Does it worth to devote time on this customer?

Santander Customer Value Prediction Challenge

Recep Yusuf BEKCI

COMP 652 Poster Presentation

Problem

With the increase in choices for financial services, Santander Group wants to enhance their chances to be preferred by customers in order to maintain its competitiveness in the banking sector. At this point, the problem that we deal with tackling the provided data is identification of value of transactions for each potential customer. Thus, Santander Group aims using these predicted results in order to give an improved service consisting of personalization and customization.

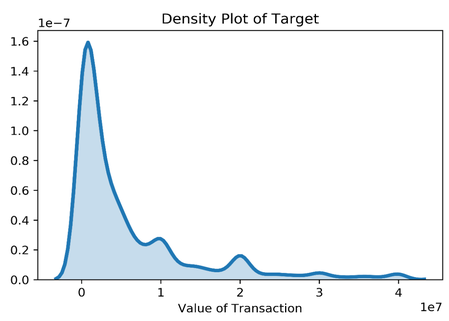
Kaggle uses root mean squared logarithmic error metric for the competition:

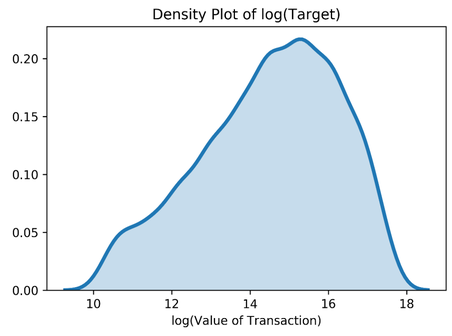
RMSLE = $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2}$ with n : total number of observations in the data set

p_i : prediction of target and a_i : the actual target for i

Explanatory Analysis of Data

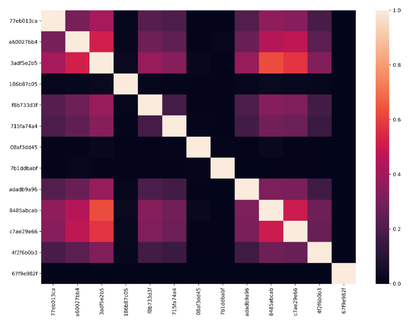
The dataset belongs to a Kaggle competition which was ended in 20 August 2018. The dataset is divided into train and test sets for the competition. The sizes are (4459, 4993) for training set and (49342, 4992) for the test set. Obviously, we have less samples in our training set. Moreover, the number of columns is more than the number of rows in training set. Our target variable is the values of transactions made by customers. So, in each row we have some information related to each transaction. However, the column names are masked so that we do not have any idea about their nature.
Here is the density plot of our target variable:

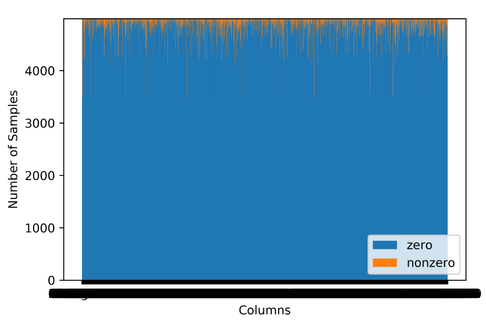




As a result, our target is negatively skewed with a reasonably long tail. In order to strengthen our analysis, we decided to use the target variable in log scale. The density of log transformation is more close to normal distribution.

Additionally, we look for correlations between predictors. It is hard to examine due to high number of predictors. We reduced the number with a threshold value considering the correlation value between predictors and target. So, we have the following heatmap for 13 predictors.





Observations: This heatmap does not indicate any problem with correlations.
The main problem in the dataset is zeros.

Preprocessing

One can realize from our exploratory analysis that our target variable can be best interpreted in log scale. Hence, we transformed our target variable into log scale. Also, having a zero-intensive dataset is very problematic when considering the range of each column. We have an average range 108502230.84 which creates difficulty in running our algorithms. In order to deal with this issue, we scaled our data. One of our trials was log scale, normalizing with standard deviation and mean and the other one was scaling between 0 and 1. Consequently; [0,1] scaling is resulted best on this subject.

We utilized following formula: $$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Another problem in our data is the number of features. We used an autoencoder to reduce the size of columns. We run autoencoder for different sizes and we test each of the sizes using ridge regression with 0.5 regularization coefficient. The results are in the following exhibit.

According to these results we decided to decrease feature size to 512, since we acquired minimum error at this point before overfitting.

Deep Learning

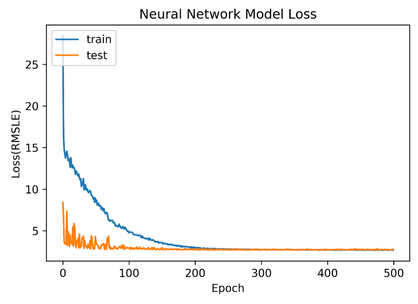
First predicted target values acquired using neural networks. We exploited various network architectures and we decided on two hidden layers in our network using ReLU as activation function on these two hidden layers. We have 400 nodes in the first layer and 41 nodes in the second layer. We initialized weights using Glorot and Bengio (2010) also known as Xavier initialization [2] which is formulated as:

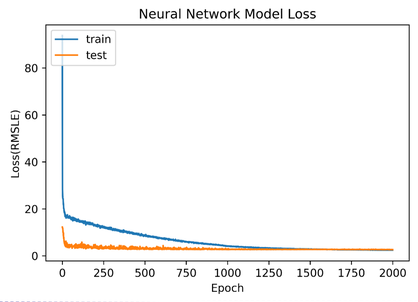
$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

We inserted ADAM (Adaptive Moment Estimation) algorithm for optimization [3]. Which is one of the most common used Stochastic Gradient Descent algorithms in Neural Networks.

