

COMS 4771 Kaggle Competition Report from Deus Ex Machina Team

BachViet Do

Computer Science, Columbia University

Kaggle Id: machinescholar

Email: bd2444@columbia.edu

Keyu Lai

Computer Science, Columbia University

Kaggle Id : keyulai

Email: kl2844@columbia.edu

Qianyuan Chen

Computer Science, Columbia University

Kaggle Id: RoyChen

Email: qc2200@columbia.edu

Abstract—The main purpose of this report is to present our team’s Kaggle result, document the competition process and explain our approach to achieve the accuracy score of 0.95667 which placed us in the 2nd position in the Public Leader Board.

We will briefly discuss about the feature design in the first section, our analysis and experiments of classifiers we employed in the competition in 2nd section, our final “recipe” of algorithm in the 3rd section, metric performance in the 4th section and end result of quiz set in the last section.

I. DATA PROCESSING AND FEATURE DESIGN

In our experiment, we have tried two feature presentation methods: label encoding and one-hot label encoding.

A. Label Encoding

Label encoding assigns an integer to every categorical feature. It’s the most simply form of feature representation. However, it assumes that all categories are ordered, which is not always true. For example, in distance-based algorithm such as SVM, it assumes that one feature is farther away from another, which is not always desirable.

B. One-hot Encoding

One-hot encoding is an efficient way to convert represent categorical features. It transforms each categorical feature with m possible values into m binary features, with only the active one as 1 and left the other as 0.

$$\Phi(x) = [0 \quad \dots \quad 0 \quad 1 \quad 0 \quad \dots \quad 0]^T \quad (1)$$

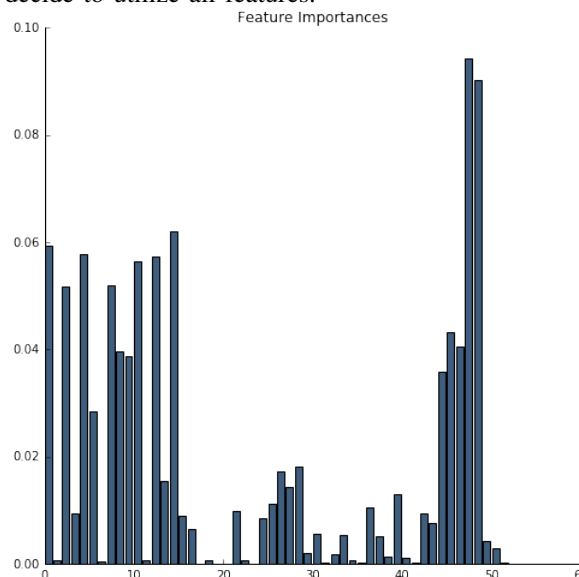
This will solve the order problem of label encoding described above.

C. Performance

We find that one-hot encoding and label encoder are comparable in term of accuracy score when using together with various standard classifiers such as decision tree, k-NN and SVM. However, we notice much faster running time with label encoder. It makes sense since label encoder maintains only 52 features while one-hot encoder creates more than 1500+ features!

On the other hand, not every the feature is equally important. The illustration in the below is extracted from one extra

tree classifier internal property. As you observe, features 27-55 are virtually nonessential in this classifier decision. Theoretically, we can drop them to gain better execution speed. Nevertheless, we notice no runtime improvement after removing these extra features while keeping them permits us to sustain about 0.1% more accuracy that is more than enough to differentiate ranking in the competition, so we decide to utilize all features.



II. MODEL DESCRIPTION

We began our investigation by testing our favorite classifiers, SVM and SGD. However, they did not perform very well in this dataset (80% vs 76%) and the class benchmark is 90%. On the second look at the dataset description and our feature design, we found out why. Firstly, we used label encoder to encode categorical variables, and so we have effectively removed the concept of distance from the most important features of the dataset. Secondly, most of features dealt with some sorts of syntatic structures which are rule-based by nature. As such, we suspected that non-linear classifier like decision tree or ensemble of trees might perform best.

In the proceeding, we will discuss in length about multiple

classifiers we used to achieve great performance. As expected, the ensemble of trees accomplished superior performance(> 94%). In order to boost our score further, we took the concept of ensembling to the extreme by employing a variation of ensemble blending algorithm popularized by Netflix Grand Prize competition.

