Title
Prediction of Electronic Component Prices: from Classical Statistical and Machine Learning Models to Deep Neural Networks with Feature Embedding

Author
Zhang, Yu

Publication Date
2019

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
Los Angeles

Prediction of Electronic Component Prices: from Classical Statistical and Machine Learning Models to Deep Neural Networks with Feature Embedding

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Statistics

by

Yu Zhang

2019

ABSTRACT OF THE THESIS

Prediction of Electronic Component Prices:
from Classical Statistical and Machine Learning Models to Deep Neural Networks with Feature Embedding

by

Yu Zhang
Master of Science in Statistics
University of California, Los Angeles, 2019
Professor Yingnian Wu, Chair

The unit price of an electronic component with certain specifications and purchase details could be crucial for the decision-making of customers. With the massive historical purchasing data at Supplyframe, Inc., classical statistical and Machine Learning (ML) models are used to capture the underlined relationship and predict accurate price. The Naive model using mean estimation is adopted as baseline models, followed by the exploration of a wide range of machine learning models including Ordinary Least Squares, Supporting Vector Machine, k-Nearest Neighbors, Random Forests (RF), Extreme Gradient Boost (XGB). To make better use of the unstructured features, Deep Neural Networks (DNN) are built based on Convolutional Neural Networks and feature embedding, which maps unstructured features to higher dimension vectors. We observe that the RF and XGB models outperform other classical statistical and ML models when only the structured features are used while the DNN model is proved to be the most powerful by combining both structured and unstructured features. Consistent superior performances are found for the DNN model in terms of the root mean squared error, the prediction interval of the ratios of observed and predicted values, the prediction coverage rate and the capture of the monotonic decreasing relationship between unit prices and purchase quantities.

The thesis of Yu Zhang is approved.

Henry Burton

Jingyi Li

Yingnian Wu, Committee Chair

University of California, Los Angeles

2019

To my parents . . .

for their unconditional love and support

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to acknowledge and thank all the professors in my committee: Prof. Yingnian Wu, Prof. Jingyi Li and Prof. Henry Burton, for their guidance in my study and research. Special thanks also go to Mr. Aleksandar Bradic, the Chief Technology Officer of Supplyframe, Inc., for giving me the chance of the summer intern as a data scientist and providing the valuable data for my project and this thesis. Advice and support given by the colleagues at UCLA and Supplyframe is also appreciated.

CHAPTER 1

Introduction

1.1 Background and Motivation

The unit price of an electronic component with certain specifications and purchase details is one of the most important piece of information needed by the customers for their decision-making process. Supplyframe, Inc. builds a widely used and trusted industry network for electronics design and manufacturing. The company provides open and connected access to the worlds largest collection of vertical search engines, supply chain tools, and online communities. One of its key product is the search engine "www.findchips.com", which is aimed to provide instant and accurate quotes of electronic components with customized specifications and purchase details. The unit price is the key to such a quote. Given the fact that the unit prices of an electronic component may vary in a large range, it is possible that such variance could be explained at least partially by their associated specifications and purchase details. For example, a microcontroller may has specifications like CPU speed, memory size, address bus width, etc., which could affect on its unit price. Even for the components with the same or similar specifications, their unit prices my vary among different manufactures and distributors. Moreover, the unit price could also depend on the purchase volume of each order as discount rates may change for bulk purchases.

The existing approach, which is widely used in the industry, estimates the unit price by fitting historical pricing data for each electronic component to a specified distribution (for example, Normal distribution), where the information of specifications and purchase details is not incorporated and considerable amount of data for the same electronic component is required to ensure its performance or even its validation. However, the data sparsity caused

by either limited historical purchases or missing data often fails such a requirement.

1.2 Objective

With the help of the massive historical purchasing data for various electronic components at Supplyframe, Inc., this study explores the relationship between unit prices and their associated specifications and purchase details of electronic components. Classical statistical and Machine Leaning (ML) models as well as Deep Neural Networks (DNN) are then built to capture the underlined relationship and predict the unit price of any given electronic component. The performance of these models are thoroughly compared and the optimal model is selected, which could provide strong data-driven support for the decision-making process of the customers. For example, specifying the required specifications and expecting a price estimate as output, a customer could tune the specifications and consider budget control based on the estimated unit price.

1.3 Chapter Overview

This study consists of 5 chapters, which are summarized as follows:

Chapter 1 gives the background, motivation and objectives of the study as well as the overview of each chapter.

In Chapter 2, Exploratory Data Analysis (EDA) is carried out to understand the basic properties of the data, followed by the feature engineering for the structured numerical and categorical data, including 1) selecting potential informative features from all the raw data, 2) transforming and scaling features to the desired domain and 3) imputing the missing data.

A series of classical statistical and ML models are explored in Chapter 3, including baseline naive model using mean estimation, Ordinary Least Squares (OLS), Least Absolute Shrinkage and Selection Operator (LASSO), Support Vector Machine (SVM), k-Nearest Neighbors (k-NN) and two tree-based ensemble methods, Random Forests (RF) and Extreme Gradient Boosting (XGB). k-fold cross validation is used to tune the model parameters and

select the optimal model, where Root Mean Squared Error (RMSE) is adopted for the optimization for the loss function. The final performance metric is evaluated using the imperial Prediction Interval (PI) of the Ratios of Observed and Predicted Values (ROPV) for unit prices. As none of these methods could directly handle unstructured features such as texts and series numbers, only structured numerical and categorical features are used in this chapter.

Chapter 4 first introduces the feature embedding of text and serial numbers using high dimension vectors. The detailed structure of the DNN is then presented, where 1-D Convolutional Neural Networks (CNN) are used to extract useful information from the embedding vectors and further combined with the structured features to achieved the final prediction. The optimal model selected from Chapter 3 is then compared with the proposed DNN model using the same metrics defined in Chapter 3. The prediction coverage rate and the monotonic decreasing relationship between unit prices and purchase quantities are also discussed.

Chapter 5 summarizes all the work and presents the final conclusions and key findings. The limitations of the current study and potential future work are also covered.

CHAPTER 2

Exploratory Data Analysis and Feature Engineering

2.1 Exploratory Data Analysis

The dataset has a total of around 1 million samples. Based on different data type, the features have been grouped into structured data consisting of numerical and categorical features and unstructured data. Numerical features are originally given as continuous values, while categorical ones are assigned discrete values. The unstructured data include two kinds of features: the texts where several sentences are used to introduce and evaluate an electronic component and series numbers where some "random" sequences of alphabets and numbers with certain lengths are used as the internal identifications of electronic components, for example the feature "Mfr Part Number". These unstructured features need special transformation or embedding to be incorporated into any predictive models.

How strong or informative a certain feature is depends mainly on two factors: the underlined relationship with the unit price and the completeness of this feature. To balance these two factors, a relatively large missing data rate of 70% is used as the threshold to filter all the features. As shown in Figure 2.1, a total of 64 features are retained from all the available purchase records and the missing data rate ranges from 0% all the way to 66.7%. For example, the feature "CPU Family" is a categorical feature and has a missing data rate of 44.5%, which is relatively high among others; however, it is observed from Figure 2.2 that the median and quantile of unit price varies significantly as the feature "CPU Family" changes (only the first 50 categories are shown), indicating a strong underlined relationship. By taking both factors into consideration, the feature "CPU Family" has the potential to be a very informative one for a predictive model, showing the importance of a relatively high

