COMP0081: Otto Group Product Classification Challenge

Wenguang Lin, Zhenglin Liu, Zhijia Chen, Zefeng Liao, Shiwen Liu

I. INTRODUCTION

The aim of this project is to analyze the Otto product classification challenge on Kaggle and try to propose a solution that yields good results. The solution is based on a three-layer stacking architecture. In the first layer, there are about 36 models 8 engineered features. All models in the first layer are trained by using a five fold cross-validation technique using always the same fold indices. In the second level, there are four models trained using 36 meta features and 8 features from 1st level: XGBoost, Neural Network(NN), CatBoost, and LGBM. In the third layer, it's composed by a arithmetic mean of second level predictions.

II. THE DATA SET

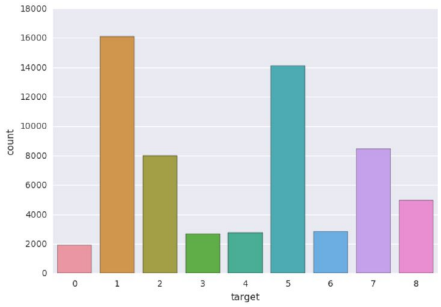

Fig. 1. The partition of classes in the data set

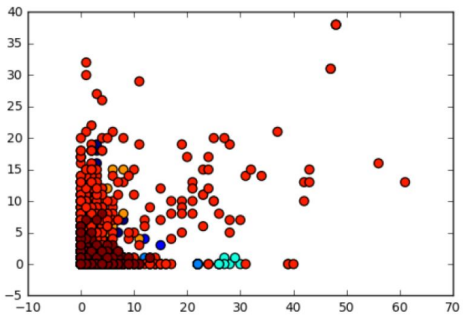

Fig. 2. data points

The OTTO Group Challenge is a competition which has as target to build a classifier, able to distinguish, for each product, a specific category to which it belongs. The data set provided by the organizers contains more than 200,000 products, where each product is described by 93 features. As mentioned before, the objective was to classify these products in one of the 9 categories. For this purpose, 30% of the data set was used for training and validation, while the remaining 70% was used to test the classifier. Figure 1 shows the partition of 9 classes in the data having a class imbalanced problem. As it is shown in the figure 2, the classes have very similar characteristics. Most of data points are accumulating at the left bottom part of the graph so it is also a class overlapping problem.

III. CLASSIFICATION ALGORITHMS

The purpose of this section is to propose a methodology (stacking) to yield good results for in this challenge. The stacking mainly comprises three layers:

- In the first level, there are about 36 models that we used their predictions as meta features for the second level, also there are 8 engineered features. All models in the first levels are trained by using a five fold cross-validation technique using always the same fold indices
- In the second level, there are four models trained using 36 meta features and 8 features from 1st level: XGBoost, Neural Network(NN), CatBoost, and LGBM.
- In the third level, it's composed by a weighted mean of 2nd level predictions.

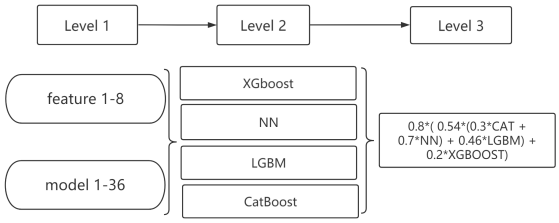

Fig. 3. Concept Diagram of Stacking

The purpose of stacking is to combine the output of 'weak' classifiers to generate a stronger classifier. The concept of staking is shown in figure 3. The section will also talk about the concept of stacking and explain what kind of pre-processing, modelling and validation we have been used for each layer.

A. First Layer

In order to avoid the over fitting, we cannot use the model trained by the whole training features to predict the training labels. Therefore, We decided to use K-fold cross validation method in this case.

As you can see in the figure 4, the basic training set is divided into five folds for 5-folds cross-validation. This means that four folds are taken as training data and one fold is taken as testing data. Take model one as an example and

assume that training set comprises 10,000 rows of data. Thus in every step of the cross-validation, the training data will comprise 8,000 row and validation data will comprise 2,000 rows.