A. K Nearest Neighbors

The KNN algorithm tries to directly approximate the decision boundaries of f . It computes the k , which is also one of the model parameter, training points that are closest to x based on Euclidean distance, then returns the plurality of the labels of these k points as the prediction result for this sample x . KNN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification.

B. Decision Tree

A decision tree is a flowchart-like structure in which each internal node represents a decision boundary on a feature, each branch represents the outcome of the test and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represents classification rules. During the learning process, the decision tree algorithm create a decision function $f: \mathcal{X} \rightarrow \mathcal{Y}$ represented by a binary tree, and for each node, it picks the rule h , which splits the node S into two children nodes S_L and S_R , that can obtain the maximally reduces in uncertainty.

$$h = \arg \max_h (u(S) - (p_L \cdot u(S_L) + p_R \cdot u(S_R))) \quad (2)$$

The uncertainty is the value to quantify the mismatch between the prediction result for a rule, and all samples within this rule (node). There are many different methods to compute the uncertainty, and in this competition the uncertainty we used are:

- 1) Classification error:

$$u(S) = 1 - \max_{k \in \mathcal{Y}} p_k^2 \quad (3)$$

- 2) Gini index:

$$u(S) = 1 - \sum_{k \in \mathcal{Y}} p_k^2 \quad (4)$$

C. Random Forest

Random Forest is a kind of boosting algorithm, which combines a lot of weak classifiers(decision tree) and vote/average these trees to get the predication result.

- 1) it grows each tree with N independently sampled points with replacement from the training data.
- 2) At each node:
 - a) It randomly select S features out of all possible features.
 - b) Find the rule that maximally reduces uncertainty on the selected m features.
- 3) It vote/average the predictions of all the trees to get the final prediction.

D. Extra tree

Extra trees algorithm is a variation of random forest. The difference is that random forest choosing the most discriminative thresholds from the random subset of candidate features, while extra trees randomly select some thresholds and pick the best one as the splitting rule.

E. Gradient Tree Boosting

The algorithm is an additive models with the following form

$$F_m(x) = \sum_{m=1}^M \gamma_m h_m(x) \quad (5)$$

where $h_m(x)$ is the weak classifier which is a fixed-size decision tree for the Gradient Tree Boosting algorithm.

At each stage the we choose a decision tree $h_m(x)$ that minimize the loos function L given the current model F_{m-1} :

$$F_m(x) = F_{m-1} + \gamma_m \arg \min_h \sum_{i=1}^n L(y_i, F_{m-1}(x_i) - h(x)) \quad (6)$$

Thus, we can solve the minimization problem by using gradient descent and the steepest descent direction is the negative gradient of the loss function:

$$F_m(x) = F_{m-1} + \gamma_m \sum_{i=1}^n \nabla_F L(y_i, F_{m-1}(x_i)) \quad (7)$$

Where the step length γ_m is

$$\gamma_m = \arg \min_{\gamma} \sum_{i=1}^n (\nabla_F L(y_i, F_{m-1}(x_i)) - \gamma \frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}) \quad (8)$$

F. Adaboost

The core idea of Adaboost algorithm is to learn a lot of weak classifiers with subset of the data. Then we assign a weight to each classifier and combine the weighted majority vote to produce the final prediction.

For this competition, we choose random forest and extra trees to be the "weak" classifier of Adaboost.

G. Bagging method

Bagging method takes several instances of a base estimator and produces random subsets of the training set to train each base estimator. At last, it takes the individual predictions of the base estimators and form the final prediction.

In our implementation, we used random forest, extra trees and decision tree as the base estimators and fed the three models to the next layer.

H. 3-Layer Ensemble Blending

As mentioned in previous subsections, algorithms which employ ensembling mechanism such boosting or bagging perform really well. However, they all seem to hit a plateau around 94.5% accuracy score. Similar to boosting technique, an intuitive idea is to find a way to make these great performers to work together in hope of achieving stellar score.

Inspired by blending algorithms popularized by Netflix competition and Neural Network, we introduce 3-layer blending algorithm.