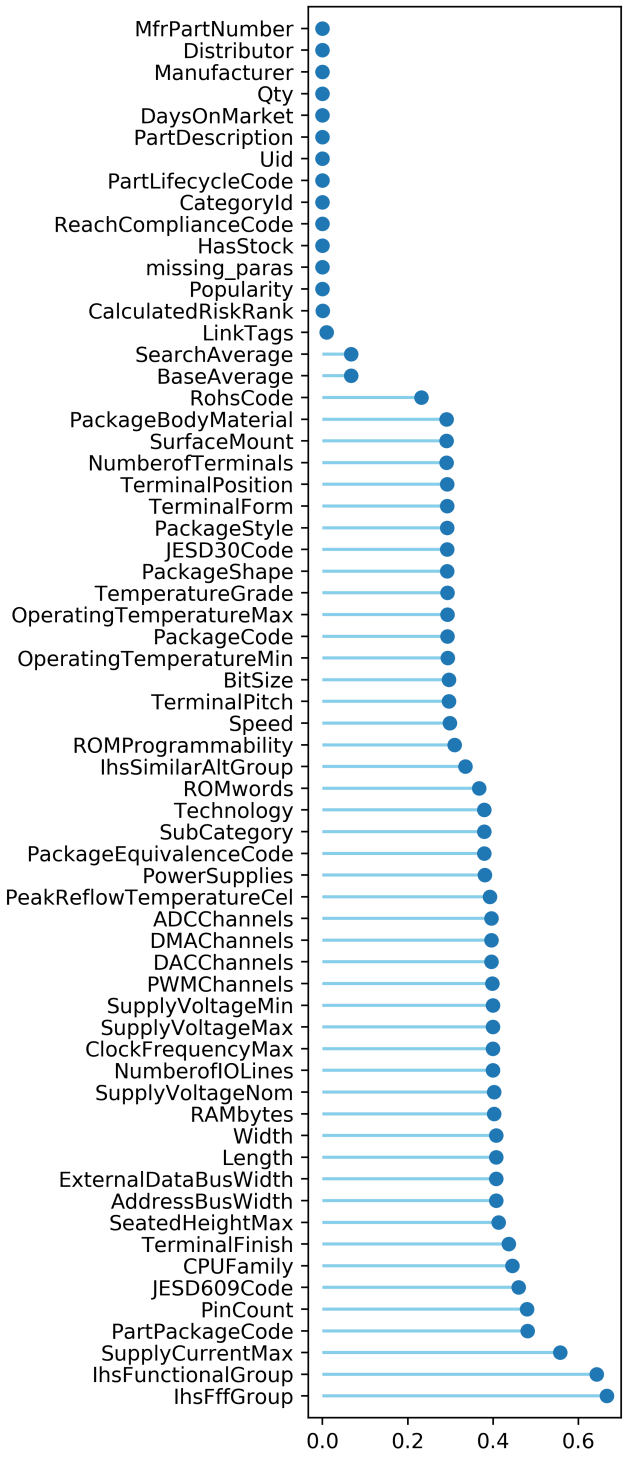

Figure 2.1: Percentage of missing data in the retained 64 features

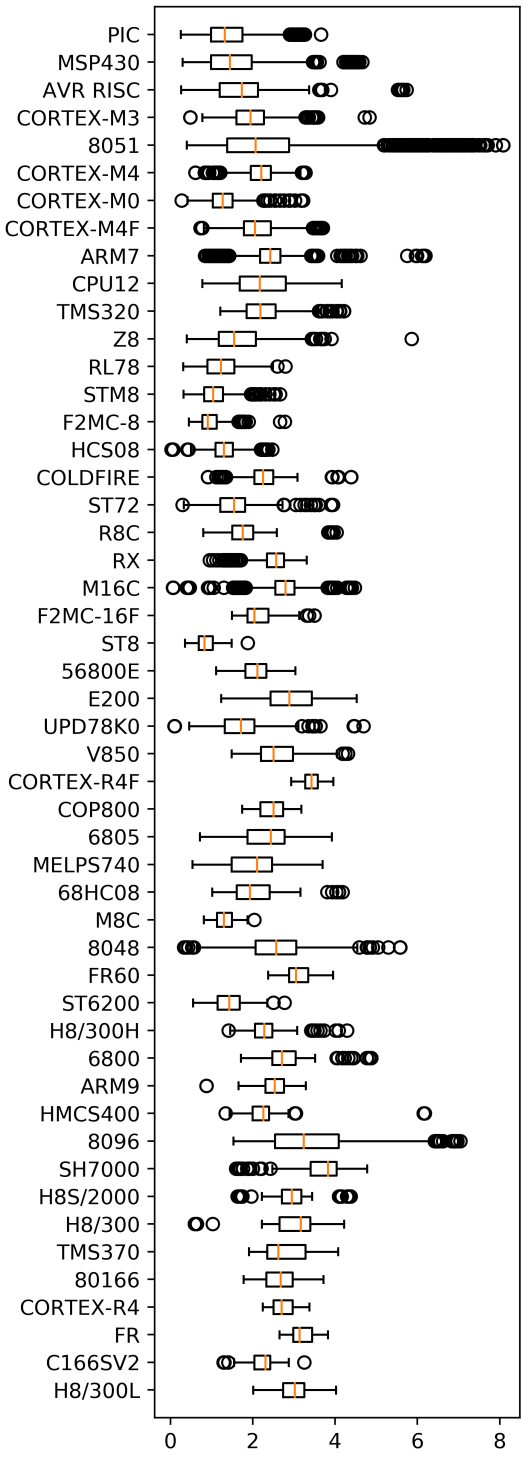

Figure 2.2: Unit price versus the feature "CPU Family"

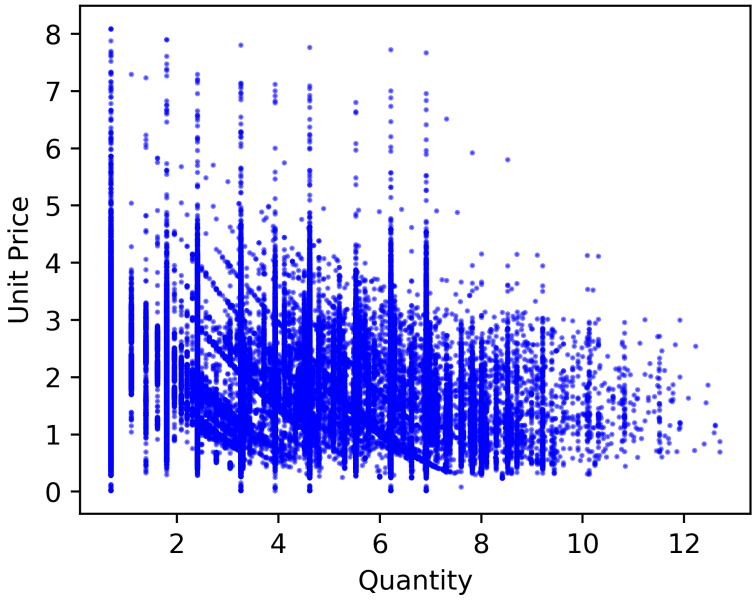

Figure 2.3: Unit price versus the feature "Quantity"

threshold for missing data rate.

Another interesting relationship is between the feature "Quantity" and the unit price, as higher discounts are expected with large purchase volumes. This is further proved in Figure 2.3, where the unit price tends to go lower as the quantity increases. However, large variance of unit price is still observed for the same quantity, especially when quantity is small, indicating additional features are needed for an accurate prediction.

Figure 2.4 shows the change of unit price as the "Address Bus Width" feature varies. Although this feature is recognized as a numerical feature and treated as continuous values, it is observed that this feature has limited values and is more like a discrete variable. Moreover, the variance of the unit price for a given "Address Bus Width" value is also relatively large. This is actually the opposite with the categorical feature "CPU Family", which has more than 100 categories and relatively lower variance of unit price within each category. Such observations also go for most of the numerical and categorical features in the data set.

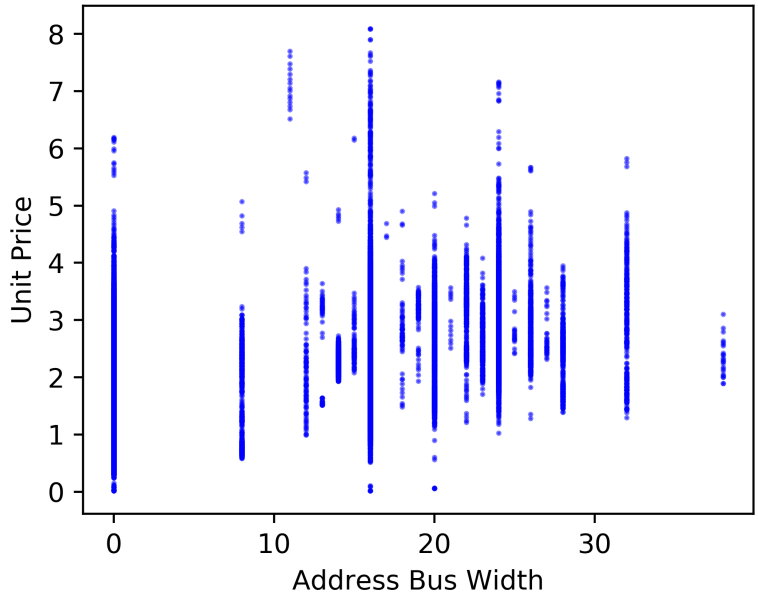

Figure 2.4: Unit price versus the feature "Address Bus Width"

2.2 Feature Engineering

In additional to the large ratios of missing data discussed in the previous section, another key property of the categorical features in the studied dataset is that most of them have a large number of classes (for example, the "CPU Family" feature alone has 99 categories), which cannot be handled directly by most of the classical statistical and ML models or by one-hot transformation. It should also be noted that for the categorical data a value of "other" is treated as a separate category rather than missing data because it excludes the possibility of being in any other categories, which is still a piece of useful information. Moreover, although some of the tree-based ensemble methods may directly take the categorical variables as input and do not need any special normalization for the numerical variables as well, it is still necessary to normalize all the features to the same dynamic range such that the distance metric for performing missing data imputation using k-NN can be defined among different variables.

