Table of Contents

Importing and Describing the Data

Import data

We loaded the data from 3 .csv files provided by Kaggle for the competition. We then separated the data into variables, y, which contains the target variable and X, which contains the explanatory variables. We excluded the rows in training data where store is closed that is the open flag is 0 and also removed rows where sales is less than or equal to zero as they are of no help and will only bias the data while training.

Split the data

The training set consists of 85 percent of the data while the validation set contains 15 percent of the data. Test data is already provided by Kaggle for evaluation.

Normalization

We normalized the data using z-score normalization for all methods. We normalized after splitting the data into train and validation. We fitted the model on the training set and transformed both validation data and test data.

Feature Selection

We used a linear support vector regression with a C of 0.1 to reduce our feature space. The SVR was fit on the entire data set and nested within a SelectFromModel function. The function selected 10 of our 27 variables. We used feature selection in our Linear SVM, Linear Regression, Artificial Neural Network and LSTM models. Other models like XGBoost and Random Forest are trained without any feature selection.

Parameter Tuning

We tried using K-Fold nested cross validation in GridSearchCV but since the data is very huge, the models were not able to run even in one day and also gave almost no improvement on the leaderboard. So, we decided to move forward with holdout validation by splitting the data in training and validation and checking the model’s performance on validation data to assess overfitting or underfitting. However, for simple models like Linear Regression, Ridge and Lasso, we used GridSearchCV to get best value of parameters.

Evaluation

To evaluate different models, we calculated the root mean squared percentage error for each model as this is the criteria that Kaggle was also using to evaluate the model performance. We compared different model using this error metric and selected the one which provided least root mean squared percentage error.

Results and Conclusion/Discussion

XGBoost Model (Decision Tree Ensemble Model)

Since, we are not able to run KFold cross validation and GridSearchCV, we ran several iterations of the model with different values of “max_depth”, “colsample_bytree”, “min_child_weight” and “num_boost_round”. Based on the results of various iterations, the best parameters were “max_depth” of 10, “colsample_bytree” of 0.4, “min_child_weight” of 6 and “num_boost_round” equal to 1200 with an early stopping of 200.

This model performed really well on both train and validation data with **train rmspe of 0.086584** and **validation rmspe of 0.098752**.

This was our best model and using this model for submission on Kaggle, our **private score is 0.13002** with **private leaderboard rank of 1669** and **public score of 0.11833** with **public leaderboard rank of 1751**.

Random Forest (Decision Tree Ensemble Model)

Random forest model was fitted for different values of n_estimators which defines how many trees will be fitted. We used 20,50,100 and 150 trees but model performed almost equally for trees above 50. So our final model used 50 trees for prediction and we kept all other parameters as default. The model gave a validation rmspe of 0.144978. We tried blending the output of random forest model with XGBoost model but it gave no improvement. It seems like that XGBoost is already capturing all the variations that Random Forest is explaining.

Support Vector Machine

We tried fitting SVM with RBF kernel, but it took forever to run. So, we tried reducing the variables by fitting the scaled data using LinearSVR and then using “SelectFromModel” function for feature selection. This did not provided any significant difference in computation time so we ran LinearSVR with selected features. This model gave validation rmspe of 0.442174 which is very high compared to above two models. So, we didn’t use the output of this model for submission on Kaggle.

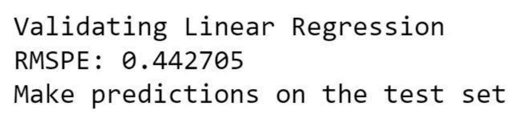
Linear Regression (Multiple, Lasso, Ridge)

The model was fit using nested cross validation. The hyperparameter space was set to possible

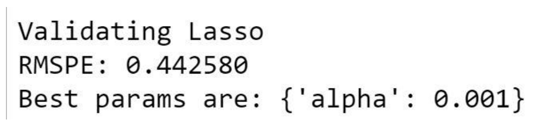
alpha values of [1e-3,1e-2,1e-1]. Scoring for grid search was customized to set it to RMSPE. The best linear regression model used ridge regularization. Compared to a model with no regularization and a

model with lasso regularization, the ridge regularization model performed the best with a

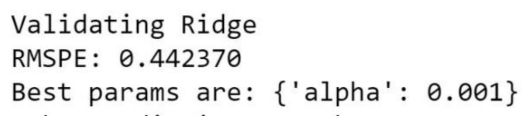
RMSPE of 0.442370 on the validation data which is very high compared to above two models. So, we didn’t use the output of this model for submission on Kaggle. The performance of the all the models is provided below:



```
Validating Linear Regression
RMSPE: 0.442705
Make predictions on the test set
```



```
Validating Lasso
RMSPE: 0.442580
Best params are: {'alpha': 0.001}
```



```
Validating Ridge
RMSPE: 0.442370
Best params are: {'alpha': 0.001}
```

