Your Job Role Determines Your Access Privileges*

Yunfeng Jiang, Jinzheng Sha, and Zhao Tian

1 Introduction

In any company, when employees start work, they need to obtain the necessary computer access in order to fulfill their job. This access might allow an employee to read or manipulate resources through various applications or web portals. For example, a software engineer needs the access to the source code repositories. Conventionally, the access is granted through trial and error—employees figure out the access they need as they encounter roadblocks during their daily work, e.g., not able to log into a reporting portal. That practice is inefficient because the task takes a knowledgeable supervisor plenty of time to manually grant the needed access in order to overcome access obstacle. At the same time, the supervisor must avoid granting unnecessary access for the reason of security. As employees move throughout a company, this access discovery/recovery cycle wastes a nontrivial amount of time and money [1].

Employees who perform the functions of the same job role should access the same or similar resources. So we can build a model, learned using historical data, that will predict an employee's access needs. The model will take an employee's role information and a resource code and will return whether or not access should be granted. The problem can be formulated as a binary classification problem.

We applied different classification techniques to approach this problem. We would assemble those sub-models to form a combined model. The idea comes from the intuition that different classifiers might have better performance on different cases, so a combined model should have better overall performance. Before combining all the models, we need to first select the best sub-model. So, we used cross validation to select the best hyper parameters for each model in the first place. Specifically, we tried Logistic Regression, Naive Bayes, Random Forest, Extremely Randomized Trees [6], and Gradient Boosted Decision Trees. Afterward, we "sum" all the models with weights assigned by different methods: average, Logistic Regression, and LP-β [5]. Our results show that the combined models have better accuracy than most of the individual classifiers on test data.

*The problem is inspired by the Amazon's Employee Access Challenge. http://www.kaggle.com/c/amazon-employee-access-challenge

Table 2.1: Data Set Columns [2]

Column Name	Description
ACTION	ACTION is 1 if the resource was approved, 0 if the resource was not
RESOURCE	An ID for each resource
MGR_ID	The EMPLOYEE ID of the manager of the current EMPLOYEE ID record; an employee may have only one manager at a time
ROLE_ROLLUP_1	Company role grouping category id 1 (e.g. US Engineering)
ROLE_ROLLUP_2	Company role grouping category id 2 (e.g. US Retail)
ROLE_DEPTNAME	Company role department description (e.g. Retail)
ROLE_TITLE	Company role business title description (e.g. Senior Engineering Retail Manager)
ROLE_FAMILY_DESC	Company role family extended description (e.g. Retail Manager, Software Engineering)
ROLE_FAMILY	Company role family description (e.g. Retail Manager)
ROLE_CODE	Company role code; this code is unique to each role (e.g. Manager)

But the smarter weight assignment methods do not outperform the average method. So, in the next step we will fine-tune those two combination model using model selection and expect that they should higher test accuracy than the average method.

2 Initial Analysis of the Data

The data we use consist of real historical data collected from 2010 and 2011, provided by Amazon [2]. In this dataset, employees are manually allowed or denied access to resources over time. In the training set, each row has the ACTION as ground truth (ACTION is 1 if the resource was approved, 0 if the resource was not), RESOURCE, and information about the employee's role at the time of approval. There are 7518 different resources to be granted. In order to describe a job role, the dataset employs 9 categorical features, such as his/her manager (4243 possible values), department names (449 possible values), and role title (343 possible values). The training set totally contains 32769 lines of access data related to current employees and their provisioned access. The testing set contains 58921 lines, which involve 4971 different resources.

In our training data, there are nine different features (Table 2.1). Because the name of the colors are vague and all the content is anonymous (integers instead of words for the descriptions), we find it hard to fathom the relations between the columns. For example, what is the difference between ROLE_TITLE and ROLE_CODE? We suspect that they can be the same thing. So we use the pairwise scatter plot to figure out which variable is redundant between different features (Figure 2.1). In the figure, we can see there is a clearly linear pattern between the feature ROLE_TITLE and the ROLE_CODE which means we can use only one of the two features. For features like MGR_ID and RESOURCE, there are much higher cardinality than other features.