2.2.1 Feature Selection

The feature engineering in this study starts with the preliminary filtering of the raw data, where more than 100 features are recorded for a certain electronic component. Actually, this step was already performed in the previous section using the 70% missing data threshold.

2.2.2 Data Normalization

Two distinct approaches are used in data normalization for the classical statistical and ML models and the DNN model with respect to how they handle the input data. Logarithm transformation is used for the response, unit price.

The data normalization method used by Gupta from Airbnb with the post named "Overcoming Missing Values In A Random Forest Classifier" are adopted here for the first normalization approach in this study. For a numerical variable X, the mapping shown in Equation 2.1 transforms each of its instance x to be within dynamic range $[0, 1]$.

$$y = F_X(x) \tag{2.1}$$

where F_X is the empirical cumulative distribution function (CDF) of variable X, and it can be shown that y is uniformly distributed over $[0, 1]$.

The fact that all the transformed variables are all uniformly distributed over $[0, 1]$ enables us to define a distance metric on a pair of feature vectors, which will be shown next. However, since the transformation relies on the empirical CDF, the transformed variable may not be uniform when the data is very sparse or only limited samples are available.

A special treatment is needed for a categorical feature, when an ordering has to be defined when transforming it to a numerical one. This is done by replacing each category value using the mean estimate of all the responses in the training dataset that falls into the specific category.

The second normalization approach is used for DNN models. All the numerical variables are scaled to have zero mean and unit 1 variance excluding the missing data, which is directly

set to 0 and is equivalent to use the mean imputation. For the boolean categorical features, 1 and -1 are used for the positive and negative cases, respectively. Similarly, all the missing boolean value is replaced with 0. For DNN models, feature embedding technique is used for all the other categorical and unstructured data and is discussed in detail later in Chapter 4.

2.2.3 Missing Data Imputation

As discussed earlier, a large proportion of data are missing for many features in the studied dataset, so the missing data imputation is needed before feeding the data to any of the models. Two imputation methods are tested here, i.e., the mean imputation where a missing value is replaced using the mean estimate of all the available values for a given feature and the k-NN imputation where the weighted mean estimate of the k nearest samples is used (see detailed steps in Section 3.1.5).

Given two feature vector $\mathbf{x_i}$ and $\mathbf{x_j}$ that have already been transformed following Section 2.2.2, the distance metric $d(\mathbf{x_i}, \mathbf{x_j})$ for the k-NN imputation is defined in Equation 2.2, which is essentially the Euclidean distance. Different with the traditional regression or classification problem using k-NN, additional definition of distance of missing data is needed, as given by Equation 2.3.

$$d(\mathbf{x_i}, \mathbf{x_j}) = \sqrt{\sum_k f(x_i^k, x_j^k)^2)} \tag{2.2}$$

$$f(x_i^k, x_j^k) = \begin{cases} \lambda & \text{if either } x_i^k \text{ or } x_j^k \text{ is missing} \\ 0 & \text{if } |x_i^k - x_j^k| < \lambda \\ 1 & \text{otherwise} \end{cases} \tag{2.3}$$

where $f(x_i^k, x_j^k)$ denotes the difference of the values from sample i and sample j for a certain feature k. Three scenarios are considered: 1) a value of λ is given as penalty if one of the values is missing; 2) 0 is used when both two values are available and differ less than λ; 3) 1 is assigned when both two values are available but differ more than λ.

CHAPTER 3

Classical Statistical Models

This chapter reviews the classical statistical and ML models used for this study. Model training, tuning and comparing are then performed to select the optimal model from the pool, which could be further analyzed with the DNN model described in Chapter 4. Performance metrics used for model selection are also defined. Python (version 3.6.6) is used for the coding in this chapter.

3.1 Model Introduction

3.1.1 Naive Model

The naive model is used as a baseline which makes predictions using the mean training response $\bar{Y}$ as shown in 3.1.

$$\hat{Y} = \bar{Y} = \frac{1}{N} \sum_{i} y_i \tag{3.1}$$

where i denotes each sample and y_i is the associated response; N is the total number of samples in the training dataset.

3.1.2 Ordinary Least Squares

The Ordinary Least Squares (OLS) is a regression method assuming linear relationship between the responses and features. Equation 3.2 shows its matrix form. OLS models are built using "*OLS*" class from Python Package "*statsmodels.api*".

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{3.2}$$

where $\mathbf{X}$ is a $n \times p$ feature matrix; $\mathbf{Y}$ is a $n \times 1$ response vector; and ϵ is the residual which is assumed to follow a Normal distribution with mean 0 and constant variance σ_n^2; β is the model parameter of dimension $p \times 1$ and can be directly solved by minimizing the Sum of Square Errors (SSE) objective function shown in Equation 3.3 to get the close form solution given by Equation 3.4.

$$J = \sum_i (\hat{y}_i - y_i)^2 \tag{3.3}$$

$$\hat{\boldsymbol{\beta}} = (\mathbf{X^T X})^{-1}\mathbf{XY} \tag{3.4}$$

where $\hat{y}_i$ is the predicted response for sample i.

3.1.3 Least Absolute Shrinkage and Selection Operator

The Least Absolute Shrinkage and Selection Operator (LASSO) [1] has been widely used to improve the stability and accuracy of traditional OLS models. This is achieved by penalizing the objective function of OLS regression using the l_1 norm of the model coefficients, as shown in Equation 3.5. More importantly, it also enables efficient feature selection in the original high dimensional space as it allows the optimal coefficients of a subset of the original feature space to shrinkage to zeros. LASSO can also potentially reduce the multicollinearity in the selected features. For example, if a subset of features is highly correlated, only the one with highest dispersion will be selected and the remainder will be removed as they penalize the objective function without providing substantial additional information.

$$J = \sum_i (\hat{y}_i - y_i)^2 + \lambda \|\boldsymbol{\beta}\|_1 \tag{3.5}$$

where y_i and $\hat{y}_i$ are the observed and predicted responses respectively; $\boldsymbol{\beta}$ is the coefficient vector; $y_i = \boldsymbol{\beta^T} x_i + \beta_0$; $\lambda \geq 0$ is a regularization term used to penalize the complexity of the

model.

For a given value of λ, there is a corresponding number of nonzero coefficients, the effective degrees of freedom of the model, which is denoted as $df(\lambda)$. When $\lambda = 0$, there is no penalty and $df(\lambda)$ is simply the total number of the features in the original data set and an OLS solution is obtained. As λ increases, $df(\lambda)$ shrinks to zero and the outcome is a naive model, where the prediction is simply the mean of the observed responses. The parameter tuning is needed to find the optimal value of $\lambda's$. Features having relatively large variances and low correlations are desired by LASSO as they not only provide small absolute values but also reduce the number of coefficients. LASSO models are built using "$linear_model.Lasso$" class from Python Package "$sklearn$".

3.1.4 Support Vector Machine

The Support Vector Machine (SVM) technique was first introduced by Vapnik [2] in 1995 to solve binary classification problems and has since been extended to regression problems [3]. Originally being used to find the optimal hyper plane for linear separate data points in different categories, SVM become much more powerful when combined with kennel tricks which transform the raw features into a high dimensional feature space such that they are easier to separate using a hyper plane for classification or are closer to a linear relationship for regression. As defined by Vapnik [3] in Equation 3.6, the ϵ-insensitive loss function only penalizes the data points lying outside an ϵ-tube. By combining the squares of l_2 norm to regularize it, the final objective function for SVM could be expressed in Equation 3.7.

$$L_\epsilon(y_i, \hat{y_i}) = \begin{cases} 0 & \text{if } \epsilon < 0 \\ |y_i - \hat{y_i}| - \epsilon & \text{otherwise} \end{cases} \tag{3.6}$$

$$J = \sum_i L_\epsilon(y_i, \hat{y_i}) + \lambda \|\boldsymbol{\beta}\|_2^2 \tag{3.7}$$

where ϵ is the width of the ϵ-tube used to fit the training dataset. By introducing two slack variables $\xi_i^+ \geq 0$ and $\xi_i^- \geq 0$ defined in Equation 3.11 and dividing the objective function

by 2λ, a constrained optimization problem given in Equation 3.12 could be obtained.

$$\begin{cases} y_i \leq \hat{y}_i + \epsilon + \xi_i^+ \\ y_i \leq \hat{y}_i - \epsilon - \xi_i^- \end{cases} \tag{3.8}$$

$$J = C \sum_i (\xi_i^+ + \xi_i^-) + \frac{1}{2} |\boldsymbol{\beta}\|_2^2 \tag{3.9}$$

where $C = \frac{1}{2\lambda}$ is the new model parameter for the regularization term.

