Regression model for auto insurance claim probability predictions

Shenchao Zhang, redID: 821247357,
Xin Xue, redID: 822103861,
Xinqiao Zhang, redID: 822099792

***Abstract*—this paper describes our work to build a regression model for Porto Seguro's insurance company in Kaggle competition to predict the auto insurance claim probability next year. We propose a model composing lightGBM and neural network. First, we transform the 57 input features to 221 input features by removing the calculation features which are neural and adding one-hot encoding on categorical features. Then we use the 221 input features to train LightGBM model and fine tune the model by sweeping different hyperparameters. The best LightGBM model can reach 0.29098 standard Gini coefficient. Next, we use unsupervised learning method-denoising autoencoder to generate up-sample features, from 221 to 4500 features, then feed new features into neural network and get a score of 0.28934. After linear ensemble of these two models, our score is 0.29388, which can beat the 3rd place result in the competition.**

***Index Terms*— LightGBM, Autoencoder, Neural Network**

I. INTRODUCTION

For auto insurance company such as Porto Seguro, a precise insurance model from machine learning can help them predict the probability that a driver will initiate an auto insurance claim in the next year correctly. With that information, they can tail the price for every individual and provide fairer insurance cost on the basis of individual driving habits.

This project comes from a finished Kaggle competition for Porto Seguro - one of Brazil's largest auto and homeowner insurance companies. We continue the competition to get a better result.

II. DATA PREPARATION.

A. *Data description*

The data comes from Porto Seguro in the traditional Kaggle form of one training and test file each: train.csv & test.csv. In these two csv files, each row corresponds to a specific policy holder and the columns describe their features. The target column signifies whether a claim was filed for that policyholder.

Training data has 595212 rows (samples) and 59 columns, including sample ID, target (label) and other 57 features (inputs). Testing data has 892816 rows and 58 columns, including only sample ID and 57 features (inputs).

The features are anonymized, the input features that belong to similar groups are tagged as such in the feature name (e.g., ind, reg, car, calc). Here, ind means individual driver's information, reg means regional information, car means auto information and calc means calculation information. For the postfix in feature names, bin indicates binary features and cat indicates categorical features. Features without these designations are either continuous(floating) or ordinal(integer). Values of -1 indicate that the feature was missing from the observation.

B. *Data visualization*

After statistics, the number of columns corresponding to ID, target (label) and different kinds of input features are listed in the table.1.

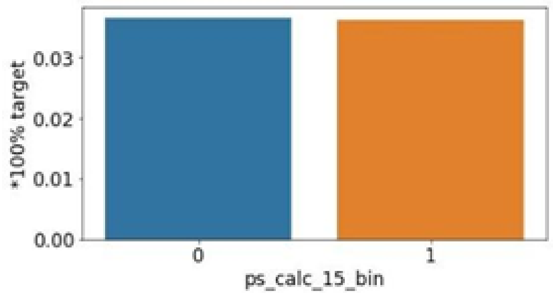
Table.1 statistics of 59 columns in training data

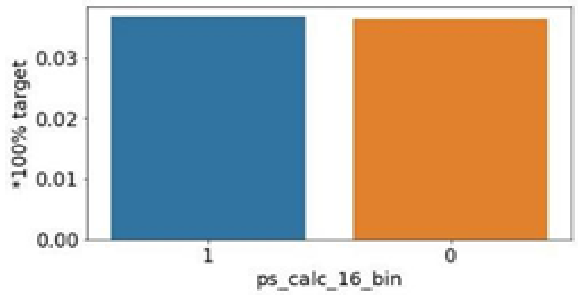
	role	level	count
ID	Id	Nominal	1
Binary feature	Input	Binary	17
continuous feature	Input	Interval	10
Categorical feature	Input	Nominal	14
Integer feature	Input	Ordinal	16
Target(label)	target	binary	1

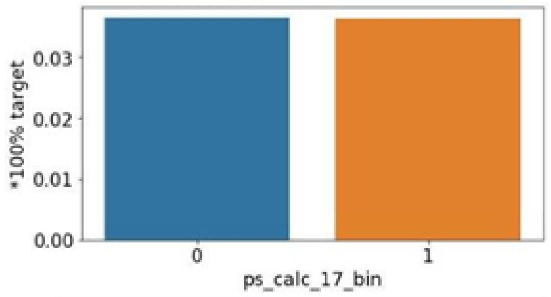
In order to observe influence of a input feature on label target, we need to visualize the claim rate(the percentage of target=1) distribution along different categories or different values for every feature. If the distribution is a constant line, this feature does not have influence on target. The more variation the distribution along the different categories or different values has, the more influence this feature has on label.

