**Kaggle Classification Competition: Cat vs Dog** **Zhu Xiangqi**
High Dimensional Data Analysis No.2120406011
Prof. Wang Yao Lab Date: 01/04/2019
Class: Statistics 51 Due Date: 01/07/2019

## 1 Problem Statement

This is a competition on Kaggle in 2013. In this competition, each attender is required to write an algorithm to classify whether images contain either a dog or a cat. This seems easy for humans, dogs, and cats, but as for computer, it becomes a bit more difficult. Apparently, this is a typical binary classification problem. The training set and input test set are cat or dog images; the labels are binary: cat and dog. The problem can be divided into 3 specific parts:

- How to extract most useful and effective features from training images.
- Which models should be built to fit images and labels.
- How to evaluate the performance of these different models.

This is also how this report is organized.

## 2 Dataset Description

The test set can be download from Kaggle official website (https://www.kaggle.com/c/dogs-vs-cats). There are 25000 pictures in the test set, including 12500 cats and 12500 dogs. The picture sizes are completely random because all pictures are downloaded from internet.
The training set is given in the *Dog VS Cat.zip*. Not as the test set, the training set contends a small part of outliers, which means pictures that are not cats nor dogs either, and they add difficulty to the modeling. In that case, the stability of the model versus outliers is highly demanded.

## 3 GoogLeNet Getting Features

To extract useful features from the training dataset effectively, the prevailing deep learning method *GoogLNnet* is applied.
GoogLeNet is a new deep learning structure proposed by Christian Szegedy in 2014. Before that, AlexNet, VGG and other structures obtained better training effect by increasing the depth of the network (layers), but the increase of layers will bring many negative effects, such as overfitting, gradient disappearance, gradient explosion and so on. Inception is proposed to improve the training results from another perspective: more

efficient use of computing resources, in the same amount of computing can extract more features, thereby improving the training results. In other words, GoogLeNet is regarded as the most efficient way to extract data features.
The ImageNet project is a large visual database designed for use in visual object recognition software research. More than 14 million images have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided. Figure 1 shows some pictures of cats on imagenet, while Figure 2 shows some pictures of dogs on imagenet.

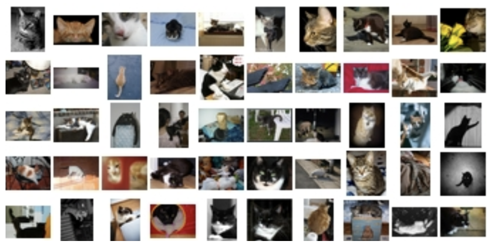
Figure 1: Cats data in ImageNet

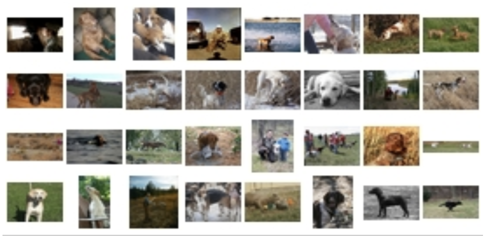
Figure 2: Dogs data in ImageNet

GoogLeNet is a convolutional neural network that is trained on more than a million images from the ImageNet database. The network is 22 layers deep and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224. The GoogLeNet can be used as one of pretrained networks in MATLAB. The following are the characteristics of GoogLeNet[1]:

- A 1×1 convolution with 128 filters for dimension reduction and rectified linear activation.
- A fully connected layer with 1024 units and rectified linear activation.
- A dropout layer with 70% ratio of dropped outputs.

With the pre-trained GoogLeNet model of Matlab2017b, the features of cat and dog images can be extracted. Finally, 1024 features per image in total are extracted after GoogLenet training.

## 4 Classifiers

In this part, 3 different classifiers are applied to solve the problem: AdaBoost, GBDT and XGBoost.

