-ols.TF
coeff.reg.HB[[i]] <-ols.HB
coeff.reg.NC[[i]] <-ols.NC

coeff.lasso[[i]] <- cbind(coef.lasso)
coeff.lasso.TF[[i]] <- summ.lasso.TF
coeff.lasso.HB[[i]] <- summ.lasso.HB
coeff.lasso.NC[[i]] <- summ.lasso.NC

cm.logit[[i]] <- logit_out_cm[2]
cm.reg[[i]] <- reg_out_cm
cm.lasso[[i]] <- lasso_out_cm
cm.tree[[i]] <- tree_out_cm
cm.rf[[i]] <- rf_out_cm
cm.boost[[i]] <- boost_out_cm

rf_plot[[i]] <- rf_var_plot
gbm_plot[[i]] <- var_imp_gbm

summ_fit_sample[[i]] <- summ_cereals_fit
summ_test_sample[[i]] <-summ_cereals_test

summ_fit_lin_sample[[i]] <-summ_cereals_fit_lin
summ_test_lin_sample[[i]] <-summ_cereals_test_lin

} #end Loop
```

6 Prepare Results Tables

This chunk includes the preparation of the results using various data frames and row and column naming. The preparation is important for presentation purposes in the paper.

```r
suppressMessages(library(stargazer))
suppressMessages(library(xtable))
```

6.1 Performance Table

```r
details2<-details1
details2<-unlist(details2, recursive = TRUE, use.names = TRUE)

run1<-round(matrix(details2[1:12], ncol = 2, byrow = TRUE), 0)
run2<-round(matrix(details2[13:24], ncol = 2, byrow = TRUE), 0)
run3<-round(matrix(details2[25:36], ncol =2, byrow = TRUE),0)
run4<-round(matrix(details2[37:48], ncol =2, byrow = TRUE), 0)

names1<-c("in sample", "out of sample")
names2<-c("logit", "lin reg", "lasso.cv", "tree", "random forrest", "boosting")
```

```r
colnames(run1)<-names1
rownames(run1)<-names2
colnames(run2)<-names1
rownames(run2)<-names2
colnames(run3)<-names1
rownames(run3)<-names2
colnames(run4)<-names1
rownames(run4)<-names2

run1.table<-xtable(run1, caption = index[1], digits = 0)
run2.table<-xtable(run2, caption = index[2], digits = 0)
run3.table<-xtable(run3, caption = index[3], digits = 0)
run4.table<-xtable(run4, caption = index[4], digits = 0)
```

6.2 Prepare Coefficients Tables

6.2.1 Logit

```r
coeff<-t(matrix(unlist(details), ncol = 3, byrow = TRUE))
colnames(coeff)<-index
rownames(coeff)<-c("price","quality","popularity")
coeff.table<-xtable(coeff, caption = "Coefficients")
```

6.2.2 Liner Regression

```r
coeff.reg2<-t(matrix(unlist(coeff.reg), ncol = 9, byrow = TRUE))
colnames(coeff.reg2)<-index
rownames(coeff.reg2)<-c("X1.TF", "X2.TF", "X3.TF", "X1.HB", "X2.HB", "X3.HB",
                        "X1.NC", "X2.NC", "X3.NC")
coeff.table.reg<-xtable(coeff.reg2, caption = "Coefficients")
```

6.2.3 LASSO

```r
coef.lasso<-t(matrix(unlist(coeff.lasso), ncol = 9, byrow = TRUE))
colnames(coef.lasso)<-index
rownames(coef.lasso)<-c("X1.TF", "X2.TF", "X3.TF", "X1.HB", "X2.HB", "X3.HB",
                        "X1.NC", "X2.NC", "X3.NC")
coeff.table.lasso<-xtable(coef.lasso, caption = "Coefficients")
```

6.3 Signal-to-Noise Table

```r
details.snr<-unlist(details3)
details.snr<-matrix(details.snr, ncol = 2, byrow = TRUE)
colnames(details.snr)<-c("Signal-to-Noise", "Noise")
rownames(details.snr)<-index
snr.table<-xtable(details.snr, camption= "Signal-to-Noise")
```

7 Show Results

This section presents the code for the results. Only the performance comparision is presented to save space. The results match the *Results* Section of the paper. For further details please see the provided r-data file.

7.1 Signal-to-Noise Ratio