It could be shown that the solution for this optimization problem is $\hat{\beta} = \sum_i \alpha_i x_i$, where α_i is a constant parameter for each sample [4]. As a key characteristic of the ϵ-insensitive loss function, α vector is sparse, and only x_i with $\alpha_i > 0$ is used in the model and is the so-called support vector. The predicted response given the feature vector x_j from a certain data point is presented in Equation 3.14:

$$\hat{y_j} = \sum_i \alpha_i \mathbf{x_i^T x_j} + \beta_0 \tag{3.10}$$

The kernel trick is then applied by replacing the inner product $\mathbf{x_i^T x_j}$ with the a certain kernel $k(\mathbf{x_i}, \mathbf{x_j})$, and optimization algorithms are used to estimate the sparse α vector. Example kernels could be a linear kernel, a polynomial kernel and a Radial Basis Function (RBF) kernel [5]. SVM models are built using "*svm.SVR*" class from Python Package "*sklearn*".

3.1.5 k-Nearest Neighbors

The k-Nearest Neighbors (k-NN) [6, 7] is a non-parametric method that could be used for classification and regression problems. Given the feature vector of a sample for prediction, it basically uses the major vote rule for classification and mean estimation for regression based on the responses of the available samples near it in the training dataset. The detailed k-NN algorithm works for a regression problem as follows: 1) Computing the Euclidean or Mahalanobis distance from the sample for prediction to the samples in the training dataset; 2) Choosing the first k samples that are closest to the sample for prediction; 3) For the chosen

samples, weighting their responses by the inverse of their distances to the sample for prediction; 4) Using the weighted sum of the responses as the predicted response. Generally speaking, cross-validation is used to find a heuristically optimal number k of nearest neighbors using predefined performance metrics. k-NN models are built using "$KNeighborsRegressor$" class from Python Package "*sklearn*".

3.1.6 Decision Tree

The Decision Tree, also referred as Classification and Regression Tree (CART) [8], is a machine learning algorithm that can be used for both discrete categorical and continues ordered predictions. By recursively partitioning the feature space, a decision tree seeks to create mutually exclusive subspaces with less impurity for classification, where Gini Index and cross entropy are widely used as the impurity measures, or less sum of within-leaf variance for regression. Given the objective of this study, the following details would be mainly focus on the application in regression.

A Decision Tree model normally includes three main step. First, the binary tree grows following the greedy algorithm trying o find a feature and an associated split point that minimizes the within-leaf variance in subspaces. Equation 3.11 is the objective function for optimization, which could be define as the sum of square errors for all the leaves of a tree T as well as the sum of within-leaf variance.

$$J = \sum_{c \in T} \sum_{i \in c} (y_i - \bar{y_c})^2 = \sum_{c \in T} n_c V_c \tag{3.11}$$

where c denotes a certain leaf; i represents a sample; $\bar{y_c} = \frac{1}{n_c} \sum_{i \in c} y_i$ is the predicted value at leaf c; n_c is the number of samples falling in leaf c, and V_c is the variance of leaf c.

The second step involves establishing an appropriate criterion for stopping the growth of the tree. Examples of stopping criteria include a minimum number of samples needed at a given node for further split, a maximum tree depth, a limit on the total number of leaves in a tree. An alternative stopping criterion that balances the tree complexity and its associated objective function could be achieved by introducing a penalty term. The concept of cost-

complexity tuning [8] is then adopted, which penalizes the objective function J by adding a term $c_p N_T$, where c_p is a constant parameter and N_T is the total number of leaves in tree T. It should be noted that the optimal value of c_p could be very different for different datasets. If it is too large, the model will be unable to adequately capture the characteristics of the dataset. On the other hand, a c_p that is too small can lead to overfitting of the training data and have poor performance on the testing data.

Finally, the tree need proper pruning to control the overfitting issue. This could be achieved by randomly selecting a hold-out set from the training data at the very beginning, passing it through the generated tree, and testing all splits from the bottom-up, deleting those whose removals would not reduce the overall impurity for the hold-out set. The pruning process continues recursively until there are no further improvements.

Based on the Decision Trees, two main ensemble methods are introduced and applied in this study, RF and XGB. The former seeks to improve the performance of multiple deep trees by reducing the prediction variance, while the latter focuses on directly reduce prediction bias using relatively shallow trees.

3.1.7 Random Forests

Although powerful, a decision tree is very sensitive to the specific dataset on which it is trained and is therefore a high-variance model. As a result, the tree and corresponding predictions generated for different subsets of the same training dataset could be quite different. This problem can be addressed by assembling bootstrap-sampled subsets of the training data. Bootstrap is a statistical technique that involves randomly sampling from a dataset to create a series of sub-datasets [9, 10]. Each random sample is placed back into the original dataset such that multiple (or no) instances of a particular sample can be included in the sub-dataset. Based on these resampled datasets, bootstrap aggregation, also known as Bagging [11], is an ensemble method, which seeks to achieve strong predictions by combining multiple week models. Bagging decision trees is used to train the predictive models using all these resampled datasets and generate aggregated predictions using the mean of their

predictions.

The Random Forests (RF) algorithm [12] goes one step further to overcome the inherent defect of Bagging where highly correlated tree structures could be generated [13]. It is was developed with a critical modification to reduce such tree correlation, i.e., at each split point during the growth of the tree, RF only applies the greedy algorithm to a randomly selected proportion of the original features. RF models are built using "*RandomForestRegressor*" class from Python Package "*sklearn*".

3.1.8 Gradient Boosting

Initially introduced by Breiman [14], Gradient Boosting is a ML technique which builds a strong prediction model by ensemble of weak prediction models, where decision trees are typically used. It has then been extended for the explicit regression boosting [15, 16] and more generalized functional gradient boosting [17, 18]. It builds and adds new trees to the existing model by following the direction of the gradient descent of the objective function. Aside from the original Gradient Boosting Machine (GBM), different variations have been developed over the years, such as Adaptive Boosting (AdaBoost) [19, 20], Extreme Gradient Boosting (XGB) [21], Light GBM [22, 23] and CatBoost [24, 25].

The XGB model is selected in this study as a representative for the Gradient Boosting methods. Initially introduced to the Distributed (Deep) Machine Learning Community (DMLC) group by Chen [21], XGB has been used as the award-winning algorithms for many competitions and become more and more popular in recent years. Readers are referred to the aforementioned studies for the detailed mathematical formulation of these methods. XGB models are built using Python Package "*xgboost*".

3.2 Model Training

The main purpose here is to get evaluations for the previously introduced classical statistical and ML models, where parameter tuning is performed for each model. As the raw dataset

contains more than 1 million samples and would require significantly long time for analyses, only 40% data are used in the following analyses in this section for better efficiency, where 30% are used for training and 10% for testing.

3.2.1 Performance Metrics

The Root Mean Squared Error (RMSE) defined in Equation 3.12 is used as the first performance metric. Aside from it, a more piratical and meaningful metric is introduced as the imperial prediction Interval (PI) of the Ratios of Observed and Predicted Values (ROPV) for unit prices, as shown in Equation 3.14. The motivation for the ROPV metric is from the needs of customers who are interested in the range of the actual unit price when given a predicted one. For example, if an unit price of \$100 is predicted, which is a point estimate, it could be even better for a customer to know that the actual price ranges from \$88 to \$111 with 90% confidence. It should be noted that this is different with the commonly-used Mean Absolute Relative Difference (MARD) metric, where the absolute relative difference is computed as the ratio of the absolute difference over the observed value.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i}(y_i - \hat{y}_i)^2} \tag{3.12}$$

$$ROPV = \frac{y_i}{\hat{y}_i} \tag{3.13}$$

where y_i and $\hat{y}_i$ are the observed and predicted responses respectively; N is the number of samples used for training.

It should be noted that the RMSE metric is computed using the logarithm transformations of the unit prices, while the ROPV metric is evaluated by converting the predictions to the original domain.

3.2.2 k-Fold Cross Validation

Within each training procedure, model parameters are determined by k-fold Cross Validation (CV) [26] using training data. A model is trained using the $(K-1)$ folds of data and evaluated by the left one fold. The CV performance is then obtained by averaging the performance metrics from the K results. $K = 5$ is used in the following analyses.

$$CV = \frac{1}{K} \sum_{k} RMSE^k \tag{3.14}$$

where k denotes a certain fold and K is the total number of folds; $RMSE^k$ is the performance metric by the k^{th} fold.

3.2.3 Parameter Tuning

For the Naive and OLS model, close form solutions are directly otained, while parameter tuning using k-fold cross validation are used for LASSO, SVM, k-NN, RF and XGB models.