## 4.1 Adaboost

Adaboost is an iterative algorithm. Its core idea is to train different classifiers (weak classifiers) for the same training set, and then aggregate these weak classifiers to form a stronger final classifier (strong classifier). AdaBoost[2] is a general classifier lifting algorithm, which uses classifiers that are not limited to a particular algorithm.
The algorithm itself is realized by changing the data distribution. It determines the weight of each sample according to whether the classification of each sample in each training set is correct or not, and the accuracy of the last overall classification. The new data set with modified weights is sent to the lower classifier for training. Finally, the classifier obtained from each training is fused as the final decision classifier. The algorithm is given:

***Algorithm*** AdaBoost

```
1. Initialize weights $w_i = \frac{1}{n}$
2. for m=1 to $M$ do
3.     fit $y = h_m(x)$ as the base weighted classifier using $w_i$ and $d$
4.     let $W_{h_m} = \sum_{i=1}^{N} w_i I\{y_i h_m(x_i) = -1\}$ and $a_m = log(\frac{1-W_h}{W_h})$
5.     $w_i = w_i \exp\{a_m I\{y_i \neq h_m(x_i)\}\}$ scaled to sum to one $\forall i \in \{1, ..., N\}$
6. end for
```

## 4.2 GBDT

GBDT is also a member of the Boosting family of integrated learning, but it is quite d-ifferent from the traditional Adaboost. Looking back at Adaboost, we use the error rate of the previous iteration weak learner to update the weight of the training set, so that the iteration of this round continues. GBDT is also iteration, using forward distribution algorithm, but the weak learner restricts the use of only CART regression tree model, at the same time, iteration ideas and Addaboost are different.
In the iteration of GBDT, suppose that the strong learner obtained by our previous iteration is $f_{t-1}(x)$. The loss function is $L(y, f_{t-1}(x))$. The goal of this iteration is to find a weak learner $h_t(x)$ of the CART regression tree model, so that the loss function L$L(y, f_t(x)) = L(y, f_{t-1}(x) + h_t(x))$ of this round is the smallest. That is to say, this round of iteration to find the decision tree, to make the sample loss as small as possible. And that is why it performs relatively well on cross validation but not as expected on test dataset, which would be discussed in part 5.

## 4.3 XGBoost

XGBoost is the abbreviation of Extreme Gradient Boosting. Gradient Boosting is introduced by *Greedy Function Approximation: A Gradient Boosting Machine*[3]
More generally, for general loss functions, XGBoost uses Taylor expansion to use second

derivatives. Traditional GBDT only uses the first derivative information in optimization, while XGBoost expands the cost function with the second Taylor expansion, and uses the first and second derivatives at the same time. By the way, the XGBoost tool supports custom cost functions as long as they can derive first and second orders.
The $t$th loss function:

$$L^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{t-1} + f_t(x_i)) + \Omega(f_t)$$

Second-order Taylor expansion of the upper formula: $g$ is the first derivative and $h$ is the second derivative.

$$L^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{t-1}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

where,

$$g_i = \partial_{\hat{y}t-1} l(y_i, \hat{y}_i^{t-1})$$
$$h_i = \partial^2_{\hat{y}t-1} l(y_i, \hat{y}_i^{t-1})$$

Additionally, XGBoost adds a regular term to the cost function to control the complexity of the model. The regular term contains the number of leaf nodes of the tree, the square of L2 modules of score output on each leaf node. From the Bias-variance tradeoff point of view, the regular term reduces the variance of the model, makes the learning model simpler and prevents over-fitting, which is also a feature of XGBoost better than traditional GBDT.

## 5 Performance and Comparison

### 5.1 Training Performance

the random seed of spliting validation dataset is set from 0 to 9. The parameter of *max_depth* is uniformly 5, and the parameter of *learning_rate* is also set uniformly 0.1. As there are so many outliers in training dataset, the parameter of *nfold* is set 10 to in case that outliers are separated too imbalanced. The concerned value of error rate are average error rate after 10 times validation and the standardized variance is used to inspect the sensitivity and stability of each model versus different proportion of outliers.
From Table 1, For the cross validation, the average error rate of AdaBoost is 12.859%; the average error rate of GBDT is 5.585%; the average error rate of XGBoost is 10.687%. the standardized error rate variance of AdaBoost is 2.622%; the standardized error rate variance of GBDT is 2.979%; the standardized error rate variance of XGBoost is 0.774%. There is no doubt that according to average error rate, the GBDT performs the best in validation. But the standardized variance of GBDT is also huge, which might refers that GBDT could be sensitive to outliers.