```
details.snr
```

```
##       Signal-to-Noise   Noise
## p=10         2.236826 1.64465
## p=50         3.017325 1.64465
## p=150        2.425730 1.64465
## p=290        2.836494 1.64465
```

7.2 Performance Comparison

```
index
```

```
## [1] "p=10"  "p=50"  "p=150" "p=290"
```

```
run1
```

```
##                 in sample out of sample
## logit                  76            78
## lin reg                76            71
## lasso.cv               72            77
## tree                   67            49
## random forrest        100            63
## boosting               88            61
```

```
run2
```

```
##                 in sample out of sample
## logit                  77            62
## lin reg                72            45
## lasso.cv               62            54
## tree                   58            41
## random forrest        100            45
## boosting              100            45
```

```
run3
```

```
##                 in sample out of sample
## logit                 100            41
## lin reg                80            33
## lasso.cv               70            65
## tree                   69            47
## random forrest        100            51
## boosting              100            49
```

```
run4
```

```
##                 in sample out of sample
## logit                 100            36
## lin reg                84            29
## lasso.cv               74            64
## tree                   65            47
## random forrest        100            50
## boosting              100            47
```

7.3 Coefficients

Logit coefficients are listed below. First are the relevant coefficients for all runs listed. Then next output lists all estimated logit coefficients for the first run.

7.4 Logit

```
coeff
```

```
##                 p=10      p=50     p=150      p=290
## price      -3.096471 -3.354573 -28.90478 -12.353016
## quality     1.851797  1.918786  16.58122   6.582303
## popularity  3.212596  3.314025  31.08082  11.217060
```

```
summary(coeff.logit[[1]])
```

```
##
## Call:
## mlogit(formula = choice ~ X1 + X2 + X3 + X4 + X5 + X6 + X7 +
##     X8 + X9 + X10 - 1, data = Cereal_fit, seed = 1, method = "nr")
##
## Frequencies of alternatives:
##   HB   NC   TF
## 0.38 0.28 0.34
##
## nr method
## 6 iterations, 0h:0m:0s
## g'(-H)^-1g = 0.000249
## successive function values within tolerance limits
##
## Coefficients :
##       Estimate Std. Error z-value  Pr(>|z|)
## X1  -3.0964715  0.4140316 -7.4788 7.505e-14 ***
## X2   1.8517973  0.2478674  7.4709 7.971e-14 ***
## X3   3.2125959  0.4196564  7.6553 1.932e-14 ***
## X4  -0.0039057  0.3015569 -0.0130    0.9897
## X5  -0.0257784  0.2895766 -0.0890    0.9291
## X6  -0.3328288  0.3096260 -1.0749    0.2824
## X7   0.0014206  0.3040154  0.0047    0.9963
## X8   0.2248430  0.2962448  0.7590    0.4479
## X9  -0.0603981  0.3093601 -0.1952    0.8452
## X10  0.2815895  0.3250167  0.8664    0.3863
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log-Likelihood: -91.375
```

7.5 Linear Regression

Linear Regression coefficients are listed. First the relevant, then the estimated coefficients per alternative for the first run.

```
coeff.reg2
```

```
##             p=10       p=50      p=150       p=290
## X1.TF -3.9066881 -4.3611262 -10.056788 -677.756368
## X2.TF  1.4512183  1.5410204   4.598484  634.968445
## X3.TF  2.1921421  2.9160312   6.152307 -386.862818
## X1.HB -4.3510830 -2.7276638   5.957411  -40.995783
## X2.HB  1.0891153  1.3686861  -8.140303  -49.804010
## X3.HB  2.6882814  1.1111883  -4.574937   43.408298
## X1.NC -4.8601217 -4.5298688   4.287643   -3.286762
## X2.NC  1.4941079  1.0705718  -6.531668   -7.961015
## X3.NC  0.9540236  0.7777074 -10.656929    4.203862
```