There are two processes in every cross-validation:

- training a model based on the training data, predicting the testing data based on the model generated by the training data
- storing the prediction from each fold.

After the entire five fold, we will get the $9\times10{,}000$ predicted value of the entire training data set from model one. Then we do the same process for the other 35 models. So that we will have $9\times36 = 324$ columns, 10,000 rows meta features.

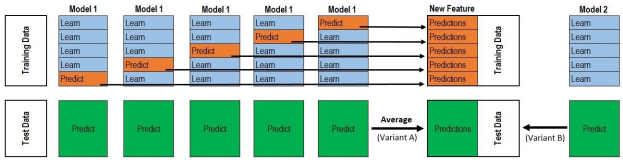

Fig. 4. Stacking

The 36 models we used in the first layers are elaborated below:

- **model1** : Random forest is a tool which takes a subset of observations and a subset of variables to build decision trees[1]. It builds multiple such decision trees and merges them together to get a more accurate and stable prediction. There are two main parameters which we modified:
  1) $\mathbf{n-estimators}$: represent is the number of the build trees. Higher number of trees a better performance but it make the code slower too. This parameter is chosen depending to the machine capacities.
  2) $\mathbf{max-features}$: represents the maximum number of features Random forest is allowed to try in an individual tree. Increasing $max-features$ generally improves the performance of the model, because more options can be taken into account.

  In order to decide the number of features, Cross-validation is used for many parameters.

- **model2** : Logistic regression is as statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extension exist, In regression analysis, logistic regression is estimating the parameters of a logistic model (a form of binary regression). For the data pre-processing part, we transform the data set to Log(X+1).

- **model3** : Extra Trees Classifier is a class implementing a meta estimator that fits a number of randomized decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. For the data pre-processing, we modified the raw data with Log(X+1).

- **model4** : KNN is used for the model 4. For the data pre-processing, we modified the raw data with Log(X+1).

- **model5** : Factorization machines (FMs) are a type of supervised learning algorithm. They are used for classification and regression tasks. They accomplish this by measuring interactions between variables within large data sets. FM's are extensions of linear models which model the interactions of variables. The difference between FMs and FFMs is that in FFMs, each feature has different latent vectors to learn the latent effect with features from different fields. Instead of using the same latent vector to measure all interactions as FMs, it will use different parameters for dot product in estimating those interactions. This model is implemented using xlearn library.

  In terms of the features, there are two main adaptations we made for this competition:

  – As all features are numerical, we converted them into libffm format by adding a dummy field for each of them. For example, feature 6 with value of 12 can be transformed to 6:6:12.
  – Since given features are sparse where lots of features with value of 0, we only transformed those features whose values are non-zero to libffm format and feed them into the FFMs model. By doing this, we significantly improved the efficiency of training and the performance of our model.

- **model6** : lightGBM is a kind of assemble algorithm but much faster than XGBoost algorithm. The gradient-based one-side sampling technology reduces the number of samples. Exclusive feature bundling reduces the number of features. For the data pre-processing part, we transform the data set to Log(X+1). Then, we used standard scaler to make processed data conform to a standard normal distribution.

- **model7** : Multinomial Naive Bayes is a supervised learning algorithm used in the classification problem by calculating the probability of each category. For the data pre-processing part, we transform the data set to Log(X+1).

- **model8** : Nerual Network is used as the model 8. We trained two types of NN for this case. The first NN is a 2 layers KerasClassifier which has 0.05 for dropout, 4096 for dense in the first layer and 0.05 for dropout, 526 for dense in the second layer. The second NN is a 2 layer MLPClassifer whose solver=adam, alpha=1e-5 and hidden layer size=(20,15). The result can be obtained by taking the average of two NN outputs. For the data pre-processing, we modified the raw data set with Scale(Log(X+1)) and Scale(X) for two types of