Figure 2.1: Pairwise plots of the feature columns

3 Methods

In the dataset, there are 8 categorical features of each employee to determine whether a certain resource should be granted to the employee or not. It can be modeled as a classification problem and many methods are available to solve the problems, such as Logistic Regression and Naive Bayes. Considering that different classifiers might have different discriminative powers and in order to take advantage of a few methods, we chose to combine a bunch of sub-models [5]. We will train our combined model via a two-stage training. In the first stage, we train each individual model and select the best hyperparameters for each single model individually through cross validation. In the second stage, we consider the results of individual models as new features, train the combined model, and assign weights to individual models.

Single models. Based on how we process the features, the methods can be classified into two categories. In the first category, we incorporate second order and third order features as well as using one-hot encoding, because some models can only capture linear relations. In the second category, we use the features as is, because the models themselves can capture non-linear relations.

Sparse features. In the training set, all the features are categorical variables which means the values in the feature represent a set of categories. In order to employ classifiers like Naive Bayes and Logistic Regression, we need to use one-hot encoding to get the sparse features. What's

more, there can be underlying correlations among different features, so we would like to use second and third order features generated by a hash function. As a result, there are tons of features for naive Bayes and logistic regression. So, before dumping all the available features into those two models, we selected a subset of features using greedy forward selection. Because we employ higher oder feature, in oder to avoid overfitting, we incorporate a regularized term into those models.

Non-sparse features. For some models, we can directly use the features as is, because the models themselves can capture non-linear relations among features, such as Random Forests, Extremely Random Trees and Gradient Boosted Decision Trees. For those ensemble methods, we can use the original non-spares features to make the classification without feature selection in the first place. The Random Forest is a classifier consisting of a collection of tree using the random selected training data and random selection of features at each split. Extremely randomized trees are similar to random forests but more random by using both a variable index and variable to split value in a node split. When using these tree models, in order to enrich the features we also constructed some new features by computing some simple statistics over the the original features, such as the frequency of the value in each column because they capture the overall distribution of the features which provides more information about the training data.

Model combination. After fine-tuning the single models, we hope to combine them and expect better performance and stability than single models. It is a feature combination problem and we need to assign weights to each individual model in the summation [5].

Naive average. Finally, inspired by idea of the ensemble methods, we combine different models to obtain a better predictive performance than each individual sub-model. For now, we just assign each model an equal weight:

$$f(x) = \frac{1}{F} \sum_{m=1}^{F} f_m(x) \ ,$$

where F is the number of single models and $f_m(x)$ represents each individual classifier and in our method, they are Logistic Regression (LR), Naive Bayes, Random Forest (RF), Extremely Randomized Trees (XT), and Gradient Boosted Decision Trees (GB) respectively.
Logistic Regression. We also consider using Logistic Regression to combine those individual models:

$$f(x) = g(\theta_0 + \sum_{m=1}^{F} \theta_m f_m(x)) \ ,$$

where g is the sigmoid function.

LP-β [1]. In [5], LP-β is proposed to train the combined model:

$$f(x) = \text{sign} \sum_{m=1}^{F} \beta_m f_m(x) \ ,$$

[1] This method is suggested by Professor Lorenzo Torresani and we really appreciate it.

and the weight coefficients are optimized using the following linear program:

$$\begin{aligned} \min_{\beta,\xi,\rho} \quad & -\rho + \frac{1}{\nu}\sum_{i=1}^{N}\xi_i \\ \text{s.t.} \quad & y_i \sum_{m=1}^{F}\beta_m f_m(x_i) + \xi_i \geq \rho,\ i = 1,\ldots,N\ , \\ & \sum_{m=1}^{F}\beta_m = 1,\ \beta_m \geq 0,\ m = 1,\ldots,F\ , \end{aligned}$$

with ξ being slack variables and ν being the only hyperparameter, which we can make an analogy to the SVM we covered in class. We handpicked $\nu = 0.5$ in our current implementation.

Two-stage training. In order to make our combined model achieve its potential to its fullest, we first need to tweak each of the sub-model to avoid underfitting or overfitting. So, we use N-fold cross validation ($N = 10$) to determine the the hyper parameters, such as the regularization terms and the depth of the trees[2]. After settling on the individual models, we will train the proposed combination models using the outputs of five individual models as features.