1) binary features

For the 17 binary features, there are 6 binary features belonging to calculation feature group and 11 binary features belonging to individual feature group. According to the reservation, we can find the claim rates for category 0 and category 1 are balanced for calculation feature group, which means these features are neural and almost does not affect the claim rate. For example, calc_15_bin, calc_16_bin, calc_17_bin and calc_18_bin plotted in Fig.1 have the balanced distributions.







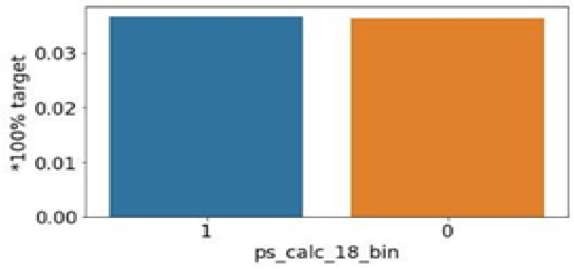
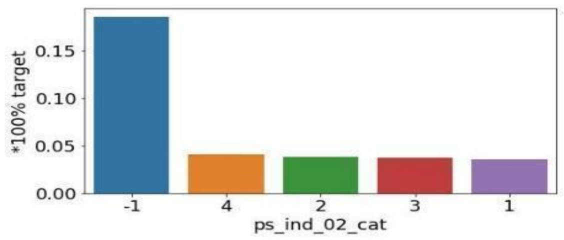
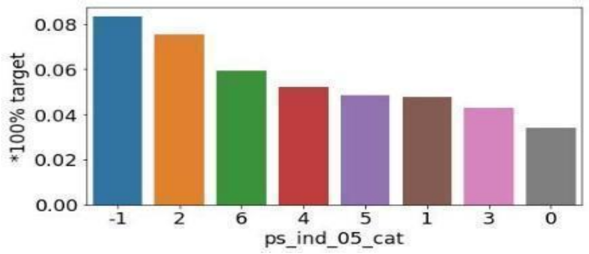


Fig. 1. Claim rate distribution for some binary calculation features

2) categorical features

For the 14 categorical features, there are 3 categorical features belonging to individual feature group and 11 categorical features belonging to car feature group. ind_02_cat, ind_05_cat, car_01_cat and car_06_cat are plotted in the Fig.2. From Fig.2, we can find that the missing data (denoted at -1) has higher claim rate in some features, such as in ps_ind_02_cat and ps_car_01_cat and so on.