For LASSO, the regularization term λ is tuned by a geometric sequence in the range from 10^{-4} to 10^2 by a common ratio of 10, and the optimal CV performance in terms of RMSE is obtained at $\lambda = 10^{-3}$. 31 numerical and categorical features are selected by LASSO from the original 64 features.

The regularization term C and the tube width parameter xi for the SVM models are tuned using 2-D grid search, where C ranges from 10^{-2} to 10^2 as a geometric sequence with a common ratio of 10 and xi is explored as 0.01, 0.05, 0.1, 0.2 and 0.3. It is found that the combination of $C = 10$ and $xi = 0.1$ gives the lowest CV RMSE.

A parameter set of 5, 10, 20, 30 is used for the k-NN models, which is optimized at $k = 5$.

The RF models are tuned at a 3-D grid defined by the maximum depth (10, 15, 30, 50), the number of trees (100, 300, 500, 700) and the number of features that could be used for each split (8, 16, 25). The CV result gives the optimal parameters in the same order as 30, 500 and 8.

Finally, the 3-D grid with the maximum depth (10, 15, 20, 25), the number of estimators

(200, 500, 800, 1000) and the learning rate (0.05, 0.1, 0.2, 0.5) are searched using CV for the XGB models. The model with maximum depth of 10, learning rate of 0.1 and using a total of 800 estimators obtains the optimal performance in terms of CV RMSE.

3.3 Performance Evaluation and Discussion

The training RMSE for all the models using CV on the training dataset, the testing RMSE and the associated boundaries of the 3 ROPV PI levels (50%, 80%, 90%) are presented in Table 3.1.

It is observed that the Naive, as a baseline model, has the highest RMSE for both the training and testing dataset as well as the widest PIs at all the 3 levels. OLS and LASSO models achieve very similar performance but the LASSO model only uses around half of the features. The SVM model using RBF kernel reduces the RMSE considerably by around 0.7 for both the training and testing dataset. This could mainly due to the nonlinear transformation by the RBF kernel. It should be noted that the Naive, OLS, LASSO and SVM models all have consistent performance for the training and testing dataset, indicating that the overfitting is probably not an issue here.

Table 3.1: Performance comparison of classical statistical and ML models

Model	*RMSE Train*	*RMSE Test*	*ROPV* 50% *PI*	*ROPV* 80% *PI*	*ROPV* 90% *PI*
Naive	0.719	0.732	[0.466, 1.640]	[0.274, 2.736]	[0.214, 3.869]
OLS	0.462	0.465	[0.685, 1.348]	[0.451, 1.915]	[0.342, 2.436]
LASSO	0.466	0.475	[0.680, 1.354]	[0.447, 1.914]	[0.340, 2.443]
SVM	0.398	0.403	[0.770, 1.250]	[0.543, 1.740]	[0.435, 2.306]
k-NN	0.303	0.407	[0.791, 1.239]	[0.564, 1.672]	[0.424, 2.200]
RF	0.167	0.347	[0.857, 1.147]	[0.647, 1.443]	[0.463, 1.851]
XGB	0.173	0.337	[0.867, 1.141]	[0.650, 1.435]	[0.464, 1.863]

While significant lower RMSE is found for the k-NN model on the training dataset, similar performance with SVM is observed for the testing dataset, meaning that the k-NN

model overfits the former a little bit without capturing additional patterns in the data.

When it comes to the tree-based ensemble models, RF and XGB, one can see that very small RMSEs around 0.17 are generated from training dataset but significantly higher ones around 0.34 are found on the testing dataset, indicating considerable overfitting in these models. However, they are still selected as the final optimal models in this chapter given their superior performance over other models on the testing dataset.

CHAPTER 4

Deep Neural Networks

Deep Neural Networks (DNN) is becoming more and more popular in recent years thanks to its high prediction performance, especially for massive data such as time series, audio and vision data as well as decision-making and optimal strategy problems in complex environment. This chapter first introduces the feature embedding technique used to transform unstructured features to their associated vector representations. The general structure of CNN, which is the key to capture informative pattern from unstructured features, is then reviewed. Finally, the proposed DNN structure is presented and its performance is evaluated and compared with the optimal models RF and XGB from Chapter 3. The open source Python deep learning library "$Keras$" with "$Tensorflow$" as the backend is used to build DNN models in this chapter.

4.1 Feature Embedding

Feature embedding technique is a powerful way to map data to arbitrarily higher dimensional space and provide the unique opportunity to explore and model the relationship between the features and the prediction responses. A very successful application is the word embedding used for Natural Language Processing (NLP) [27, 28, 29], where words with similar meaning are learned to be closer to each other in a word space thus increasing the continuity of the words compared to using one-hot encoding. A similar approach has also been carried out for categorical data [30].

Using one feature in the current dataset for example, the categorical feature "Manufacturer" has a set of different categories such as"AMD", "ANALOG DEVICES INC", "ASIX

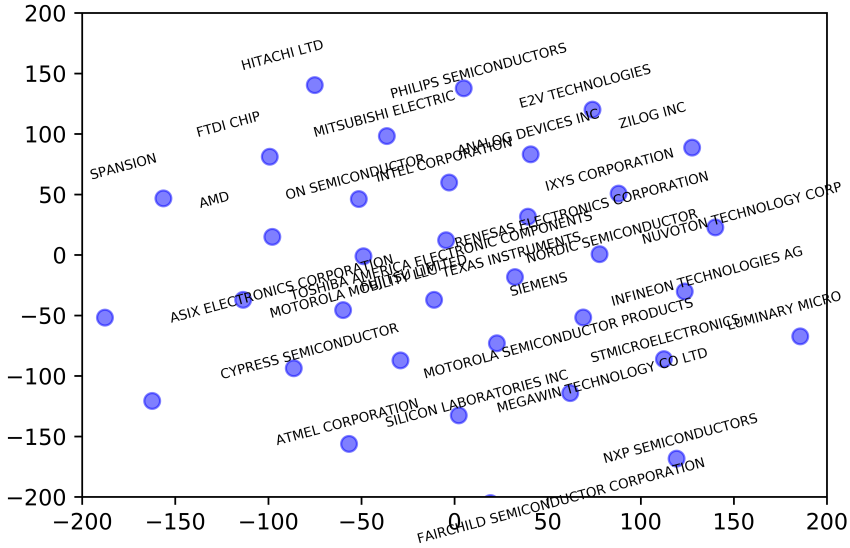

Figure 4.1: 2-D relative location of embedding vectors for different "Manufacturer"

ELECTRONICS CORPORATION", "ATMEL CORPORATION", "CYPRESS SEMICONDUCTOR", and et al. First, we map each of these categories to an unique integer ranging from 1 all the way to the total number of the categories. The only requirement is that the mapping is unique and one-to-one. Similarly, 0 is used for all the cases with missing data. $(N + 1)$ mappings could be obtained for N categories in the data set. For each of these integers, we then map it to an unknown vector with arbitrary length L (some instructions can be followed to find an appropriate L), which is called the embedding vector. As a result, instead of having $(N + 1)$ categories, we have $(N + 1)$ vectors representing them. A 2-D plot using t-SNE transformation of the embedding vectors learned by the DNN (described in the following sections) for "Manufacturer" is shown in Figure 4.1. The relative location of all the different categories is a much stronger indicator than traditional ordered number or one-hot transformation, which could be proved latter.

A more complex approach goes for the unstructured features. Using a series number of an electronic component for example, the feature "Mfr Part Number" has samples like "MAX231MJD", "XC2C512-10FGG324I", "PIC18F46K80-E/PT", and et al. There are thousands of different "Mfr Part Number" for this serial feature. First, we remove all the

special symbols and split the remaining part of each "Mfr Part Number" into a series of single digits. Second, similar with the first step for categorical data, we put all the single digits in a set and map each of these digits to an unique integer ranging from 1 all the way to the total number of the digits throughout all the samples for this serial feature. Third, each original value is replaced by a integer series with the same length. For example, "MAX231MJD" might be replaced by "1-3-21-48-23-22-14-6-58" based on how the alphabets and numbers are mapped to integers. As "Mfr Part Number" could have different lengths, the maximum length S_{max} is used to generate a vector of size S_{max} using the corresponding integer series followed by 0 padding at the empty entries. Therefore, series of size S_{max} could be obtained where all 0 entries are used for missing data. Again, for each of the integers, we then map it to an unknown vector with arbitrary length L, which is also called the embedding vector. Finally, each of all the "Mfr Part Number" series is able to be transformed into a embedding matrix with S_{max} rows and L columns, where each row represents a digit in a specific series.

For unstructured features with texts, we split a sentence into a series of words rather than split a series into digits, and all the rest steps are the same with the serial features. Additionally efforts using more advanced NLP techniques are needed, which are not covered in detail for the current study.