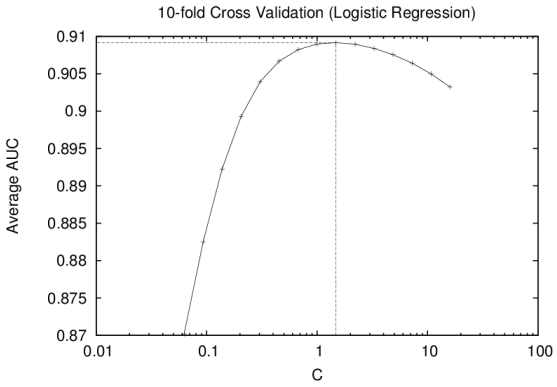
4 Results

In this part, we will illustrate the results of the cross validations and show the best hyper parameters indicated by those results. Then, we combined those tweaked sub-models using the above form and compared its accuracy with all the individual models.

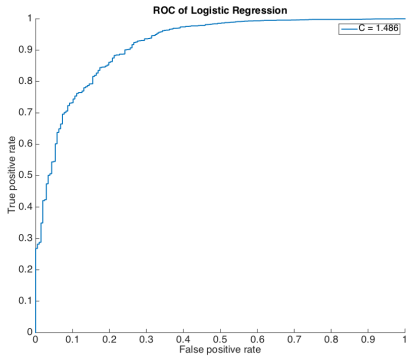
In order to compare the performance of those classifiers, we employ a metric called receiver operating characteristic curve, i.e. ROC curve (Figure 4.1 (b)). The curve is created by plotting the true positive rate against the false positive rate at various thresholds. Then we calculate the area under the curve, i.e. AUC score. The higher the score, the better the classifier. An area of 1 represents a perfect classifier, while an area of 0.5 represents a worthless one [4].

Specifically, as you can see in the results of 10-fold cross validation for logistic regression (Figure 4.1 (a)), we vary the value of C, which is the inverse of regularization strength. So smaller values indicate stronger regularization. In our case, logistic regression perform the best when C equals 1.486. And we can observe over-fitting and under-fitting with smaller and bigger C respectively. The ROC curve for this optimal C is also shown long. And similar analysis can be performed on the corresponding figures of naive Bayes (Figure 4.2).

[2]We made a mistake by using the number of trees as the hyper parameter for those tree-based models. But we now have known since the milestone that the number of trees will never cause overfitting. So we changed the hyperparameter to the depth after the milestone. But we will show both the results using the number of trees the depth of trees.