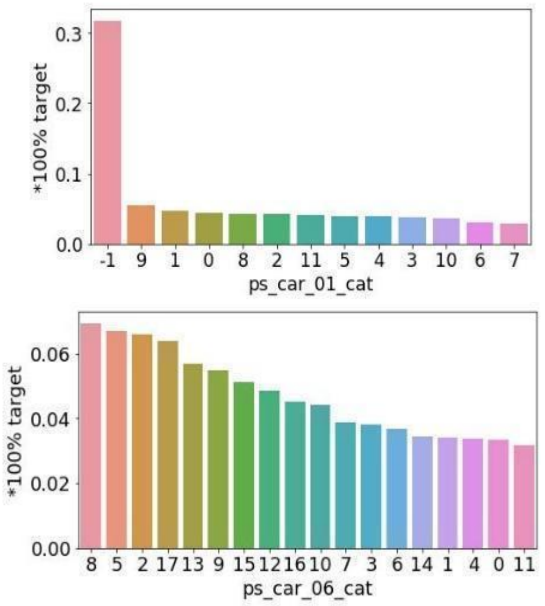
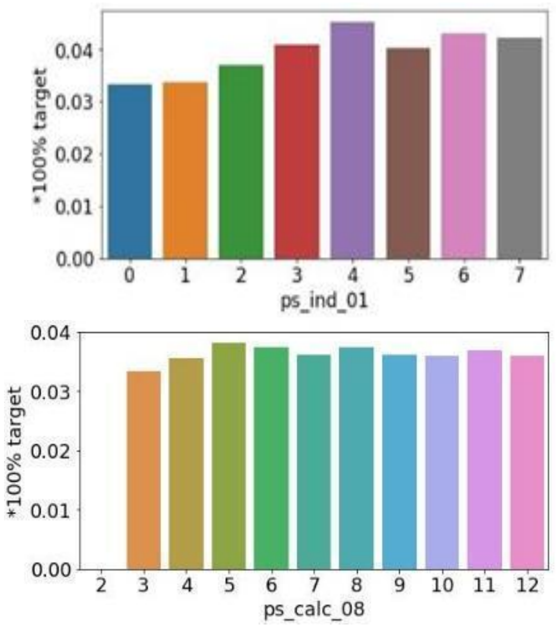


Fig. 2. Claim rate distribution for categorical features

3) integer features

For the 16 integer features, there are 4 integer features belonging to individual feature group, 1 integer feature belonging to car feature group and 11 integer features belonging to calculation feature group. The claim rate distribution for ind_01, calc_08, calc_09 and calc_11 are plotted in the Fig.3. We can find the claim rate distributions have little variation for these calculation features, which means the calculation features almost does not affect the claim rate.



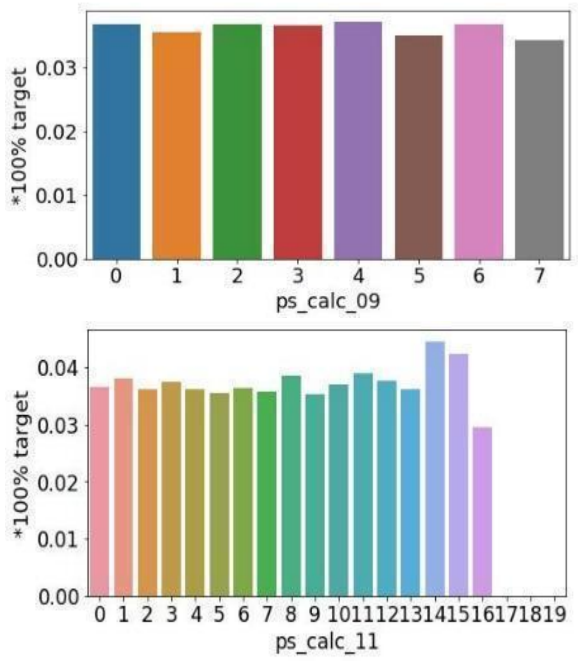
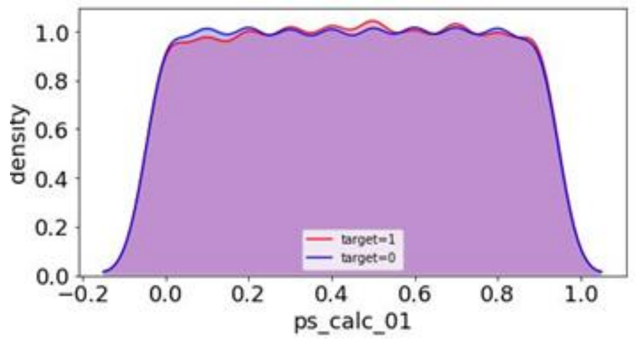
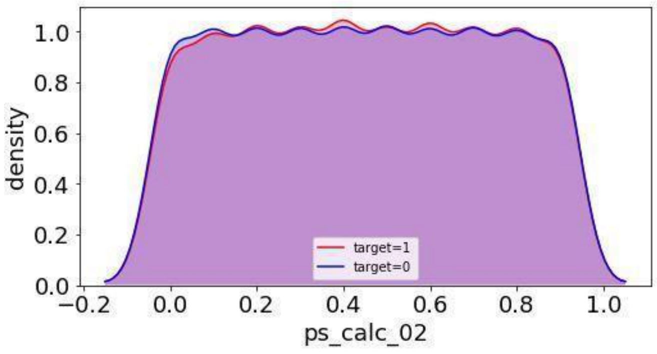