Artificial Neural Network

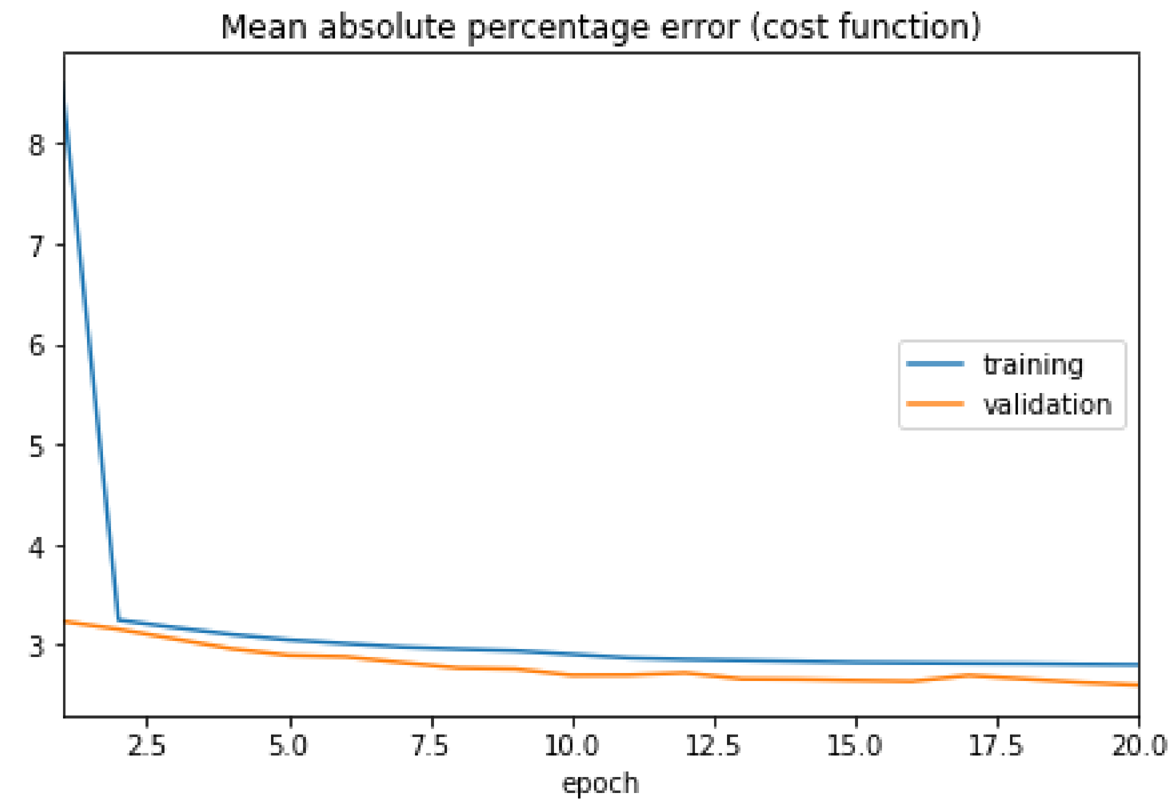
We considered the following options to tune our multi-layer perceptron model:

1. Number of layers
2. Number of neurons in each layer
3. Activation function: ReLu, Sigmoid, Tanh
4. Kernel initializer = Normal, Uniform
5. Optimizer: SGD, Adam
6. Epoch count and batch size

After trying many combinations of the above mentioned parameters, we came up with the final

model that have 5 hidden layers with layers having 50, 25, 10 and 1 neurons. For model

training, we used ReLu activation in all the layers with uniform initialization and adam optimizer optimizing custom function of RMSPE as loss metric. We also added two dropout layers after first and second dense layer.



After training the model for 20 epochs on batch size of 200, we got a RMSPE of 0.325285 on the validation data which is very high compared to above two models. So, we didn't use the output of this model for submission on Kaggle. Above is the loss graph for training and validation data corresponding to all epochs values.

Blend of LSTM and GRU with dense layers

We considered the following options to tune our LSTM model:

1. Number of LSTM and GRU layers

2. Number of neurons in each LSTM and GRU layer

3. Number of dense layers

4. Number of neurons in each dense layer

5. Activation function: ReLu, Sigmoid, Tanh

6. Kernel initializer = Normal, Uniform

7. Optimizer: SGD, Adam

8. Epoch count and batch size

9. Number of drop out layers and drop out percentage

After trying many combinations of the above mentioned parameters, we came up with the final model that have 5 hidden layers with layers having 150, 50, 25, 10 and 1 neurons. For model

training, we used ReLu activation in all the layers with uniform initialization and adam optimizer optimizing "mean_absolute_percentage_error" as loss metric. We also added two dropout layers after first and third dense layer.

After training the model for 20 epochs on batch size of 200, we got a R-squared value of 0.690

on the validation data. Above is the loss graph for training and validation data corresponding to all

epochs values.

SUMMARY

Below is the summary of the models we tried and their performance on validation data and Public and Private Leaderboard.