CS334 Final Project Report

George Saito, Qiao Lin

Abstract:

Regression-based machine learning algorithms have become an important tool for predicting numeric target such as price. This project gives an experimental analysis on how we implement several machine learning algorithms on a Spanish rail tickets dataset to predict the ticket price target. By giving evaluation and comparison of each algorithm, the report shows which one is the best fit for our project and how well it performs.

Introduction:

Our project analyzes the dataset “Spanish High-Speed Rail ticket pricing-Renfe” provided on Kaggle. Renfe is the national rail operator in Spain which provides Spanish high-speed train service and tickets pricing monitoring systems. The company operates passenger trains on the Spanish national rail network. Among the trains that Renfe operates, are the AVE trains which are modern and comfortable. Users can set up alarms using email, choosing an origin and destination, time, etc on the website. This dataset is created by a professional team that scrapes tickets price data and stores them in a database periodically.

The row data has about 258 thousand row with 9 variables, besides one numeric variable, all other variables are categorical. Among these features, we choose the numeric variable “price” as our target variable. The reason is price is sensitive to other features such as start date and train type, as it changes constantly according to other variables throughout the dataset. Based on the reflective nature of price, we decide to use prediction machine learning methods to implement a price prediction project.

This project could be very helpful to potential Renfe user who is about to start a journey. We found some Renfe users complaining on the official website about the unstable and unpredictable price which caused them trouble during their trip(tripadvisor.com). As mentioned before, the ticket price is sensitive to the season, day or time, etc. Machine learning provides us a way to solve this problem. If provided a user’s travel information like month, day, destination and expected end date, the price will be restricted to a specific interval. We can also give suggestions like which certain

train types to take to save more money. By training a solid model from this dataset using what we learned in this course, we can give this project a high real-world value.

Background:

Previous Kaggle work “A Very Extensive Renfe analysis” focused on time-related features. They analyzed the distribution of start date and end date and concluded that most journeys are completed within a single day. Similarly, they analyze travel time and weekday. Finally, they conclude that most journeys ended within a day and the Turista class trains are mostly chosen in every journey by Spanish. However, most comparisons are made within the same feature. Although they did include some analysis on price, such as “The Turista train class was cheaper than all classes”, the results were not used efficiently to give an informative and predictive result on price variable, which is our primary concern. (Extensive Spanish High-Speed Rail tickets pricing data analysis, Kaggle) In other words, we are the first team to implement a machine-learning-based predictive model on this dataset. After feature extraction and preprocessing, we include basic machine learning models, KNN, Decision Tree, Naive Bayes and SVM as well as advanced random forest models, ensemble methods and Neural Network. Each of them has their own merits and demerits. We try to find patterns and give a thorough comparison of each model before reaching a conclusion.

Methods:

We first try several regression-oriented machine learning methods including KNN regressor, Support Vector Regression(SVR) and regression trees. We evaluate our model through the Root Mean Square Error (RMSE).

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

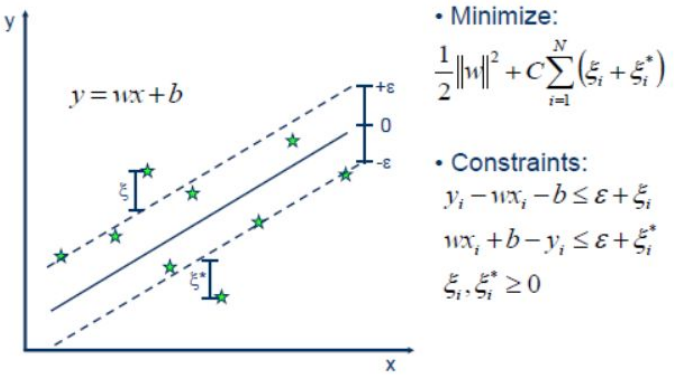
1.KNN regressor

The algorithm measures the feature similarity of a new point to existing points and assign value accordingly. Depends on k, the final prediction will be the average of k nearest points' value. Although KNN regressor is highly interpretive and adaptive, it is sensitive to preprocessing and dimensionality. We use Euclidean distance to measure the distance in this project.