To prevent overfitting and stabilize our training phase, we utilized dropout with probability 0.5 [5].

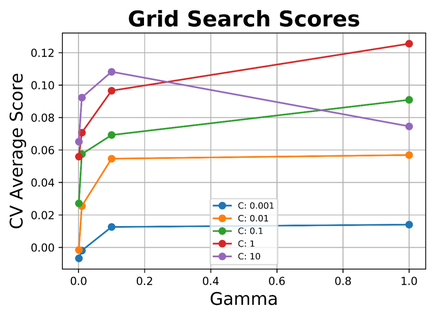
When we use default **learning rate 0.001**, we observed fluctuations of the loss of test set. Then, we tried a lower **learning rate 0.0001**. Even the setting with lower learning rate needs more epochs to converge, it reduces the fluctuations of the loss function.

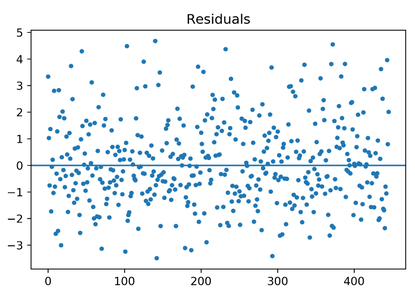




Support Vector Regression

Secondly, for prediction, we tried SVR with three different kernels: Linear, polynomial of various degrees and radial basis function. When we observed that RBF kernel outperforms others, we focused on tuning the model with RBF kernel. We conducted tuning processes on 5-fold cross-validation setting on a grid of hyperparameters.
We obtained following results:





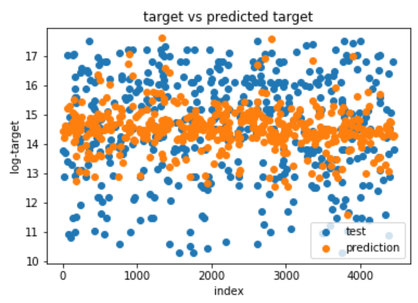
Gaussian Processes

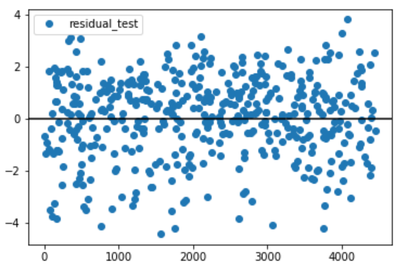
Finally, Gaussian Processes is conducted on the data as a promising method, since it can contain different kernels together (multiplying and adding kernels) and their hyperparameters can be specified [1,4].
Additionally, Python has a powerful tool: GaussianProcessRegressor together with gp.optimizer maximizing log-marginal-likelihood. Hence, this allows us to estimate and repeat the processes until we reach a better result.
The kernel called WhiteKernel deals with the noise in the data which is an important step during estimations.

RBF + WhiteKernel	• Log ML = -7731.3726 • Train Error = 1.6047 • Test Error = 1.6063
RBF + RationalQuadratic + WhiteKernel	• Log ML = -7702.9566 • Train Error = 0.1245 • TestError = 1.5671
WhiteKernel + RBF + RBF + RationalQuadratic	• Log ML = -7666.0698 • Train Error = 0.1357 • Test Error = 1.5525
RationalQuadratic + WhiteKernel	• Log ML = -7685.3246 • Train Error = 1.4316 • Test Error = 1.5817
RBF + DotProduct + RationalQuadratic + WhiteKernel	• Log ML = -7718.0526 • Train Error = 0.1652 • Test Error = 1.5995
RBF*DotProduct + RationalQuadratic + WhiteKernel	• Log ML = -8632.9294 • Train Error = 0.1279 • Test Error = 1.5644

BEST

After many trials and parameter tuning, the best kernel combination is selected to perform on the real test data.
The predicted target values plotted against validation target values together with residuals can be observed below.
The predicted values are accumulated around the mean of real target values (log scaled target values).





Conclusions

- ✓ According to size of this dataset
- ✓ Thanks to our dimension reduction
- ✓ Depending on the selected methods
- ✓ By repeating to tune the parameters of functions

BEST (min RMSLE) prediction of customer values are revealed by **Gaussian Process** with previously selected kernels. Real test error results retrieved from Kaggle challenge website:

Neural Networks
RMSLE = 1.7739

Support Vector Regression
RMSLE = 1.7352

Gaussian Process
RMSLE = 1.7260

Future Work

- ➢ Better and complex feature selection algorithms can be applied further in order to reduce the data size to obtain meaningful portion of columns, this resulted data may allow us to improve predictions while using the same methods.
- ➢ Since we obtained better results with GP, we can come up with neural networks with non-parametric activation functions corresponding to a x-layer deep Gaussian Process. At this point, the trade-off is the compilation time versus gain in terms of error minimization [1].

References

[1] Duvenaud, D. (2014). *Automatic model construction with Gaussian processes* (Doctoral dissertation, University of Cambridge).
[2] Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249-256).
[3] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
[4] Rasmussen, C. E., & Williams, C. K. (2006). *Gaussian process for machine learning*. MIT press.
[5] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929-1958.