4.2 Proposed Neural Networks

The core structure of the modern Neural Networks can be seen as multi-layer addition of linear combination and nonlinear activation functions, which are applied from the input layer all the way to the output layer through hidden layers. A basic example with two hidden layers is shown in Figure 4.2. For the linear combination, a matrix of weights $\mathbf{W}$ and a vector of biases $\mathbf{b}$ are utilized as the unknown parameters to linearly transform the output vector (post-activation values) $\mathbf{x}$ of neurons at one layer to the input vector (pre-activation values) $\mathbf{z}$ of neurons at the next layer. A special case is used for the original input layer, where no activation functions are applied and the original values are directly treated as the "post-activation values". The number of hidden layers and the number of neurons in each

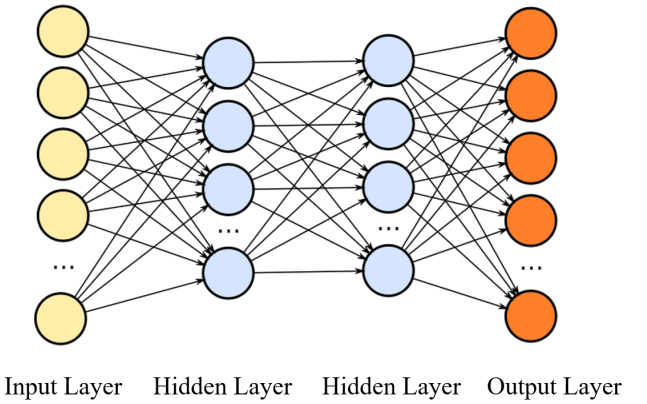

Figure 4.2: An example Multi-Layer Neural Network

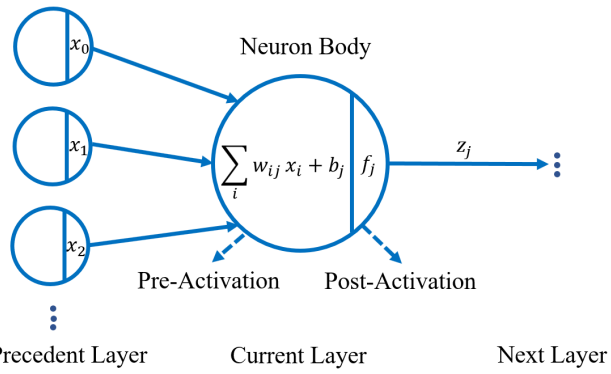

Figure 4.3: Detailed structure of a typical neuron

hidden layers are arbitrary and could be tuned to optimize the prediction performance of the Multi-Layer Neural Network.

A more detailed explanation of each neuron is presented in Figure 4.3. A neuron contains a pre-activation value and a post-activation one. As defined in Equation 4.1, linear transformation $\sum_i w_{ij}x_i + b_j$ is used to compute the pre-activation value of the neuron j in the current layer, which is then transformed by the (nonlinear) activation function f_j. The nonlinear activation is aimed to transform a given variable to its desired domain, such as $(-1, 1)$ for the hyperbolic tangent function, $(0, 1)$ for the sigmoid function, $[0, x]$ for the rectified linear unit (ReLU) function and a series of probabilities of different categories for the softmax function.

$$z_j = f_j(\sum_i \omega_{ij}x_i + b_j) \tag{4.1}$$

where x_i denotes each element of the post-activation values $\mathbf{x}$ of a precedent layer (or the original inputs); w_{ij} represents the element of the weight matrix $\mathbf{W}$ associated with x_i in the precedent layer and z_j in the current layer; b_j is the bias term added to z_j after the linear transformation.

The overall structure of the DNN model proposed in this project is shown in Figure 4.4. The key properties of this DNN model are described as follows:

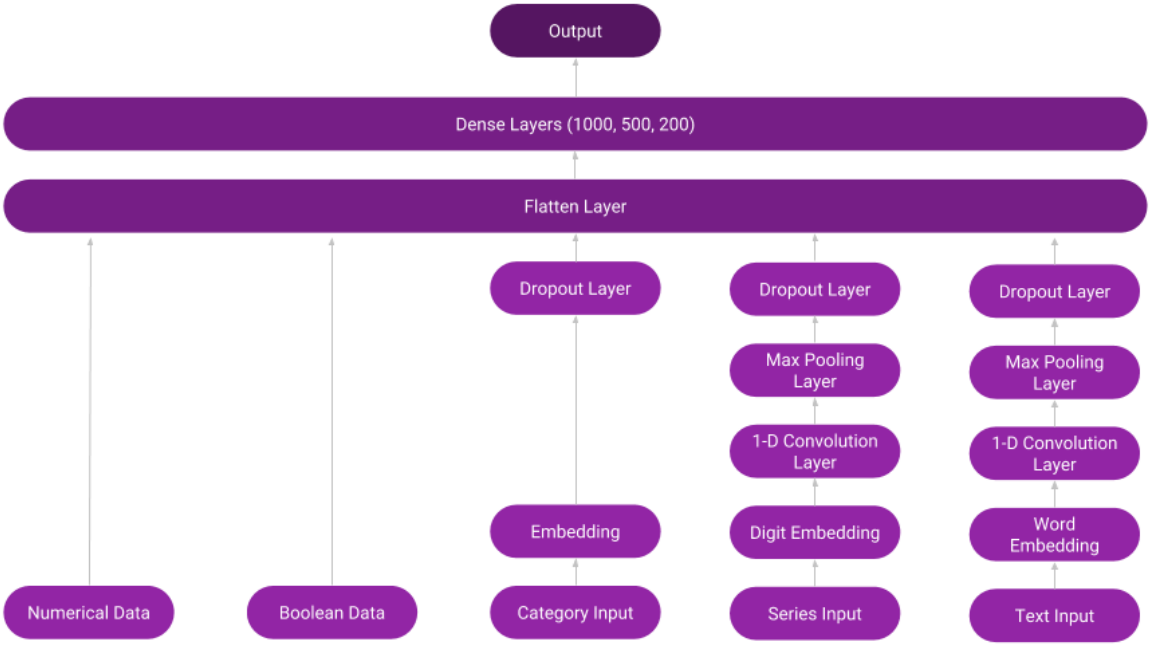

Figure 4.4: Structure of the proposed DNN

1) The embedding layers are used tor search for the appropriate embedding vectors for the categorical and unstructured features. The vector sizes for the categorical features are chosen to be close to the square roots of the number of categories in a certain feature, while the vector sizes of the words in the texts and alphabets/numbers in the series numbers are fixed at 15 and 8 respectively after a series of parameter tuning. The embedding vector representing missing values is padded at the end if the length of the texts or series numbers is not matched with the desired length.

2) The 1-D CNN layers (See Section 4.3 for detailed introduction) are applied to scan and filter certain local information in these embedding vectors. The filter size of the text and series number vectors are tested within a series of $2, 3, 4, 5$ and 3 is finally selected based on the performance. A total of 10 filters are used for both text and series number embedding vectors.

3) The max pooling layers are employed right after the 1-D CNN layers to consolidate the extract useful information by them.

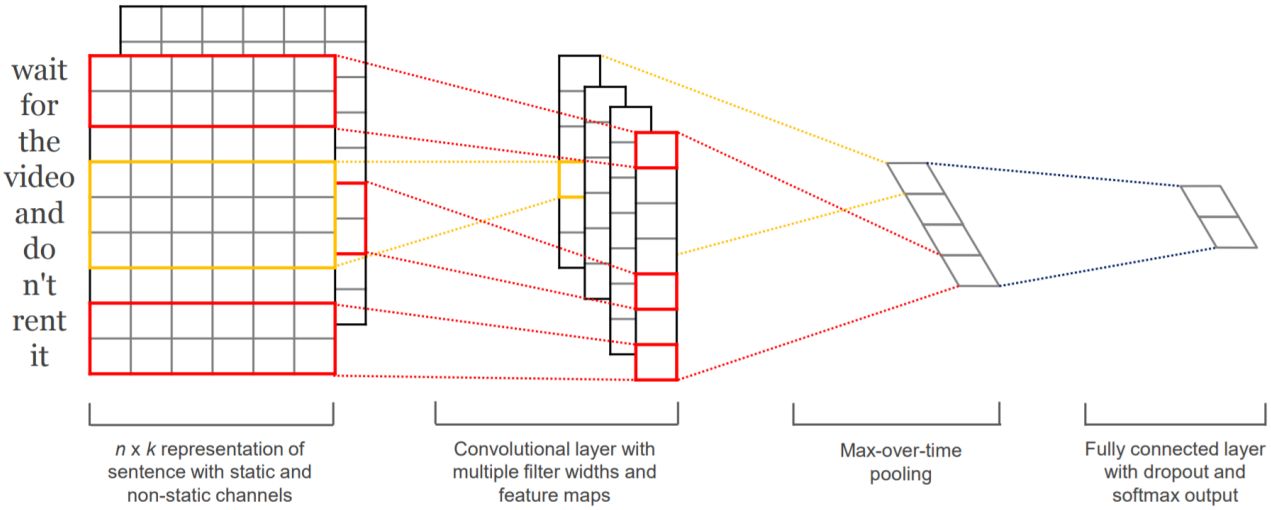

Figure 4.5: Example CNN used for sentence classification by Kim

4) The flatten layer is constructed to expand the pooling layers and concatenate all different types of information.