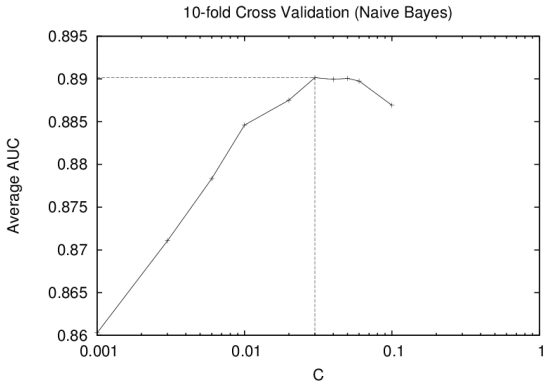


(a) Cross Validation

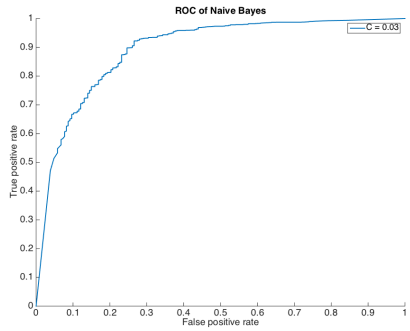


(b) ROC

Figure 4.1: Logistic Regression



(a) Cross Validation



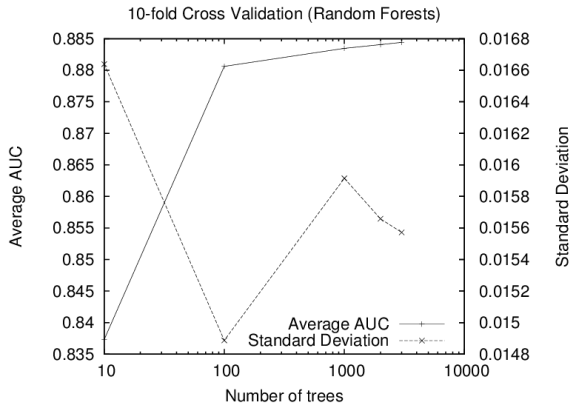
(b) ROC

Figure 4.2: Naive Bayes

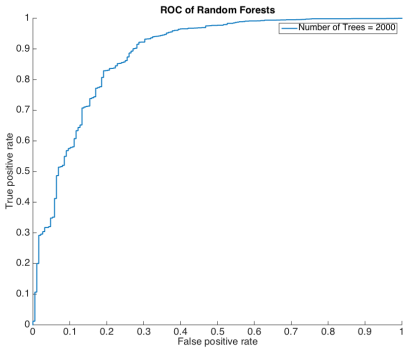
In terms of tree-based models, before the milestone we varied the number of trees [3] to achieve better performance. In the results of random forests, there are two lines in the left graph. The solid line represents the relationship between average AUC and the number of trees. While the dotted line stands for the standard deviation of the cross validation results. It is obvious that the more trees we use, the better results we get.

As mentioned above, we made a mistake by selecting the tree-based model over the number of trees. After the milestone, we re-select the tree-based model over the depth of trees. The cross validation results are shown in Figure 4.5. We can draw two conclusions from the result. First, adding statistic features improve the performance the tree-based models, because more information about the feature distribution is provided. Second, only the Gradient Boosted Decision Trees showed the sign of over-fitting when the depth went above 15. The cause might be that we set the number of trees to 50 for the Gradient Boosted Decision Trees while we set 600 for the other two tree-based models. The increase of number of trees is able to prevent the over-fitting

[3] We should have used the depth of the trees instead. We will fix this mistake.

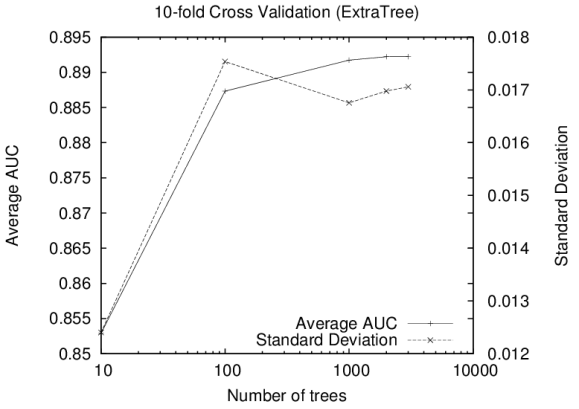


(a) Cross Validation

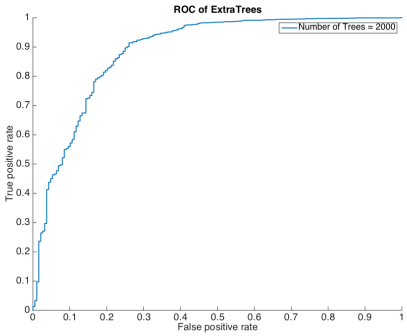


(b) ROC

Figure 4.3: Random Forests



(a) Cross Validation



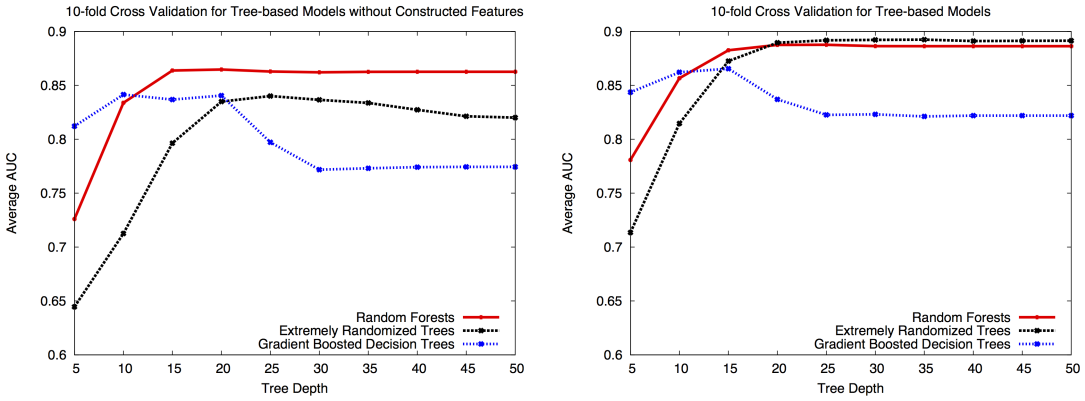
(b) ROC

Figure 4.4: ExtraTrees

when we increase the depth of the trees.