- **Layer 1: Online training** In this layer, we collect an ensemble of high achievers consisting of 2 random forest classifiers, 2 extra tree classifiers, one gradient boosting machine classifier, one AdaBoost of random forest, one AdaBoost of extra tree classifier, one bagging classifier of random forest, one bagging classifier of extra tree classifier, one bagging classifier of a simple decision tree, 2 decision tree classifiers and 3 kNN classifiers. Each is trained over the dataset multiple times using 5-fold cross validation and everyone outputs the mean of their "soft" predictions (probability) rather than hard predictions (classes). Then, their results are combined together as a new dataset where each column representing one specific model guesses and each row is a data example.
- **Layer 2: Online Logistic Regression** inspired by Neural Network structure, we use logistic regression in the 2nd layer to regress layer 1 probabilistic guesses into the final "hard" predictions. In addition, we also compute the 3-fold cross validation score and save predictions and cv score to a file.
- **Layer 3: Offline aggregating** Over the course of the competition, we have trained literally dozen of models, some have very high cross validation score. By choosing only one best model, we lose most of our efforts doing extensive model experimentation in last couple of weeks and be unfair to only marginally worse models. As such, we come up with offline hard voting scheme where we load all quiz prediction files with high internal cross validation score and aggregate them all together by using majority voting scheme.

III. MODEL SELECTION

To select the best parameters, we uses 3-fold cross validation. The k-fold cross validation procedure is as followed:

- 1) Split the data into K parts.
- 2) For each value of \mathbf{h} :
 - For each $k \in 1, 2, \dots, K$
 - a) Train classifier $\hat{f}_{\mathbf{h},k}$ using all S_i except S_k .
 - b) Evaluate classifier $\hat{f}_{\mathbf{h},k}$ using S_k : $\text{err}(\hat{f}_{\mathbf{h},k}, S_k)$
 - Cross-validation error for \mathbf{h} : $\frac{1}{K} \sum_{k=1}^K \text{err}(\hat{f}_{\mathbf{h},k}, S_k)$
- 3) Select the value $\hat{\mathbf{h}}$ with lowest cross-validation error.

Cross validation is used to tune the parameters for each algorithms listed above. Then we choose the models with the fair performance and feed them into the blending algorithm to see if they contribute to the final result. At last, we choose a combination of models that produce the best result.

IV. PREDICTOR EVALUATION

In the beginning, we tended to base our algorithm decision on the Kaggle Public Leaderboard score. However, we quickly realized that it was not a good idea. By doing so, we are exposed ourselves to chances of overfitting the public leaderboard and prone to spending unproductive time twisting parameters to gain only minimal gain.

Gradually, we learn to trust our local cross validation. By using sklearn GridSearchCV to tune parameters, we are able to combine the task of tuning parameters with the quest of searching for a high cross validation score model. We discover that there is a high correlation between hiking the local cross validation score and boosting the public leaderboard score.

V. RESULT OF EVALUATION AND ANALYSIS

By spending many hours exploring and experimenting, we are able to come up with the 3-layer algorithm discussed in **Model Description**. The algorithm is able to accomplish 95.667% accuracy in the Kaggle Public Leaderboard and attain around 95.27% local 3-fold cross validation score. As you can see there is a slight discrepancy between local cv score and leaderboard score. This reflects our bias of trying to submit the results that increase the leaderboard ranking. However, the discrepancy is small enough to be negligible.

VI. INDIVIDUAL CONTRIBUTIONS

A. Bach Viet Do

Designed the whole architecture of predictive algorithms and worked on parameters tuning; Contributed to the report writing.

B. Keyu Lai

Tested the performances of different algorithm and worked on parameters tuning; Contributed to the report writing.

C. Qianyuan Chen

Experimented with different features selection and classification algorithms; Contributed to the report writing.

ACKNOWLEDGMENT

We would like to thank the instructor Satyen Kale and COMS4771 teaching staff for putting together a well -hought competition to stimulate us to apply machine learning theory into practice.

REFERENCES

- [1] Andreas Taoscher and Michael Jahrer , The Big Chaos Solution to the Netflix Grand Prize
- [2] <http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing>
- [3] <http://www.math.usu.edu/adele/RandomForests/Ovronnaz.pdf>
- [4] <http://scikit-learn.org/stable/modules/ensemble.html>
- [5] <http://www.satyenkale.com/coms4771/content/lec3-nn.pdf>
- [6] <http://www.satyenkale.com/coms4771/content/lec4-dt.pdf>
- [7] <http://www.satyenkale.com/coms4771/content/lec12-boosting.pdf>