Fig. 3. Claim rate distribution for integer features

4) continuous features

For 10 continuous features, there are 3 continuous features belonging to regional feature group, 4 continuous features belonging to car feature group and 3 continuous features belonging to calculation feature group. For continuous features, the Probability density function(PDF) is used to specify input feature distribution. The PDF is the probability of the random variable falling within particular range of values, as opposed to taking on any one value. The density distributions for ps_calc_01, ps_calc_02 and ps_calc_03 are plotted in the Fig.4. Continuous calculation features show almost perfect overlap, which means they almost do not have influence on target.





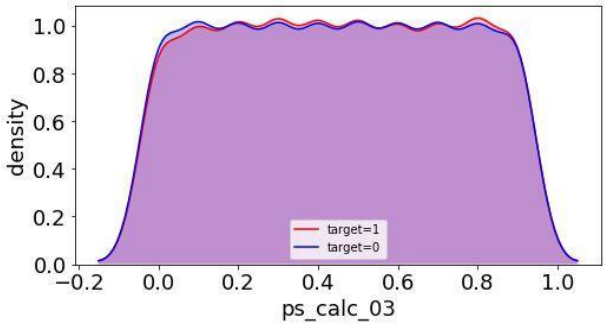


Fig. 4. Claim rate density distribution for *calc continuous features

C. *Feature engineering*

1) *Feature selection*

From the above result, we find the calculation features are neutral and almost do not have influence on target, so we remove all the calculation features.

To prove that the feature selection helps to improve model accuracy, we apply one-hot encoding on raw data and data after feature selection respectively, then feed them into lightGBM model with same hyperparameters. We find the model fed with preprocessed data have better standard Gini coefficient (0.29047) than the model fed with raw data (0.25778).

2) *one-hot encoding for categorical features*

Categorical variables represented in integer numeric do not have ordinal relationship in fact. Direct usage of integer numeric allows the model to assume a natural ordering between categories, which may result in poor performance or unexpected results. So, we apply one-hot encoding on 14 categorical features to remove the ordering relationship. After feature selection and one-hot encoding, we get 221 features.

3) *Rand Gaussian normalization*

Input normalization for gradient-based models such as neural nets is critical. For lightgbm/xgb it does not matter. The best what we found during the past and works straight of the box is "RankGauss". It’s based on rank transformation. First step is to assign a linspace to the sorted features from 0 to 1, then apply the inverse of error function ErfInv to shape them like gaussians, then we subtract the mean. Binary features are not touched with this approach (eg. 1-hot ones). This works usually much better than standard mean/std scaler or min/max.
For example:

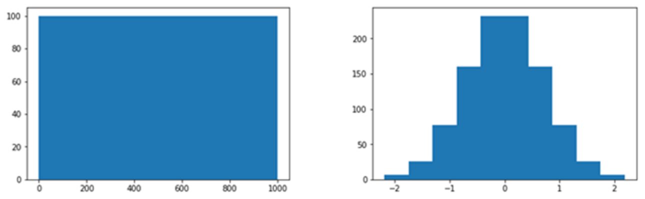


Fig. 5. original data histogram vs normalized data histogram

III. Models and Result

A. *lightGBM*

Light GBM is a gradient boosting framework that uses tree-based learning algorithm. Light GBM grows tree vertically while other boosting algorithm grows trees horizontally, that means Light GBM grows tree leaf-wise while other algorithms grows tree level-wise. It will choose the leaf with max delta loss to grow. When growing the same leaf, Leaf-wise algorithm can reduce more loss than a level-wise algorithm. The working principle of light GBM and other boosting algorithms are illustrated in Fig.6.