```
summary(coeff.reg.TF[[1]])
```

```
##
## Call:
## lm(formula = form.TF, data = cereals.TF)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -9.889 -1.379  0.512  1.771  5.318
##
## Coefficients:
##        Estimate Std. Error t value Pr(>|t|)
## X1.TF  -3.90669    0.43701  -8.940 2.06e-15 ***
## X2.TF   1.45122    0.33132   4.380 2.31e-05 ***
## X3.TF   2.19214    0.52916   4.143 5.91e-05 ***
## X4.TF  -0.18126    0.55436  -0.327   0.7442
## X5.TF   0.03678    0.48089   0.076   0.9391
## X6.TF   0.06824    0.50473   0.135   0.8926
## X7.TF   0.22404    0.55816   0.401   0.6887
## X8.TF   1.18130    0.55561   2.126   0.0352 *
## X9.TF   0.20141    0.52722   0.382   0.7030
## X10.TF -0.08117    0.58449  -0.139   0.8898
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.911 on 140 degrees of freedom
## Multiple R-squared:  0.6675, Adjusted R-squared:  0.6437
## F-statistic:  28.1 on 10 and 140 DF,  p-value: < 2.2e-16
```

```
summary(coeff.reg.HB[[1]])
```

```
##
## Call:
## lm(formula = form.HB, data = cereals.HB)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.2658  -1.3238   0.3358   1.8578   6.2953
##
## Coefficients:
##         Estimate Std. Error t value Pr(>|t|)
## X1.HB  -4.351083   0.436698  -9.964  < 2e-16 ***
## X2.HB   1.089115   0.547734   1.988   0.0487 *
## X3.HB   2.688281   0.375667   7.156 4.26e-11 ***
## X4.HB  -0.025502   0.541557  -0.047   0.9625
## X5.HB   0.461733   0.566486   0.815   0.4164
## X6.HB   0.461519   0.552594   0.835   0.4050
## X7.HB   0.334463   0.568390   0.588   0.5572
## X8.HB   0.007566   0.548774   0.014   0.9890
## X9.HB  -0.022064   0.563030  -0.039   0.9688
## X10.HB  0.204218   0.618313   0.330   0.7417
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.077 on 140 degrees of freedom
## Multiple R-squared:  0.6443, Adjusted R-squared:  0.6189
## F-statistic: 25.36 on 10 and 140 DF,  p-value: < 2.2e-16
```

```
summary(coeff.reg.NC[[1]])
```

```
##
## Call:
## lm(formula = form.NC, data = cereals.NC)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.0819 -1.7971  0.0966  1.6785  5.6364
##
## Coefficients:
##        Estimate Std. Error t value Pr(>|t|)
## X1.NC  -4.86012    0.45618 -10.654   <2e-16 ***
## X2.NC   1.49411    0.46432   3.218   0.0016 **
## X3.NC   0.95402    0.36686   2.601   0.0103 *
## X4.NC  -0.28330    0.57875  -0.490   0.6252
## X5.NC  -0.04915    0.52389  -0.094   0.9254
## X6.NC   0.30359    0.51397   0.591   0.5557
## X7.NC   0.54486    0.48957   1.113   0.2676
## X8.NC  -0.34925    0.47651  -0.733   0.4648
## X9.NC   0.07644    0.53379   0.143   0.8863
## X10.NC -0.76902    0.51780  -1.485   0.1397
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.803 on 140 degrees of freedom
## Multiple R-squared:  0.7201, Adjusted R-squared:  0.7001
## F-statistic: 36.01 on 10 and 140 DF,  p-value: < 2.2e-16
```

7.6 LASSO

LASSO coefficients are listed. First the relevant, then the estimated coefficients for the first run per alternative.

```
 coef.lasso
```

```
##             p=10       p=50      p=150      p=290
## X1.TF -3.9012652 -3.8703536 -4.1970812 -3.5156274
## X2.TF  1.4647826  0.7921983  1.4707380  0.7801829
## X3.TF  2.1496418  3.6677306  2.7068046  3.0942549
## X1.HB -4.3478177 -2.7278889 -3.5658226 -3.2054035
## X2.HB  1.1806458  0.8591900  0.9522660  1.6118244
## X3.HB  2.6579528  1.3227454  2.0299172  1.6411899
## X1.NC -4.8652362 -3.9992464 -5.9801915 -4.2136059
## X2.NC  1.4939412  1.0644524  0.9997719  0.8643321
## X3.NC  0.9650781  1.2361749  2.1236241  0.8196607
```