2.Regression tree

Regression Tree is a variant of Decision Tree which gives a prediction of value instead of classification. The algorithm split at where the sum of squared deviations of the mean in the two partitions is minimized. Since it allows the input to be a mixture of numeric and categorical value and deal well with dependent features, it's a good fit for our project.

3.Support Vector Regression



(www.saedsayad.com)

Support vector regression is using the same principle as SVM except for it is used for prediction. One difference is when SVR is looking for maximum margin, it has a tolerance(epsilon) for error, which makes the prediction easier. SVR usually perform well for high-dimensional space in prediction, thus we want to include it in our models.

4. Random Forest Regressor

Random Forest is a meta learner based on regression tree. It uses bootstrap with replacement from all data and builds a single regression tree on each subsample. By fitting the model on many trees, Random Forest is able to increase prediction accuracy while at the same time avoid overfitting. Due to its high performance in general and the fitness with our data, we want to include the Random Forest regressor in our models.

5. XGBoost Regressor

We found other Kaggle project using XGBoost regressor to predict price and generated good results. Since XGBoost adjusts weight constantly based on previous results, it performs well generally. We also want to see its performance on our dataset.

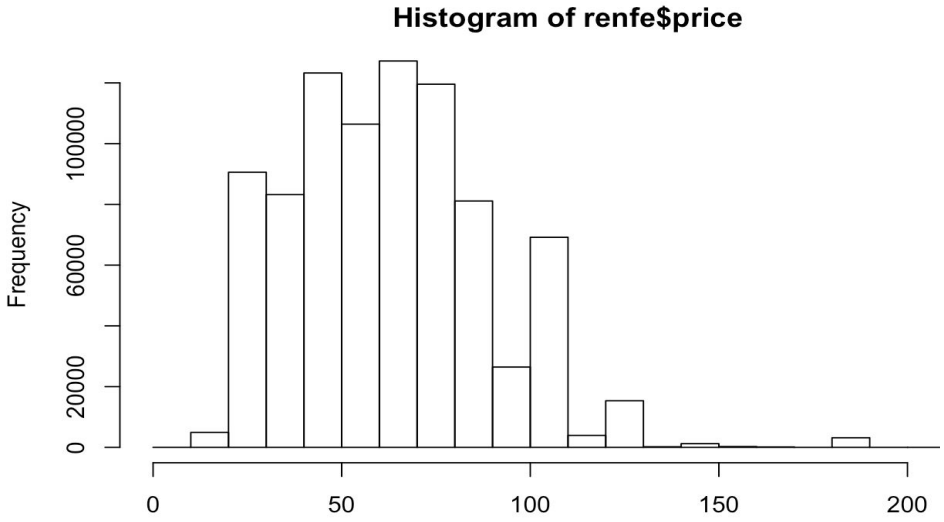
6. LightGBM

LightGBM is a gradient boosting framework that uses tree-based learning algorithms. (LightGBM documentation). According to documentation, it tends to have a better performance on a large-scale dataset. Since our original dataset is on the scale of millions, we think LightGBM might be fit for us.

Experiments/Results:

1.Data description:

The original dataset has 257977 rows and 9 features. “insert_date” represents the date and time when the price was collected and written in the database; “origin” and “destination” which indicates the origin and destination city of a journey; “start_date” and “end_date” indicate departure and arrival time of a journey; “train_type” which indicated train service name; “price” which is the target ticket price; “train_class” indicates the ticket class, including tourist, business, etc; And finally “fare” which stands for ticket fare such as round trip. The distribution of target price is approximately normal:



Before we get started, we first do feature engineering and preprocessing to make sure machine learning models predict efficiently. The raw dataset maybe containing too heterogeneous features, noise, dependent or redundant factors that could harm the prediction accuracy or make the results hard to interpret. To avoid those mistakes, we did the preprocessing and features engineering carefully in this project.