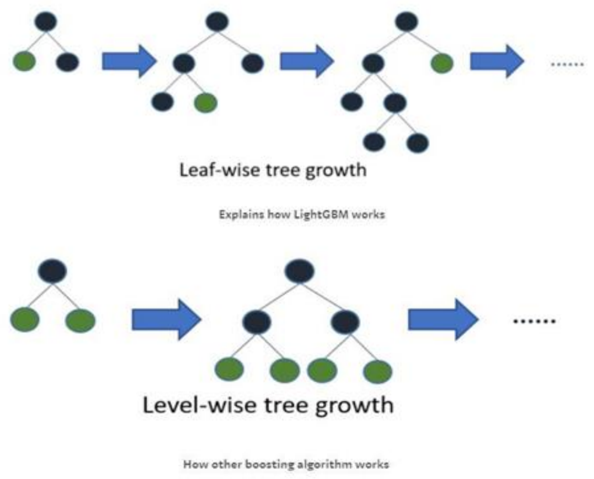

Leaf-wise tree growth
Explains how LightGBM works
Level-wise tree growth
How other boosting algorithm works

Fig.6. working principle of lightGBM vs another boosting algorithm

The advantage of light GBM includes its high speed, low memory requirement; the disadvantage of light GBM is its sensitivity to overfitting, especially for small data set.

After feature selection (removing calculation features) and one-hot encoding for categorical features, we feed the prepared data into lightGBM model and fine tune the hyperparameters, including learning rate, num_leaves, feature_fraction and bagging_fraction.we sweep one of those parameters while others' fixed. The result(kaggle score) comparison of different learning rate is listed in table.2. The result(Kaggle score) comparison of different num_leaves is listed in table.3. The result (Kaggle score) comparison of different feature_fraction is listed in Table.4. The result (Kaggle score) comparison of different bagging_fraction is listed in Table.5.

Table.2 score comparison of different learning rate

Learning rate: *(This determines the impact of each tree on the final outcome. GBM works by starting with an initial estimate which is updated using the output of each tree. The learning parameter controls the magnitude of this change in the estimates. Typical values: 0.1, 0.001, 0.003…)*	value	Kaggle score
	0.001	0.25412
	0.005	0.28514
	0.01	**0.29047**
	0.05	0.26893
	0.08	0.25017
	0.10	0.23982
	0.3	0.17491
	0.5	0.12964

Table.3 score comparison of different num_leaves

num_leaves *(number of leaves in full tree, default: 31)*	value	kaggle score
	10	0.28690
	20	0.29029
	30	0.29038
	31	**0.29047**
	40	0.29064
	50	0.28959

Table.4 score comparison of different feature_fraction

feature_fraction *(Used when your boosting(discussed later) is random forest. 0.8 feature fraction means LightGBM will select 80% of parameters randomly in each iteration for building trees.)*	value	Kaggle score
	0.5	**0.29098**
	0.6	0.29087
	0.7	0.29047
	0.8	0.29079

Table.5 score comparison of different bagging_fraction

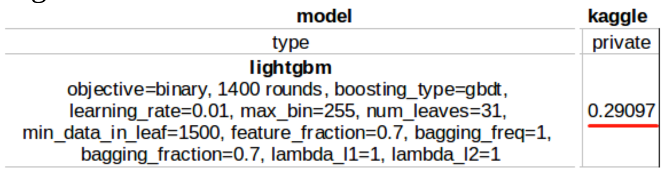
bagging_fraction *(specifies the fraction of data to be used for each iteration and is generally used to speed up the training and avoid overfitting.)*	value	kaggle score
	0.5	0.29041
	0.6	0.29038
	0.7	0.29047
	0.8	0.29047
	0.9	**0.29087**

After fine tune of hyperparameters, we get the best lightGBM model with following parameters. Its score can reach 0.29098, which is even higher than the 1st player in Kaggle competition.