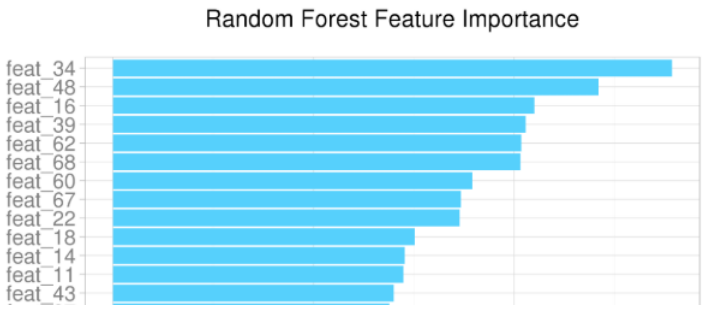

Fig. 5. 13 features Random Forest Importance

NN respectively. Scale here is to standardize the data from -1 to 1.

- **model9** : Nerual Network is used as the model 9. We trained MLPClassifier NN with 2 layers for 10 times in this model. There are 20 and 15 nodes for the 2 layers respectively. The average value of 10 outputs of MLPClassifiers is chosen as the result.

- **model10** : T-sne is dimension reduction methods to reduce the raw data into 3 dimensions. For the data pre-processing, we added two extra kmeans features into the raw data by using the T-sne 3 dimension. Then we also modified the new data set with Log(X+1).

- **model11 – 13** : Sofia-ml (suite of fast incremental algorithms for machine learning) can be used for training models for classification, regression, ranking, or combined regression and ranking. It uses some speedy fitting algorithms known as "Pegasos" (short for "primal estimated sub-gradient solver for SVM"). "Pegasos" has an advantage in that you do not need to define pesky parameters such as learning rate. Another nice feature in sofia-ml is that it supposedly also can optimize ROC area via selecting smart choices of samples when iterating over the data set. There are two types of smart choices used in our model which are balanced-stochastic and combined-roc respectively. The learner type of Sofia used for our case is logistic regression (with Pegasos Projection).
  Figure 5 shows the 13 most important features based in random Forest importance. These features are utilized to construct some 3 level feature interactions. Each interaction is simply a multiplication of a combination of 3 features chosen among those 13 most important features. We tested the performance of stacking one interaction each time for all different combinations against using raw features as baseline by 5-fold cross validation. Then top 4 interactions resulting in the smallest loss are chosen.
  - In model 11, we trained using one against all with learner type="logreg-pegasos" and loop type= "balanced-stochastic". For the data pre-processing, we standardize the data using scale(X).
  - In model 12, we trained using one against all with learner type="logreg-pegasos" and loop type= "balanced-stochastic". For the data pre-processing, we stacked several extra features to the raw features, including 1. T-sne with 3 dimensions of the original features, 2. some 3 level interactions between 13 most important features based in randomForest importance. Then we standardize the new features using Scale(X_new).
  - In model 13, we trained one against all with learner type="logreg-pegasos" and loop type= "combined-roc". For the data pre-processing, we stacked several extra features to the raw features, including 1. T-sne with 3 dimensions of the original features, 2. some 3 level interactions between 13 most important features based in random Forest importance. Then we add 1 to the raw features and interactions by 1 and take the logarithm of them.

- **model14 – 18** : XGBoost (Extreme Gradient Boosting) algorithm is a supervised algorithm [1], working based on the ensemble trees, where the decision depends on the ensemble trees, where the decision depends on summation of the scores of multiple optimized trees. The leaves values aren't the target but they are real values (score). The objective function consists of two functions (training loss and regularization). We used Xgboost with different pre-processings for the model 14, 15, 16, 17, 18.
  - In the model 14, We trained one against all. For the data set, we added another column of feature which is the summation of zeros in original features. All zeros are replaced by NA.
  - In the model 15, We trained Multiple class Soft-Prob. For pre-processing part, We added several extra features to the data set, including 1. 7 different K-means with different number of clusters (6, 7, 8, 9 ,10, 11, 12 where 9 is determined to be the optimal k value using both the Elbow method and the Silhouette method), 2. the row summation of X==0, 3. the row summation (scale(x) $>$ 0.5) and 4. the row summation (Scale(x)$<$-0.5). The scale here is to standardize the data from -1 to 1.
  - In the model 16, We trained Multiple class Soft-Prob. For pre-processing part, We added several extra features to the data set, including 1. T-sne features 2. a K-means with 9 clusters of the data set X.
  - In the model 17, We trained Multiple class Soft-Prob. For pre-processing part, We added several extra features to the data set, including 1. T-sne features 2. a K-means with 9 clusters of log(1+X). The Log transformation here is to make the data set more comparable.
  - In the model 18, We trained Multiple class Soft-Prob. For pre-processing part, We added several