```r
summary(coeff.lasso.TF[[1]])
```

```
##
## Call:
## lm(formula = Y.train.TF ~ x - 1)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -9.699 -1.429  0.451  1.817  5.179
##
## Coefficients:
##        Estimate Std. Error t value Pr(>|t|)
## xX1.TF  -3.9013     0.4266  -9.146 4.95e-16 ***
## xX2.TF   1.4648     0.3213   4.559 1.09e-05 ***
## xX3.TF   2.1496     0.5111   4.206 4.53e-05 ***
## xX7.TF   0.1994     0.5368   0.371   0.7109
## xX8.TF   1.1627     0.5361   2.169   0.0317 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.863 on 145 degrees of freedom
## Multiple R-squared:  0.6668, Adjusted R-squared:  0.6553
## F-statistic: 58.03 on 5 and 145 DF,  p-value: < 2.2e-16
```

```r
summary(coeff.lasso.HB[[1]])
```

```
##
## Call:
## lm(formula = Y.train.HB ~ x - 1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.6714  -1.3513   0.1692   1.9714   6.3899
##
## Coefficients:
##        Estimate Std. Error t value Pr(>|t|)
## xX1.HB  -4.3478     0.4224 -10.292  < 2e-16 ***
## xX2.HB   1.1806     0.5253   2.248   0.0261 *
## xX3.HB   2.6580     0.3584   7.417 9.04e-12 ***
## xX5.HB   0.4698     0.5531   0.849   0.3971
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.026 on 146 degrees of freedom
## Multiple R-squared:  0.6414, Adjusted R-squared:  0.6315
## F-statistic: 65.27 on 4 and 146 DF,  p-value: < 2.2e-16
```

```r
summary(coeff.lasso.NC[[1]])
```

```
##
## Call:
## lm(formula = Y.train.NC ~ x - 1)
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -9.1309 -1.7957  0.1941  1.6694  5.7915
##
## Coefficients:
##         Estimate Std. Error t value Pr(>|t|)
## xX1.NC   -4.8652     0.4353 -11.176  < 2e-16 ***
## xX2.NC    1.4939     0.4519   3.306  0.00119 **
## xX3.NC    0.9651     0.3506   2.752  0.00667 **
## xX7.NC    0.5452     0.4739   1.150  0.25192
## xX10.NC  -0.8190     0.4958  -1.652  0.10076
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.764 on 145 degrees of freedom
## Multiple R-squared:  0.7181, Adjusted R-squared:  0.7083
## F-statistic: 73.86 on 5 and 145 DF,  p-value: < 2.2e-16
```

7.7 Confusion Matrices

The provided confusion matrices in the paper are shown below. Only the first one is shown for clarity.

```
cm.logit[[1]]
```

```
## $table
##           Reference
## Prediction TF HB NC
##         TF 40  4  3
##         HB  8 38  4
##         NC  4 10 39
```

```
# cm.logit[[4]]
# cm.reg[[1]]
# cm.reg[[4]]
# cm.lasso[[1]]
# cm.lasso[[4]]
# cm.tree[[1]]
# cm.tree[[4]]
# cm.rf[[1]]
# cm.rf[[4]]
# cm.boost[[1]]
# cm.boost[[4]]
```

7.8 Preparation of Y-Variable (Observed Choices)

```
sum.stat.fit<- t(cbind(summ_fit_sample[[1]],
summ_fit_sample[[2]],
summ_fit_sample[[3]],
summ_fit_sample[[4]]))
rownames(sum.stat.fit)<- run.names

sum.stat.test<- t(cbind(summ_test_sample[[1]],
summ_test_sample[[2]],
summ_test_sample[[3]],
summ_test_sample[[4]]))
rownames(sum.stat.test)<- run.names
```

```
sum.stat.fit.lin <- t(cbind(summ_fit_lin_sample[[1]],
summ_fit_lin_sample[[2]],
summ_fit_lin_sample[[3]],
summ_fit_lin_sample[[4]]))
rownames(sum.stat.fit.lin)<- run.names

sum.stat.test.lin <- t(cbind(summ_test_lin_sample[[1]],
summ_test_lin_sample[[2]],
summ_test_lin_sample[[3]],
summ_test_lin_sample[[4]]))
rownames(sum.stat.test.lin)<- run.names
```

7.8.1 Tabulation of the Y-Variable (Observed Choices)

The y-variable is tabulated in order to determine the distribution of the three classes inside the train and test samples. The *cereals* and *cereals.lin* data frames are the same in regards to the categorical y-variable.