2. Heterogeneity of Data:

First, we observe that there were missing data entries for train_class and fare variables, the way we handle this is to replace them with NA's and simply drop them during preprocessing. While this could be information losing if the proportion of missing data is high, in our case it is less than 5%, thus removal is our choose. The next thing to consider is how to deal with categorical features, which is our primary feature data type. Simply giving each category a numerical label will lead to bias for distance measuring algorithms like KNN, thus we choose to use one-hot encoding and create dummy variables for all categorical features. The drawback is by including dummy variables, we seriously increase the feature dimension of our dataset, which may contain redundant features.

3. Feature Extraction:

Besides the dummy variables generated by one-hot encoding, the original features provided are very limited. Through feature extraction, we are able to create new features that are more useful and replace the original features with them. Firstly, we found that the origin and destination variables are very hard to interpret as separate features, as 50% of origins and destinations are Madrid, however, it makes no sense for a journey to start and end at the same place. One-hot encoding on these two features might also ruin algorithms like KNN. Two points with the same metric distance as another point might be two different routes that significantly influence the ticket price. As a result, we transform these two features into one feature “Route” which is more interpretable and avoid the confusion of the nonexisting route. Secondly, we observed that all time-related features in the original dataset were written in timestamp format. This format contains three important features: “Month”, “Day” and “Time”. Unix timestamp, KSP date format and separate into new features are three common ways to deal with timestamp data. Unix timestamp chooses an initial time and measures how long time has passed since that time, however, the results also depend heavily on how we choose the initial time. KSP date format generates a single numeric value that combine all three features.(towarddatascience.com)

$$KSP\_date = year + \frac{days\_from\_1\_Jan - 0.5}{365 + 1\_if\_leap\_year}$$

We choose to separate timestamp data into three features as month, date and time are all important features to be considered when determining the ticket price. After preprocessing, we split the dataset into training and test set with the proportion of 7:3.

4. Dimension Reduction:

If our dataset has a high dimension but our problem only depends on a small subset of the feature space, it could potentially harm the accuracy of the machine learning model. For KNN, as the data become more sparse, the variance increase and a biased prediction could be given. Moreover, a high dimensionality makes the results hard to interpret and significantly increase the time and storage cost of any model. To avoid the curse of dimensionality, we choose to use PCA(Principal Component Analysis) and NMF(Nonnegative Matrix Factorization) to remove irrelevant features while at the same time give a comparison between these two methods. We also run all models on the dataset without dimension reduction and see the effect of dimension reduction on models.

Surprisingly**,** the first principal component explains more than 95% of the variance. The reason might be this is a highly feature correlated dataset. While for NMF 17 principal features are left after dimensional reduction.

Overall, the preprocessing improved the quality of data by removing null values, reduce redundancy, reduced bias, and feature dimension.

5. Modeling choices:

We tune the parameters for each model by using one of the cross-validation methods, “GridSearchCV” that enables us to input a list of parameters to tune for each feature. Then it compares the CV score of every possible combination of parameters to determine what is the optimum one. However, we still have the risk of overfitting on the training set. To solve this, we measured the RMSE on the test set for the same time. If the RMSE score for the test set begins to increase, then probably overfitting is happening and we would adjust the parameters.

6. Empirical results and comparisons:

For KNN:

	Optimum K	trainScore	testScore	RMSE of testset
Without Dimension Reduction	7	0.9735	0.9702	4.46
PCA	5	0.9732	0.9712	4.38
NMF	7	0.9732	0.9711	4.39

The "trainScore" and "testScore" is measured by the coefficient of determination R^2 of the prediction. The coefficient R^2 is defined as (1 - u/v), where u is the residual sum of squares ((y_true - y_pred) ^ 2) and v is the total sum of squares ((y_true - y_true.mean()) ^ 2). (Scikit Learn Documentation). We also do not observe overfitting on training set since the RMSE on test set doesn't show an increasing trend at optimum K.