5) Finally, a total of 3 fully-connected layers are used, where each layer has 1000, 500 and 200 neurons respectively. The dropout rate of 10% is adopted to reduce potential overfitting.

In such a way, the constructed DNN is able to make better use of the categorical features and even take advantage of unstructured features that cannot be directly used in all the previous models mentioned in Chapter 3.

4.3 Conventional Neural Networks

Conventional Neural Networks (CNN) has emerged as a powerful tool to automatically and systematically learn mid- to high-level matrix or tensor representations for input data, especially in the areas of computer vision [31, 32, 33, 34, 35]. More recently, CNN has also been used in the field of NLP, for example sentence classification [36].

In the CNN, each layers weights are stored in a number of feature maps. Such a map is also called a channel and is obtained by a filter operating on the input or fully-connected variables, which are also organized into a number of feature maps. Each filter is a locally

weighted summation, plus a bias, followed by a nonlinear transformation. Figure 4.5 [36] is used here for illustration. The input is a sentence with n words, which are mapped to a $n \times k$ matrix following the feature embedding introduced earlier. CNN layers with multiple filter widths and feature maps are then used to extract the location patterns within the embedding matrix and generate new variable vectors, which are passed through the max pooling layers to concatenate pattern information and reduce dimension. The softmax output is used for the final layer in a classification problem, which is not necessary for the regression problem in the current study.

Although only CNN is used in the current study, other networks such as Recurrent Neural Networks (RNN) and their extension Long Short-Term Memory (LSTM) networks could also be tested in the future work.

4.4 Performance Evaluation and Discussion

The RF, XGB and DNN models are all retrained using the entire raw dataset with around 1 million samples. A total of round 80000 real traffic queries over a recent time period that are not contained in the training dataset are used to test the final performance for these models using the two metrics RMSE and ROPV defined in Section 3.2.1. As the overall performance of RF and XGB models are very close, only the optimal RF model from Chapter 3 is used for the following comparison. Moreover, based on the real traffic queries, the customers could sometimes only provide the "Mfr Part Number", which is an unstructured feature and cannot be directly handled by the RF model. As a result, the prediction coverage rate defined by the ratio of predictable queries out of all the queries is also adopted as a performance metric.

For the RF model, when only unstructured features such as "Mfr Part Number" is given the real traffic queries, it has to look for the exact match of the given "Mfr Part Number" in the previous records (training dataset) and use all the other features associated with it as the features to predict the desired unit price. Because the previous records could by no means cover all the "Mfr Part Number" queried by customers, the RF model cannot return a predicted unit price in these cases, ending up with a prediction coverage rate of around

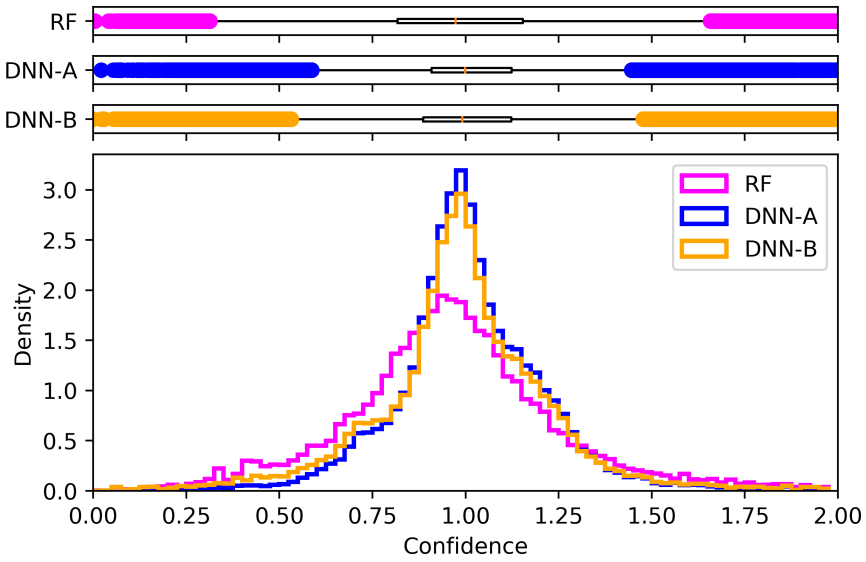

Figure 4.6: ROPV comparison between the RF and DNN models

90% for the real traffic queries tested here. On the contrary, the DNN model could achieve 100% prediction coverage rate by mapping a given"Mfr Part Number" to a high dimensional embedding space and trying to find a similar pattern captured through the convolutional layers. In such a way, it can not only distinguish even the minor difference between"Mfr Part Number" features but also make prediction for any"Mfr Part Number" not recorded in the prepared data set using the data with similar patterns.

Figure 4.6 shows the comparison of the performance in terms of the ROPV distribution and prediction intervals using RF and DNN models. The DNN-A model gives the prediction results of the queries that the RF model is able to return, while the DNN-B model is for all the queries. As we can see that the DNN-A model significantly outperforms the RF model for both of the two performance metrics. The performance of the DNN-B model is very close to the DNN-A model, proving that the DNN model is able to predict reasonable unit price even when the given "Mfr Part Number" is not available in the current records used for training. Overall, the RF model gives the RMSE of 0.41 while the DNN-A and DNN-B models achieve the RMSE of 0.33 and 0.34 respectively. Moreover, the DNN-A model reduces the 50%, 80% and 90% ROPV PIs by 36%, 44% and 60% respectively with the same

prediction coverage rate, while the DNN-B model shortens the three ROPV PIs by 30%, 35% and 52% respectively with a 100% prediction coverage rate. It should be noted that these results are not comparable with those in Chapter 3 as totally different testing datasets are used.

Table 4.1: Performance comparison using the PIs of ROPV from the RF and DNN models

Model	*50% PI*	*80% PI*	*90% PI*	*50% PI Size*	*80% PI Size*	*90% PI Size*
RF	[0.818, 1.154]	[0.639, 1.491]	[0.497, 2.189]	0.336	0.852	1.692
DNN-A	[0.909, 1.123]	[0.777, 1.257]	[0.690, 1.362]	0.215	0.480	0.672
DNN-B	[0.886, 1.123]	[0.714, 1.269]	[0.600, 1.413]	0.236	0.556	0.813

Base on the historical purchase data and common sense, the unit price of a given electronic component should not rise (if not drop) when the purchase quantity in the order increases, i.e., a monotonic decreasing relationship exists for the unit price and quantity. Figure 4.7 and Figure 4.8 show some example comparisons of the price-quantity relationship constructed using the RF and DNN models respectively, where the unit price is normalized by the price for a single item purchase (the maximum unit price possible). While both models directly learn from the given data without being explicitly enforced to any predefined monotonic function, the DNN model has significant better modeling of the relationship in terms of the monotonic decreasing as well as smoothing. As we can see, although not common, the DNN model could have some predictions that violate the monotonic decreasing relationship, suggesting that a potential improvement is needed in the future to fully address this issue.

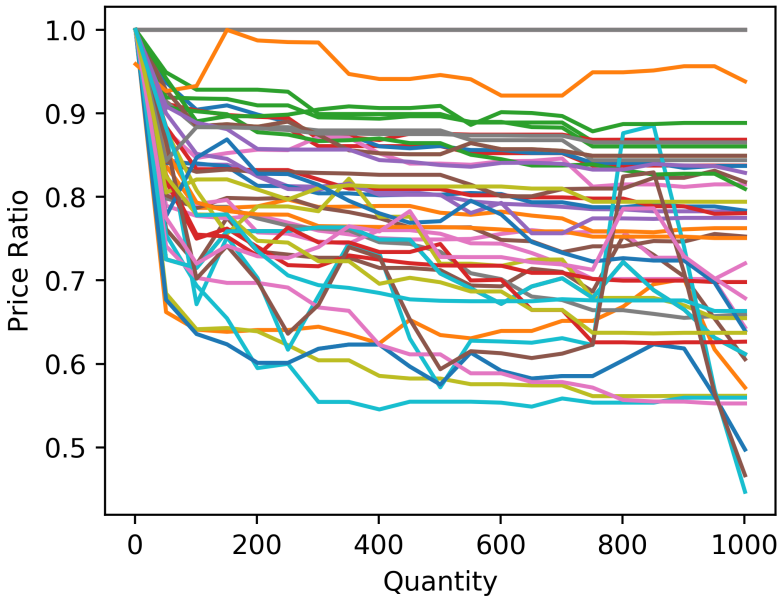

Figure 4.7: Price-quantity relationship by the RF model[1]