After getting the best performance on individual models, we combined those models such that the combined one can classify real data with higher accuracy and robustness. But how can we do that? Naively, we use linear combination with equal weights for different models. Surprisingly the combined model has a higher mean AUC score than any individual model. As you can see in Figure 4.6, the right-most error-bar stands for it. And our following work is to find a better way to combine individual models. For example, we would like to learn the linear combination weights based on their performance rather than hardcode them.

Seeing that the naive average has better performance than the best individual model, we want to try smarter ways to combine those models and expect further improvement. As discussed in the previous section, we employed Logistic Regression and LP-β to smartly assign the weights to each individual model. The weights we obtained by using those three combination methods



(a) Without constructed features

(b) With constructed features

Figure 4.5: Tree-based model selection using depth of trees

Table 4.1: Weights assigned to individual models using different combination methods

Method	**LR**	**NB**	**RF**	**XT**	**GB**
Naive Average	0.2	0.2	0.2	0.2	0.2
LR	0.0656	0.0604	0.109	0.134	1.28
LP-β	0.0002	0.6131	0.0002	0.0001	0.3864

are shown in Table 4.1. We can see that different combination methods will assign different weights to the individual models.

We assess our individual models as well as combined models using the test dataset (Figure 4.7). Interestingly, the naive average combination shows the best performance among all models. For now, we only handpicked the hyperparameter for the LP-β model, but we expect that if we use cross validation to select hyperparameter it should show better performance than the naive average combination. Due to time limitation, we will leave this as future work.

5 Conclusion

We built combined models to predict access needs in a company and we trained the combined model through a two-stage training. Our results show that the combined models have better performance than most of the individual features. We previously only used training error to select our model, but the accuracy on test data might be different. This conclusion can help us avoid the selection of the worst model. In this sense, the combined model improves the performance and stability of individual models.

6 Future Work

As we analyzed in the previous section, neither of the combination models using Logistic Regression and LP-β outperforms the naive average combination on the test dataset. The reason

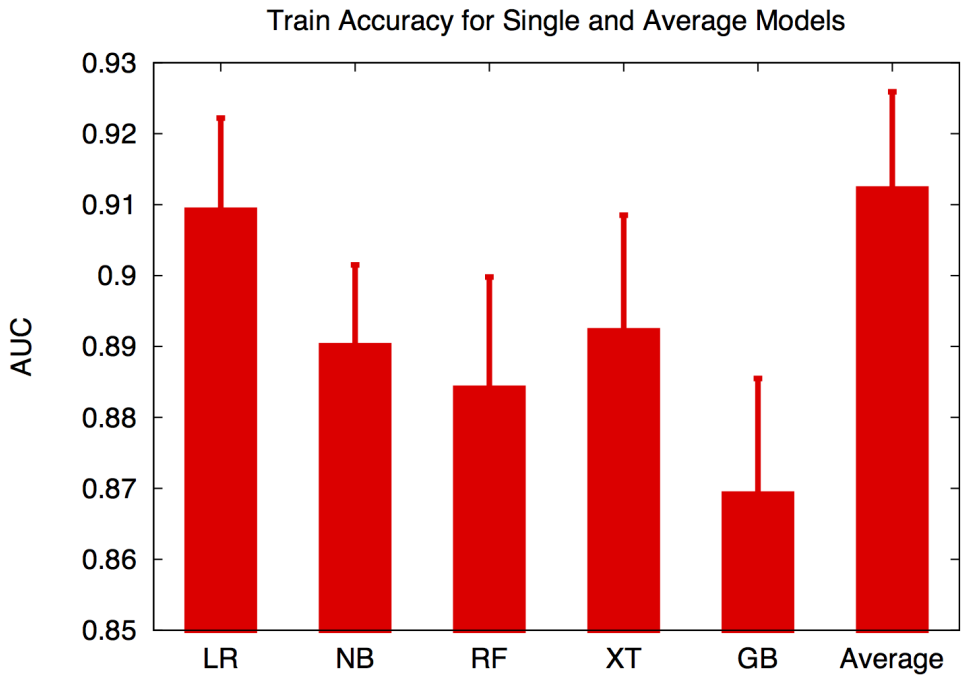


Figure 4.6: Comparison of individual models and the combined model on training set

might be that we did not employ cross validation to select hyperparameter for the combined models. In the future, we would perform model selection on the combined models in addition to the individual models and expect the smarter weight assignment methods should outperform the naive average.