For SVR:

	trainScore	testScore	RMSE of testset
Without Dimension Reduction	0.844	0.842	10.26
PCA	0.220	0.220	22.79
NMF	0.584	0.585	16.63

For Regression Tree:

	max_depth	min_samples_leaf	trainScore	testScore	RMSE of testset
Without	9	2	0.916	0.914	7.58

Dimension Reduction					
PCA	9	2	0.939	0.937	6.50
NMF	9	2	0.799	0.791	11.79

For Random Forest:

	Optimum number of estimators	trainScore	testScore	RMSE of testset
Without Dimension Reduction	50	0.9779	0.9770	3.91
PCA	80	0.9778	0.9769	3.92
NMF	60	0.9739	0.9719	4.32

For XGBoost:

	trainScore	testScore	RMSE of testset
Without Dimension Reduction	0.9770	0.9760	3.99
PCA	0.9390	0.9348	6.59
NMF	0.9777	0.9769	3.93

The tuned parameters for XGBoost is {colsample_bytree: 0.4, learning_rate: 0.5, max_depth: 7, min_child_weight: .5, n_estimators: 80}. The tuning method is Grid Search cross validation. We use the trend of RMSE on testset to prevent overfitting.

Discussion:

We can see from the tables that each model fits pretty well into our dataset. However, there do exist certain patterns.

Considering the effect of dimension reduction, we observe that there exists a gap between basic models and ensemble models. For KNN, Regression Tree and SVR, PCA tends to increase the performance on test set by reducing RMSE. These models, without ensemble, are more sensitive to noisy and highly correlated features, thus a dimension reduction is necessary. On the other side, both PCA and NMF is not significantly performing well for Random Forest and XGBoost. We think the reason is that due to the ensemble nature of these algorithms, they are resistant to noise and colinearity. These models are also hard to suffer from overfitting. Based on these reasons, a dataset with less preprocessing or without dimension reduction may be a better suit for ensemble algorithms. Without dropping collinear features, more information are provided to the meta learner and therefore easier to reach a higher performance.

Among PCA and NMF, we conclude that PCA is generally performing well based on RMSE on test set. One reason might be PCA works better for high dimension dataset since it is less sensitive to original features lose, while NMF requires regularization and induces sparsity as learned in class. Especially after one-hot encoding, our features become more sparse and may cause NMF to lose more accuracy.

We also observe the extremely low train and test score for SVR. We think the reason also lies in one-hot encoding for categorical features. Since one-hot makes features more sparse, linear regressor like SVR may suffer from this sparsity and give a low prediction score.

In conclusion, we find that Random Forest is the most suited for our dataset since it gives a consistently high performance on both test set score and test set RMSE. It indicates the strong predictive power of this algorithm as well as its high resistance to overfitting and noise.

Contributions:

George Saito: Decide on which dataset to use. Implement most coding missions, generate train and test accuracy scores and most python output. Provide insight on which algorithms to use and which to discard based on the output.

Qiao Lin: Provide insight on how to choose dataset. Decide on what algorithms and methods to include in the project. Compare and analysis on different models based on the output. Provide insight on the overall structure and theory base for the project. Write the report.

References:

Google Drive Link for our codes:
https://drive.google.com/open?id=1ELRz6JqB-KXRMmRcq-0urLj-gI4tCVxx

A Very Extensive Renfe analysis, Extensive Spanish High Speed Rail tickets pricing data analysis. Kaggle, 2019.
https://www.kaggle.com/ratan123/a-very-extensive-renfe-analysis#Extensive-Spanish-High-Speed-Rail-tickets-pricing-data-analysis

All you want to know about preprocessing: Data preparation, Towards Data Science. May 29, Maksym Balatsko.
https://towardsdatascience.com/all-you-want-to-know-about-preprocessing-data-preparation-b6c2866071d4

Renfe tickets Madrid - Barcelona expensive! Tripadvisor,2016.
https://www.tripadvisor.com/ShowTopic-g187497-i44-k9498410-Renfe_tickets_Madrid_Barcelona_expensive-Barcelona_Catalonia.html

Scikit Learn Documentation, sklearn.neighbours.KNeighborsRegressor
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html

Support Vector Machine - Regression (SVR)https://www.saedsayad.com/support_vector_machine_reg.htm

LightGBM Documentation. Welcome to LightGBM documentation.
https://lightgbm.readthedocs.io/en/latest/