```
  sum.stat.fit
```

```
##      TF HB NC
## run1 51 57 42
## run2 51 57 42
## run3 49 62 39
## run4 51 57 42
```

```
  sum.stat.fit.lin
```

```
##      TF HB NC
## run1 51 57 42
## run2 51 57 42
## run3 49 62 39
## run4 51 57 42
```

```
  sum.stat.test
```

```
##      TF HB NC
## run1 52 52 46
## run2 44 52 54
## run3 52 57 41
## run4 48 53 49
```

```
  sum.stat.test.lin
```

```
##      TF HB NC
## run1 52 52 46
## run2 44 52 54
## run3 52 57 41
## run4 48 53 49
```

Therefore they yield the same two train-samples and same two test-samples. This is to show that although there are two data frames, the data is comparable.

```
 sum.stat.fit==sum.stat.fit.lin
```

```
##        TF   HB   NC
## run1 TRUE TRUE TRUE
## run2 TRUE TRUE TRUE
## run3 TRUE TRUE TRUE
## run4 TRUE TRUE TRUE
```

```
sum.stat.test==sum.stat.test.lin
```

```
##        TF   HB   NC
## run1 TRUE TRUE TRUE
## run2 TRUE TRUE TRUE
## run3 TRUE TRUE TRUE
## run4 TRUE TRUE TRUE
```

7.9 ML Variable Importance Plots

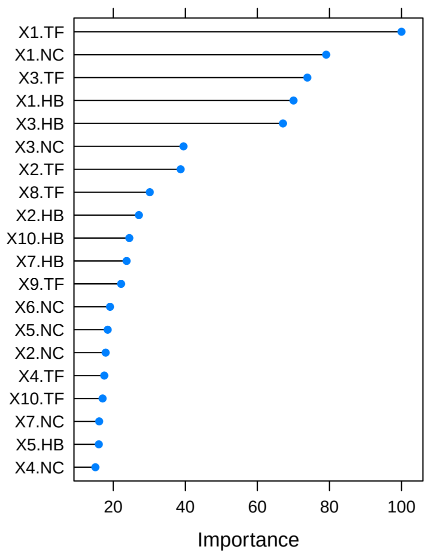
7.9.1 Tree Plot

The decison tree split plot is not provided, because the *prp()* output could not be saved in a data frame.

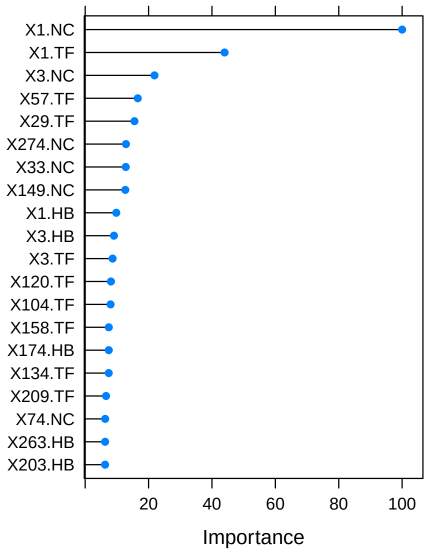
7.9.2 Random Forest

First the variable importance of the random forest is presented for the first and last run, such as in the paper.