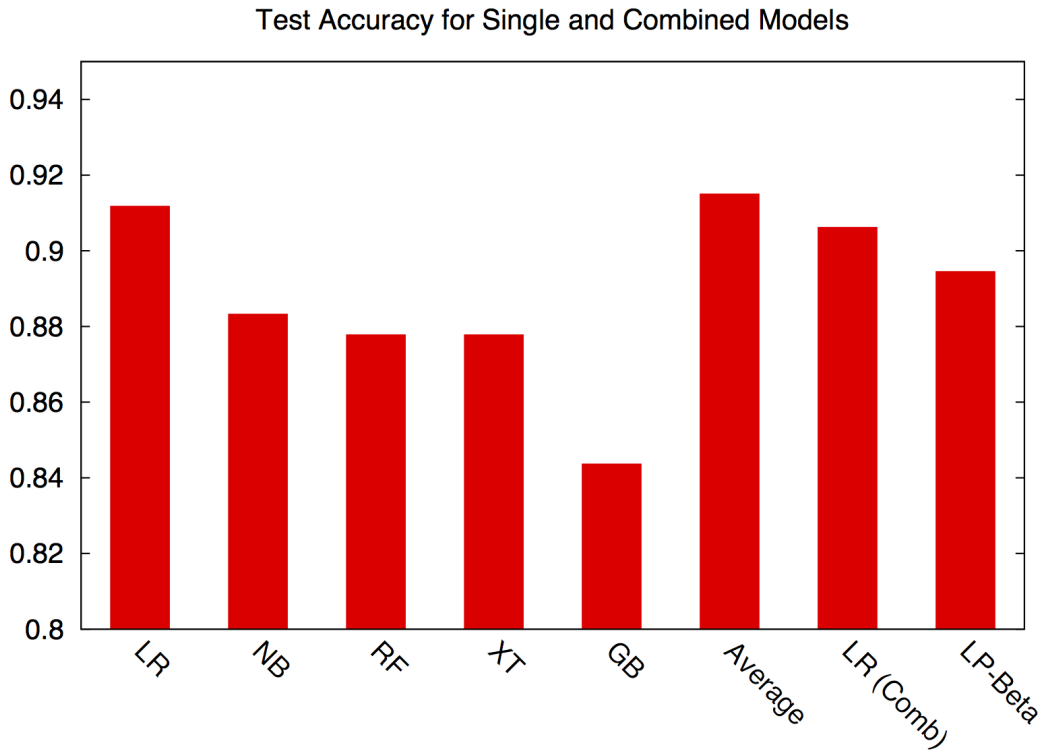


Figure 4.7: Comparison of individual models and the combined model using on test set

7 Implementation Details

When training the five individual models, we used the external machine learning library scikit-learn [7]. We implemented the feature preprocessing and cross validation by ourselves.

1. `LogisticRegression`

   In order to use the Logistic regression model for our categorical variables, we need to use the one-hot encoding to transform our categorical variables to binary variables. We use the OneHotEncoder (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) method in scikit-learn package to make the transformation. For the Logistic regression model we use the LogisticRegression (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) method in scikit-learn package.

2. `naive_bayes`

   For the Naive Bayes model, we use the same one-hot encoding in scikit-learn package. We also use the naive_bayes (http://scikit-learn.org/stable/modules/naive_bayes.html) method

in scikit-learn package.

3. `RandomForestClassifier`

   For the Random forest model, we use the RandomForestClassifier (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html) method in scikit-learn package.

4. `ExtraTreesClassifier`

   For the Extra Trees model, we use the ExtraTreesClassifier (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html) method in scikit-learn package.

5. `GradientBoostingClassifier`

   For the Gradient Boosting model, we use the GradientBoostingClassifier (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html) method in scikit-learn package.

When training the combined models, the naive average does not need training and the Logistic Regression is trained using scikit-learn. The training of LP-β is implemented by ourselves in MATLAB using the linear solver `linprog`. Further implementation details can be found the source codes we submit.

References

[1] Amazon employee access challenge, http://www.kaggle.com/c/amazon-employee-access-challenge.

[2] Data set, Amazon employee access challenge, http://www.kaggle.com/c/amazon-employee-access-challenge/data.

[3] ExtraTreesClassifier, http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html.

[4] The Area Under an ROC Curve, http://gim.unmc.edu/dxtests/roc3.htm.

[5] P. Gehler and S. Nowozin. On feature combination for multiclass object classification. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 221–228. IEEE, 2009.

[6] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.