'boosting_type': 'gbdt',
'learning_rate': 0.01,
'num_leaves': 31,
'min_data_in_leaf': 1500,
'feature_fraction': 0.5,
'bagging_freq' :1,

'bagging_fraction': 0.9,
'lambda_l1':1,
'lambda_l2':1

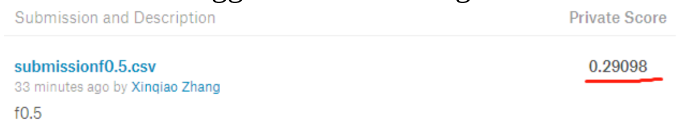
First player’s result from lightGBM is shown in Fig.7.



model	kaggle
type	private
lightgbm objective=binary, 1400 rounds, boosting_type=gbdt, learning_rate=0.01, max_bin=255, num_leaves=31, min_data_in_leaf=1500, feature_fraction=0.7, bagging_freq=1, bagging_fraction=0.7, lambda_l1=1, lambda_l2=1	0.29097

Fig.7. kaggle score from first player’s lightGBM

Our result in Kaggle is shown in Fig.8.



Submission and Description	Private Score
submissionf0.5.csv 33 minutes ago by Xinqiao Zhang f0.5	0.29098

Fig.8. kaggle score from our lightGBM

B. Denoising autoencoder+ Neural Network

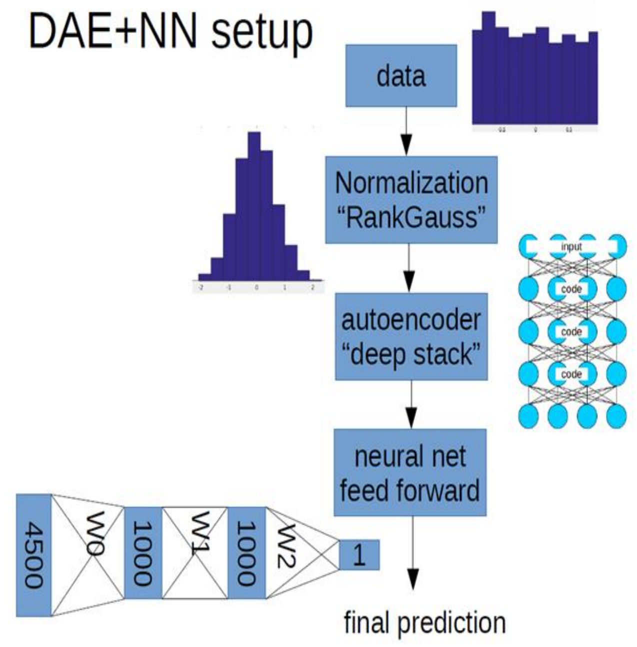


Fig.8. DAE and NN Setup

Denoising Autoencoder

Denoising autoencoders (DAE) are nice to find a better representation of the numeric data for later neural net supervised learning. One can use train+test features to build the DAE. The larger the test set, the better.

An autoencoder tries to reconstruct the inputs features. So, features just are targets. Output layer activation function is linear. A denoising autoencoder tries to reconstruct the noisy version of the features. It tries to find some representation of the data to better reconstruct the clean one.

The biggest problem in up-sampling autoencoder is : learning the identity, that will not help to representation the data. The critical part here is to invent the noise. In tabular datasets we cannot just flip, rotate, sheer like people are doing this in images. Adding gaussian or uniform additive / multiplicative noise is not optimal since features have different scale or a discrete set of values that some noise just did not make sense.

We used a noise schema called "swap noise". Here we sample from the feature itself with a certain probability "inputSwapNoise" . 0.15 means 15% of features replaced by values from another row.

Deep stack, where the new features are the values of the activations on all hidden layers. This DAE step usually blows the input dimensionality to 1k..10k range.