```
plot(rf_plot[[1]], top = 20)
```



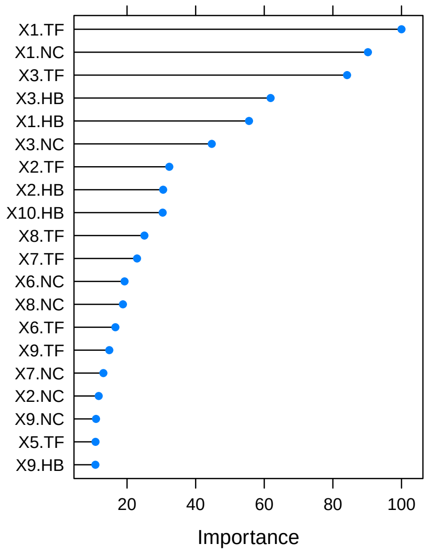
```
plot(rf_plot[[4]], top = 20)
```



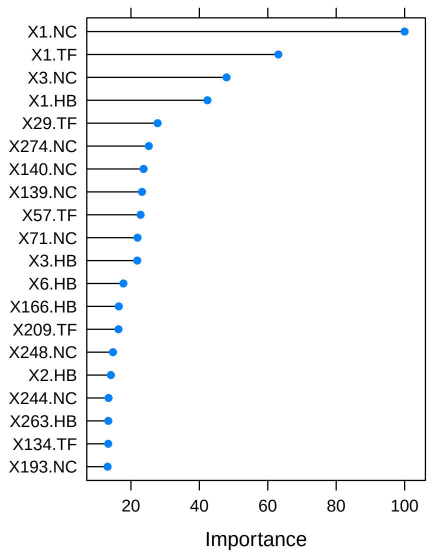
7.9.3 Gradient Boosting

Second the variable importance of the boosted trees is presented for the first and last run, such as seen in the paper.

```
gbm_plot[[1]]
```



```
gbm_plot[[4]]
```



8 Session Info and Warnings

The first code chunk shows with which version of R the code was ran. Please see r-data file for more info.

```
sessionInfo()
```

```
## R version 3.6.0 (2019-04-26)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 18362)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=German_Germany.1252  LC_CTYPE=German_Germany.1252
## [3] LC_MONETARY=German_Germany.1252 LC_NUMERIC=C
## [5] LC_TIME=German_Germany.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] xtable_1.8-4        stargazer_5.2.2     gbm_2.1.5
##  [4] randomForest_4.6-14 rpart.plot_3.0.8    rpart_4.1-15
##  [7] glmnet_3.0-1        Matrix_1.2-17       caret_6.0-86
## [10] ggplot2_3.2.1       lattice_0.20-38     mlogit_1.0-2
## [13] lmtest_0.9-37       zoo_1.8-5           Formula_1.2-3
## [16] evd_2.3-3           mvtnorm_1.0-11
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.1           lubridate_1.7.8      class_7.3-15
##  [4] assertthat_0.2.1     digest_0.6.25        ipred_0.9-9
##  [7] foreach_1.4.7        R6_2.4.0             plyr_1.8.4
## [10] stats4_3.6.0         e1071_1.7-3          evaluate_0.13
## [13] pillar_1.4.4         Rdpack_0.11-1       rlang_0.4.6
## [16] lazyeval_0.2.2       data.table_1.12.2   rmarkdown_2.3
## [19] splines_3.6.0        statmod_1.4.32      gower_0.2.1
## [22] stringr_1.4.0        munsell_0.5.0       compiler_3.6.0
## [25] xfun_0.7             pkgconfig_2.0.2     shape_1.4.4
## [28] htmltools_0.3.6      nnet_7.3-12         tidyselect_0.2.5
## [31] gridExtra_2.3        tibble_3.0.1        prodlim_2019.11.13
## [34] codetools_0.2-16     crayon_1.3.4        dplyr_0.8.5
## [37] withr_2.1.2          MASS_7.3-51.4       recipes_0.1.12
## [40] ModelMetrics_1.2.2.2 grid_3.6.0          nlme_3.1-144
## [43] gtable_0.3.0         lifecycle_0.2.0     magrittr_1.5
## [46] pROC_1.16.2          scales_1.0.0        bibtex_0.4.2.2
## [49] stringi_1.4.3        reshape2_1.4.3      timeDate_3043.102
## [52] ellipsis_0.3.0       vctrs_0.3.0         generics_0.0.2
## [55] lava_1.6.7           iterators_1.0.12    tools_3.6.0
## [58] glue_1.3.1           purrr_0.3.4         survival_2.44-1.1
## [61] yaml_2.2.0           colorspace_1.4-1    gbRd_0.4-11
## [64] knitr_1.23
```

search() shows the loaded packages

```
search()
```

```
##  [1] ".GlobalEnv"          "package:xtable"       "package:stargazer"
##  [4] "package:gbm"         "package:randomForest" "package:rpart.plot"
##  [7] "package:rpart"       "package:glmnet"       "package:Matrix"
## [10] "package:caret"       "package:ggplot2"      "package:lattice"
## [13] "package:mlogit"      "package:lmtest"       "package:zoo"
## [16] "package:Formula"     "package:evd"          "package:mvtnorm"
## [19] "package:stats"       "package:graphics"     "package:grDevices"
## [22] "package:utils"       "package:datasets"     "package:methods"
## [25] "Autoloads"           "package:base"
```

The warnings have not been presented in order to save space, but could be viewed in the r-data file if interested.