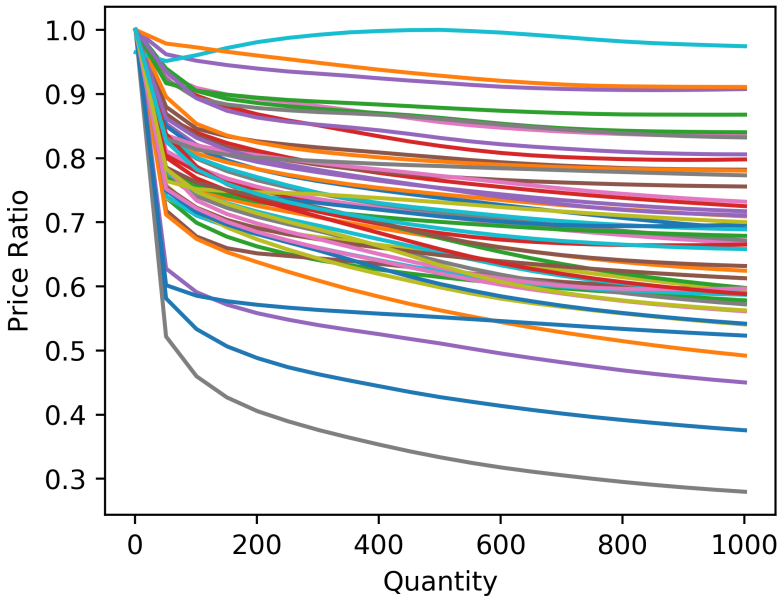

Figure 4.8: Price-quantity relationship by the DNN model[1]

[1] Colors are used only to distinguish different tests of electronic components

CHAPTER 5

Summary and Conclusions

This study investigates the relationship of the unit price of a given electronic component and its specifications and purchase details using the massive historical purchasing data for various electronic components at Supplyframe, Inc. The developed models and useful findings could provide strong data-driven support for the decision-making process for the customers.

To achieve a better understanding and using of the available data, Exploratory Data Analysis (EDA) is first carried out, followed by feature engineering such as feature selection, feature transformation and missing data imputation. A series of classical statistical and ML models are then explored to capture the underlined relationship. Starting from the baseline Naive model using mean estimation, a series of models are trained, tested and evaluated, including Ordinary Least Squares (OLS), Least Absolute Shrinkage and Selection Operator (LASSO), Support Vector Machine (SVM), k-Nearest Neighbors (k-NN), Random Forests (RF) and Extreme Gradient Boosting (XGB). In additional to the widely-used Root Mean Squared Error (RMSE) metric for regression problems, the imperial Prediction Interval (PI) of the Ratios of Observed and Predicted Values (ROPV) is defined based on the more practical business scenarios. k-fold Cross Validation (CV) is applied to tune the model parameters and select the optimal predictive model. Feature embedding technique is Incorporated into the 1-D Convolutional Neural Networks (CNN) to make better use of the unstructured features such as text and series numbers. Finally the DNN model combines all the numerical, categorical and unstructured data is established.

The results show that the nonlinear models such as SVM, k-NN, RF and XGB significantly outperform the Naive baseline model and OLS model, indicating strong nonlinear relationship between the electronic component unit prices and their specifications and pur-

chase details. It is observed that RF and XGB models have considerable overfitting issues over the SVM and k-NN models; however, they still achieve the optimal performance in terms of the least RMSE and smallest PIs of ROPV in the CV procedures. Thus, they are deemed as the optimal models among the classical statistical and ML models.

The proposed DNN model is constructed as follows: 1) the embedding layers searching for the appropriate embedding vectors for the categorical and unstructured features; 2) the 1-D convolutional layers scanning and filtering certain local information in these embedding vectors; 3) the max pooling consolidates the extracted information; 4) the flatten layers concatenating all different types of information and 5) the fully-connected layers with certain dropout rates making the final predictions.

Due to the fact that some "Mfr Part Number" in the real traffic queries are not recorded in the previous training dataset, the RF and XGB models are not able to make predictions, ending up with a prediction coverage rate of around 90%. On the contrary, the DNN model could achieve 100% prediction coverage rate and even distinguish the minor difference between"Mfr Part Number" features to give slightly different price predictions. The RF and XGB models give very similar RMSEs around 0.41 on the final testing dataset for the 90% real traffic queries, while the DNN model reduces it to as low as 0.33 and even keeps the similar performance for the 10% queries with new "Mfr Part Number" features and cannot be predicted by the RF and XGB models.

The DNN model reduces the 50%, 80% and 90% ROPV CIs by 36%, 44% and 60% respectively for the same prediction coverage rate with RF and XGB, while decreases the three ROPV CIs by 30%, 35% and 52% respectively with a 100% prediction coverage rate. Moreover, the DNN model better captures the non-strict monotonic decreasing relationship for the unit price and quantity, i.e., the unit price of a given electronic component should not rise (if not drop) when the purchase quantity in the order increases.

A limitation of the current study is that the classical statistical and ML models use only the structured data (numerical and categorical features) while the DNN model incorporates extra information from the unstructured data. The learned embedding vectors from

DNN could be added to the classical statistical and ML models to further compare their performance with the DNN model.

An interesting future work could be the measurement of the feature importance for the DNN model such that effective feature selection could also be performed as the classical statistical and ML models. Another potential improvement is to incorporate the underlined monotonic decreasing relationship into the predictive models. Finally, only CNN is used for the DNN in the current study, but other networks such as Recurrent Neural Networks (RNN) and their extension Long Short-Term Memory (LSTM) networks could also be tested in the future work.

REFERENCES

[1] Robert Tibshirani. Regression Shrinkage and Selection via the Lasso, 1996.

[2] Vladimir Naumovich. Vapnik. *The nature of statistical learning theory*. Springer, 1995.

[3] Vladimir Vapnik and Steven E Golowich. Support Vector Method for Function Approximation, Regression Estimation, and Signal Processing·.

[4] Bernhard. Scholkopf and Alexander J. Smola. *Learning with kernels : support vector machines, regularization, optimization, and beyond*. MIT Press, 2002.

[5] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[6] N. S. Altman. An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. *The American Statistician*, 46(3):175–185, 8 1992.

[7] William S. Cleveland and Susan J. Devlin. Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting. *Journal of the American Statistical Association*, 83(403):596–610, 9 1988.

[8] Friedman J. Stone C. J. & Olshen R. A. Breiman, L. *Classification and regression trees*. CRC press, 1984.

[9] Joseph Felsenstein. Confidence Limits on Phylogenies: An Approach Using the Bootstrap. *Evolution*, 39(4):783, 7 1985.

[10] Bradley Efron and Robert Tibshirani. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical science*, pages 54–75, 1986.

[11] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[12] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[13] Max Kuhn and Kjell Johnson. *Applied predictive modeling*. Springer, 2013.

[14] Leo Breiman. ARCING THE EDGE. Technical report.

[15] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[16] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2 2002.

[17] Llew Mason, Jonathan Baxter, Peter L Bartlett, and Marcus R Frean. Boosting algorithms as gradient descent. In *Advances in neural information processing systems*, pages 512–518, 2000.

[18] Llew Mason, Jonathan Baxter, Peter L Bartlett, and Marcus R Frean. Boosting algorithms as gradient descent. In *Advances in neural information processing systems*, pages 512–518, 2000.

[19] Yoav Freund, Robert E Schapire, and others. Experiments with a new boosting algorithm. In *icml*, volume 96, pages 148–156. Citeseer, 1996.

[20] Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[21] Tianqi Chen and Carlos Guestrin. XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, pages 785–794, New York, New York, USA, 2016. ACM Press.

[22] Dehua Wang, Yang Zhang, and Yi Zhao. LightGBM: an effective miRNA classification method in breast cancer patients. In *Proceedings of the 2017 International Conference on Computational Biology and Bioinformatics*, pages 7–11. ACM, 2017.

[23] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc., 2017.

[24] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. CatBoost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363*, 2018.

[25] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. CatBoost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, pages 6639–6649, 2018.

[26] Ron Kohavi and others. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, volume 14, pages 1137–1145, San Mateo, CA, 1995.

[27] Yoshua Bengio, Rjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[28] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[29] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[30] Cheng Guo and Felix Berkhahn. Entity embeddings of categorical variables. *arXiv preprint arXiv:1604.06737*, 2016.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[32] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[33] Pierre Sermanet, David Eigen, Xiang Zhang, Michal Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[34] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[35] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[36] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.