Table 1: The training-validation error rate(%) for every classifier

| Classifiers | AdaBoost | GBDT | XGBoost |
|---|---|---|---|
| Average Error Rate(%) | 12.859 | **5.585** | 10.687 |
| Error Rate Standard Variance(%) | 2.622 | 2.979 | **0.774** |

## 5.2 Test Performance

The test set is downloaded from Kaggle official website with 12500 cat images and 12500 dog images. For each classification method, 10 individual model based on different random seed in training is given. And the average error rate and standardized error rate variance are considered.

From Table 2, the average error rate of AdaBoost is 3.656%; the average error rate

Table 2: The test error rate(%) for every classifier

| Classifiers | AdaBoost | GBDT | XGBoost |
|---|---|---|---|
| Average Error Rate(%) | 3.656 | 3.843 | **3.517** |
| Error Rate Standard Variance(%) | **o(e-16)** | 0.066 | 0.179 |

of GBDT is 3.843%; the average error rate of XGBoost is 3.517%. the standardized error rate variance of AdaBoost is less than $10^{-18}; thestandardizederrorratevarianceofGBDTis0.066\%; thestan$
$TheGBDTmethodhasfascinatingaverageerrorrate5.585\%asshowninvalidationwhileitdoesnotactasexpe$

## 5.3 Runtime Comparison

The considered runtime is the model training time plus test time. The random seed is set from 0 to 9. And the runtime value concerned is the average runtime and standardized variance of runtime after 10 times fitting and testing.

From Table 3, the average runtime of AdaBoost is 16.317 sec; the average runtime of

Table 3: The running time(seconds) for every classifier

| Classifiers | AdaBoost | GBDT | XGBoost |
|---|---|---|---|
| Average Runtime(sec) | 16.317 | 5.158 | **2.053** |
| Runtime Standard Variance(sec) | 3.752 | 0.263 | **0.082** |

GBDT is 5.158 sec; the average runtime of XGBoost is 12.053 sec. the standardized runtime variance of AdaBoost is 3.752 sec; the standardized runtime variance of GBDT

is 0.263 sec; the standardized runtime variance of XGBoost is 0.082 sec.
Noted that if you directly run the given example code of GBDT or AdaBoost package, the runtime might show that the time consuming of GBDT and AdaBoost is almost 100-300 times of XGBoost total runtime. That is because the default values of parameter *n_estimators* are different in these 3 packages. After some simple test, the speed of converging is much higher than defaulted, so the parameter could alter from 300 or 1000 to only 10, which highly improved the speed of models.
The inner iteration of AdaBoost model is much more than other 2 methods, so the runtime is significantly increased compared to others. And the total work and time complexity of XGBoost should be lower than GBDT and apparently, lower than AdaBoost. Also through the comparison of runtime variance, the training of XGBoost is more stable, which means amount of outliers and division of training set and validation set make less different on it.

## 6 Conclusion

It is obvious that XGBoost wins both the highest classification accuracy (3.517% average error rate) and the highest speed (2.053 sec average runtime).
It is worth concerning that, as in Table 1, the cross validation performance of GBDT is significantly better than XGBoost, but when the model comes to solving the test problem, the accuracy of GBDT is not as good as expected. Here are two plausible explanations: one is that the parameters of GBDT model are not all adjusted perfectly so there might be some overfitting problems; the other is that the XGBoost is more stable with outliers in training dataset, so the XGBoost model would not be affected by these outliers as much as tree methods, like GBDT. Plus, the complexity of XGBoost model could be much smaller than GBDT model, so the running speed is also much improved in XGBoost model.

## References

[1] Szegedy C , Liu N W , Jia N Y , et al. Going deeper with convolutions[C]. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE Computer Society, 2015.

[2] Wu B, Ai H, Huang C, et al. Fast rotation invariant multi-view face detection based on real Adaboost[C]. IEEE International Conference on Automatic Face Gesture Recognition. 2004.

[3] Friedman J H . Greedy Function Approximation: A Gradient Boosting Machine[J]. The Annals of Statistics, 2001, 29(5):1189-1232.