extra features to the data set, including 1. T-sne features 2. a K-means with 9 clusters of Scale(X). The scale here is to standardize the data from -1 to 1.

- **model19 – 20** : Neural Network is used as the model 19 and 20. A 2 layers NN is trained for 120 times with different number of epochs for the model 19. A 3-layers NN is trained for 120 times with different epochs for the model 20. The average values of 120 outputs are taken as the result for both models.

- **model21** : Xgboost is used for the model 21. The raw data set is used to train the model with Multiple Class Soft-Prob for 30 times. Before training every model, the training set will be shuffled completely. After the 30 times training, the average value will be taken as the result.

- **model22 – 33**, KNN (K Nearest Neighbourhood) is used for the model 22-33 but with different neighbours. It is a non-parametric method used for classification and regression. Both for classification and regression, a useful technique can be to assign weights to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor.
  - In the model 22, all features where has zero values are changed to 1. This can be done by feature X = int(X==0)
  - In the model 23, all features where has zero values are changed to 1.Then Log(X+1) is added on the all features. This can be done by feature X + int(X==0) + log(X+1).
  - In the model 24, 2 neighbours are chosen for the KNN with the raw data set.
  - In the model 25, 4 neighbours are chosen for the KNN with the raw data set.
  - In the model 26, 8 neighbours are chosen for the KNN with the raw data set.
  - In the model 27, 16 neighbours are chosen for the KNN with the raw data set.
  - In the model 28, 32 neighbours are chosen for the KNN with the raw data set.
  - In the model 29, 64 neighbours are chosen for the KNN with the raw data set.
  - In the model 30, 128 neighbours are chosen for the KNN with the raw data set.
  - In the model 31, 256 neighbours are chosen for the KNN with the raw data set.
  - In the model 32, 512 neighbours are chosen for the KNN with the raw data set.
  - In the model 33, 1024 neighbours are chosen for the KNN with the raw data set.
- **model34 – 35** CatBoost is used for the model 34 and 35 with different pre-processing methods. The purpose of CatBoost is to train and apply models for the classification problems. It can provide compatibility with the scikit-learn tools. The default optimized objective depends on various conditions:
  - Logloss–The targert has only two differnt values or the target border parameter is not None.
  - MultiClass–The target has more than two different values and the border count parameter is None.
  - In the model 34, we loaded the dataset using Pool. Pool is a fastest way to pass the feature data to the Pool constructor, thus CatBOost Classifier method can accept it. We trained the catboost classifier with the raw data set.
  - In the model 35, for the data pre-processing, we added several extra features to the data set, including 7 different K-means with different number of clusters (6, 7, 8, 9, 10, 11, 12), the row summation (Scale(x)¿0.5) and the row summation (Scale(x)¡-0.5). The scale here is to standardize the data from -1 to 1.

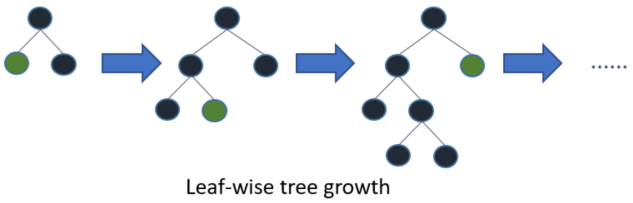

Fig. 6. Leaf-wise tree growth

- **model36** : LGBM [2] is a relatively new algorithm recently, and is used for the model 36. LightGBM is a gradient boosting framework that wuses tree based learning algorithm. It grows tree leaf-wise/vertically. 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 which other algorithm grows. The figure 6 shows how the LightGBM grows with Leaf-wise tree growth.