```
warnings()
```

References

[1] Bajari, P., Nekipelov, D., Ryan, S. & Yang, M. (2015): *Machine Learning Methods for Demand Estimation.* American Economic Review: Papers & Proceedings 105(5): 481-485.

[2] Berry, S., Levinson, J. & Pakes, A. (1995): *Automobile Prices in Market Equilibrium.* Econometrica, Vol. 63, No 4 (Jul., 1995), pp. 841-890.

[3] Breiman, L. (2002): *Manual On Setting Up, Using, And Understanding Random Forests V3.1.* URL: `https://www.stat.berkeley.edu/~breiman/Using_random_forests_V3.1.pdf.`, retrieved on 15.06.2020.

[4] Chernozhukov, V., Goldman, M., Semenova, V. & Taddy, M.(2018): *Orthogonal ML for Demand Estimation: High Dimensional Causal Inference in Dynamic Panels* Cornell University Working Paper.

[5] Chernozhukov, V., Hansen, C. & Spindler , M. (2016): *hdm: High-Dimensional Metrics.* R Journal 8(2), 185-199. URL: `https://journal.r-project.org/archive/2016/RJ-2016-040/index.html`.

[6] Croissant, Y. (1993): *Estimation of multinomial logit models in R :The mlogit Packages.* University de la Reunion.

[7] Croissant, Y. (2020): *Multinomial Logit Models.* R package version 1.1-0, URL: `https://cran.r-project.org/package=mlogit`.

[8] Dahl, D. B., Scott, D., Roosen, C., Magnusson, A. & Swinton, J. (2019): *xtable: Export Tables to LaTeX or HTML.* R package version 1.8-4. URL: `https://CRAN.R-project.org/package=xtable`.

[9] Foeller, D. (2012): *Simulation of Hypothetical Mergers within the German Automobile Market.* Muenchen, GRIN Verlag OHG. Available at: `https://www.diplomarbeiten24.de/document/208107`, retrieved on 02.15.2020.

[10] Friedman, J., Hastie, T. & Tibshirani, R. (2010): *Regularization Paths for Generalized Linear Models via Coordinate Descent.* Journal of Statistical Software, 33(1), 1-22. URL: `http://www.jstatsoft.org/v33/i01/`.

[11] Genz, A., Bretz, F., Miwa, T., Mi, X., Leisch, F., Scheipl, F. & Hothorn, T. (2020): *mvtnorm: Multivariate Normal and t Distributions.* R package version 1.0.11, URL: `https://CRAN.R-project.org/package=mvtnorm`.

[12] Green, G. & Richards, T. (2016): *Interpreting Results of Demand Estimation from Machine Learning Models.* Selected Paper prepared for presentation for the 2016 Agricultural & Applied Economics Association, Boston, MA, July 31-August 2.

[13] Greenwell, B., Boehmke, B., Cunningham, J. and GBM Developers (2019): *gbm: Generalized Boosted Regression Models.* R package version 2.1.5. URL: `https://CRAN.R-project.org/package=gbm`.

[14] Hastie, T., Tibshirani, R. & Friedman, J. (2008): *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, New York 2nd edition.

[15] Hill, S. (1989): *Managerial Economics: the analysis of business decisions.* 100-131, Macmillan, London 1. Edition.

[16] Hlavac, M. (2018): *stargazer: Well-Formatted Regression and Summary Statistics Tables..* R package version 5.2.1. URL: `https://CRAN.R-project.org/package=stargazer`.

[17] James, G., Witten, D., Hastie, T. & Tibshirani, R. (2013): *An Introduction to Statistical Learning: with Applications in R.* Springer, New York.

[18] Kuhn, M. (2008): *Building Predictive Models in R Using the caret Package.* Journal of Statistical Software, 28(5), 1 - 26. doi: `http://dx.doi.org/10.18637/jss.v028.i05`

[19] Kuhn, M. (2019): *The Caret Package.* Available at: `https://topepo.github.io/caret/model-training-and-tuning.html`. Retrieved on: 31.07.2020.

[20] Laurent, G. (2020), *An Introduction to Machine Learning with R: 5.4 Classification Performance.* Available at: `https://lgatto.github.io/IntroMachineLearningWithR/supervised-learning.html#classification-performance`, retrieved on 05.05.2020.