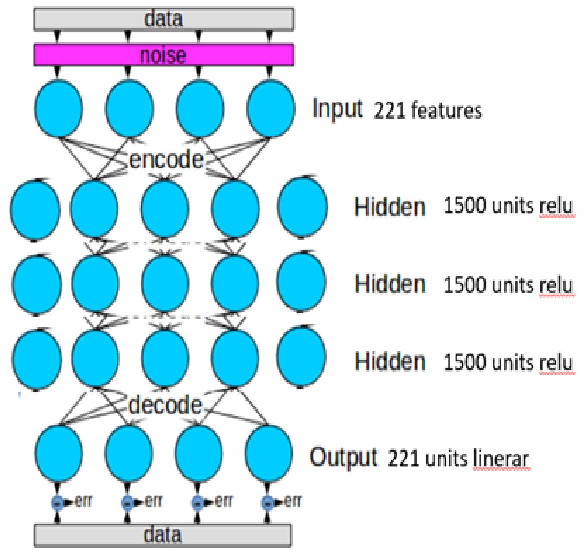


Fig.9. DAE Working Flow

Neural network

The neural network figure is shown as follows:

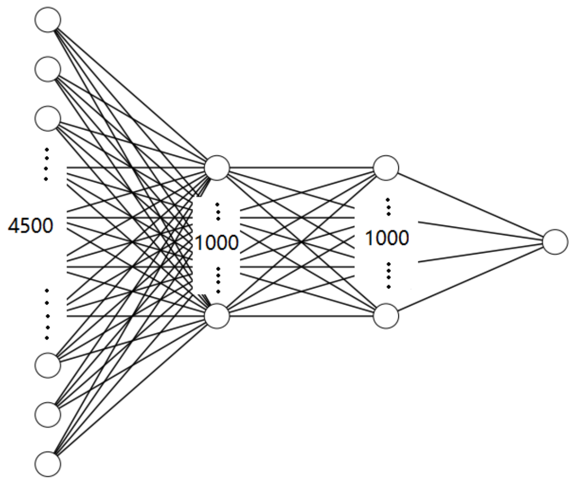


Fig.10. Neural Network Topology

At first we put "train_data.csv(113MB)" and "test_data (164MB)" into DAE. After 20 hours we get new data files which are "new_train_data(14.3GB)" and "new_test_data:(23.2GB)".

Next, we use the new data and put them in neural network to train a model. It took 15 hours to get result.

.

Parameter	***Score***
Topology = 4500-1000r-1000r-1s, Dropout =0.75 -0.5, Epochs=20, Batch_size=512, Verbose=2	*0.28934*

C. Ensemble:

We try nonlinear and linear ensemble with one lightGBM and one neural network, but it failed, the final score doesn't improve.

We built a neural network with 5 hidden layers and another neural network with one hidden layer to do nonlinear and linear ensemble. However, the score doesn't improve. It seems like there is a lot of noise and very easy to overfit the training data and validation data.

So, finally, we only stick with two models with the same weight, and just do the average ensemble, the score improves.

Ensemble	**Score**
Single lightgbm	0.29098
Single neural network	0.28934
Nonlinear	0.28527
Linear	0.28861
Average	0.29388

Our final score for this finished competition is 0.29388, which can beat the 3rd place result.

simple_average_all.csv 0.29388
3 days ago by Roger Zhang
add submission details

IV. CONCLUSION

In this project, we get a decent result by fine tuning the lightGBM model after adding one-hot encoding for categorical features. Besides that, we normalize the 221 inputs features by rand gaussian normalization, and then use DAE to up-sample the input with much more dimensions than before. After DAE, we feed the up-sampling data into neural network. What is more, our feature engineering like feature selection is wise and it helps get higher accuracy than before.

V. References

[1] A. Mesaros, T. Heittola, and T. Virtanen, “TUT database for acoustic scene classification and ound event detection,”in 24th European Signal Processing Conference (EUSIPCO2016). Budapest, Hungary: IEEE, Aug 2016, pp. 1128–1132.

[2] Michael Jahrer, https://www.kaggle.com/c/porto-seguro-safe-driver-prediction/discussion/44629

[3] Bishop, C. M. (1995). Training with noise is equivalent to tikhonov regularization. Neural Computation, 7, 108– 116.

[4] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th International Conference on Machine Learning, pages 1096–1103. ACM, 2008.

[5] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. In Proceedings of the 27th International Conference on Machine Learning, pages 3371–3408. ACM, 2010.