There are 8 features used for training in the first layer.

- Feature 1: Distances to nearest neighbours of each classes. There are nine columns of the outputs,
- Feature 2: Sum of distances of 4 nearest neighbours of each classes.
- Feature 3: Sum of distances of 4 nearest neighbours of each classes.
- Feature 4: Distances to nearest neighbours of each classes in TFIDF space.
- Feature 5: Distances to nearest neighbours of each classes in T-SNE space. In the T-SNE, we stacked the features of training set and the features of testing set together first and then reduced the dimensions of the

whole new feature set to 3, otherwise the training feature X after the T-SNE dimension reduction might be very different from the testing feature set.

- Feature 6: Clustering features of original data set.
- Feature 7: Number of non-zeros elements in each row.
- Feature 8: The raw input data X.

B. Second Layer

In the 2nd layer, there are 4 models trained using meta features and 8 features that we have obatined in the first layer: XGBOOST, Neural Network, CATBOOST, and LGBM. These models are fine tuned using five fold cross validation. After we found some good hyperparameters, we trained those models in the second level by using the entire training set. The loss for the second layer models are 0.40314(Catboost), 0.39904(NN), 0.39891(LGBM), and 0.39814(Xgboost) respectively.

C. Third Layer

In the third layer, we used an arithmetic mean of CatBoost, NN, LGBM, and XGBoost: $0.8 \times (0.54 \times (0.3 \times CatBoost + 0.7 \times NN) + 0.46 \times LGBM) + 0.2 \times XGBoost = 0.3955$. The final loss is 0.3955.

IV. Result and Discussion

We tried using AdaBoost with Extra tree or Decision tree as the base estimator, but it did not make any contribution to the final loss and made it even worse. So we decided not to use AdaBoost.

We also tried different pre-processing, such as doing T-sne reduction, log(X+1), sqrt(X + 3/8), scaling, kmean etc. We applied models on different type of pre-processing, so that we could reuse models, generated different outputs, and combined these outputs to obtain a better result.

We did not discard models whose results are not good in the first layer, since we found that they also improve the performance in the second layer.

The table I and II show all logloss results of different models in the first two levels respectively. The final logloss in the third level is 0.39736 as you can see in the figure7.

Fig. 7. logloss value in the third level

V. CONCLUSIONS

In this Otto product classification challenge, We used 36 models and 8 features to do the prediction of the products. We first fine tuned the models and tried generated some features that are helpful in the first layer. Then we fine tuned the models in the second layer. We tried different models that can do the bagging, namely XGBoost, Catboost, Neural Network, AdaBoost, and LGBM. Finally we decided to abandon AdaBoost. In the third layer, we took the arithmetic mean of the model in the second layer. The final loss is small than any second layer models, went down to 0.3955 eventually. The finally result might be improved by adding more models in the first layer and do more runs in the second layer.

TABLE I
Layer 1 accuracy

algorithm	accuracy
Model 1	0.53324
Model 2	0.67297
Model 3	0.69193
Model 4	0.68454
Model 5	0.51690
Model 6	0.51064
Model 7	1.22440
Model 8	0.51325
Model 9	0.50733
Model 11	1.02501
Model 12	1.01110
Model 13	0.83862
Model 14	0.46249
Model 15	0.43803
Model 16	0.43616
Model 17	0.43582
Model 18	0.43823
Model 19	0.50587
Model 20	0.53421
Model 21	0.43911
Model 22	0.78578
Model 23	0.69587
Model 24	3.13295
Model 25	1.94711
Model 26	1.27646
Model 27	0.93057
Model 28	0.75690
Model 29	0.69391
Model 30	0.69623
Model 31	0.71859
Model 32	0.76826
Model 33	0.84823
Model 34	0.45266
Model 35	0.45564
Model 36	0.43855

TABLE II
Layer 2 accuracy

Neural Network	0.39904
LGBM	0.39891
Catboost	0.40314
XGboost	0.40001

REFERENCES

[1] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[2] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in neural information processing systems*, 2017, pp. 3146–3154.