[21] Liaw, A. & Wiener, M. (2002): *Classification and Regression by randomForest.* R News 2(3), 18–22.

[22] McFadden, D. (1974): *Conditional logit analysis of qualitative choice behaviorbehavior, in P.Zarembka, ed., 'Frontiers in Econometrics',.* Academic Press, New York, pp. 105-42.

[23] Mueller, A., & Guido, S. (2016): *Introduction to Machine Learning with Python.* O'Reilly Media, Sebastopol 1st edition.

[24] Nevo, A. (2000): *A Practitioner's Guide to Estimation of Random-Coefficients Logit Models of Demand.* Journal of Economics & Management Strategy, Volume 9, Number 4, Winter 2000, pp. 513-548.

[25] Nevo, A. (2001): *Measuring Market Power in the Ready-to-Eat Cereal Industry.* Econometrica, Vol. 69, No. 2 (March, 2001), pp. 307-342.

[26] Packaging Distributors of America, (2020): *Packaging Stories: The Cereal Box.* Available at: `https://www.pdachain.com/2018/04/10/packaging-stories-the-cereal-box/`, retrieved on 10.02.2020.

[27] Paredes, M., Hemberg, E., O'Reilly, U. & Zegras, C.(2017): *Machine Learning or Discrete Choice Models for Car Ownership Demand Estimation and Prediction?.* IEEE: 2017, 780-785.

[28] R Core Team (2013): *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria. URL: `http://www.R-project.org/`.

[29] Spindler, M. (2018a), *Lecture Introduction: Taxonomy of Data Sets* [Powerpoint slides]. Machine Learning with Applications in Business Administration, University of Hamburg, delivered 15. October 2018.

[30] Spindler, M. (2018b), *Lecture 2: Modern Linear Regression for High-Dimensional Data* [Powerpoint slides]. Machine Learning with Applications in Business Administration, University of Hamburg, delivered 12. November 2018.

[31] Spindler, M. (2018c), *Regression 3.1. Modern Nonlinear Regression. Trees, Random Forests, and Boosted Trees* [Powerpoint slides]. Machine Learning with Applications in Business Administration, University of Hamburg, delivered 9. December 2018.

[32] Spindler, M. (2019), *Lecture 4: Causal Framework Introduction* [Powerpoint slides]. Machine Learning with Applications in Business Administration, University of Hamburg, delivered 15. January 2019.

[33] Stephenson, A. G. (2002): *evd: Extreme Value Distributions.* R News, 2(2):31-32, June 2002. URL: `https://CRAN.R-project.org/doc/Rnews/`

[34] Therneau, T. & Atkinson, B. (2019): *rpart: Recursive Partitioning and Regression Trees.* package version 4.1-15. URL: `https://CRAN.R-project.org/package=rpart`.

[35] Train, T. (2013): *The TikZ and PGF Packages.* Manual for version 3.0.0. URL: `http://sourceforge.net/projects/pgf/`.

[36] Train, K. (2002): *Discrete Choice Methods with Simulation.* Cambridge University Press, pp. 1-61.

[37] Varian, H. (2014): *Big Data: New Tricks for Econometrics.* Journal of Econometric Perspectives, 28 (2): 3-28.

[38] Wikiversity, (2020): *Managerial Economics/Demand estimation.* Available at: `https://en.m.wikiversity.org/wiki/Managerial_Economics/Demand_estimation`, retrieved on 30.11.2019.

Declaration in Lieu of Oath

„I hereby declare in lieu of oath that I have composed the enclosed master's thesis entirely on my own and without any help from others. I have not used any outside sources without declaring them in the text. Any concepts or quotations applicable to these sources, are clearly attributed to them. This master's thesis has not been submitted in the same or substantially similar form to any other authority for grading. The submitted written version corresponds to the version on the electronic storage medium."

Place, Date	Signature