Milestone
Your Job Role Determines Your Access Privileges*

Yunfeng Jiang, Jinzheng Sha, and Zhao Tian

1 Introduction

At 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 performance on all cases. 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 forests, extremely randomized trees [5], and gradient boosting. Afterward, we just "sum" all the models with equal weight. Our results show that the combined model have the best accuracy among all the classifiers.

*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 improvement of the accuracy is quite marginal. So, in the next step we will tweak the weights as well as try different form of the combination, such as logistic regression, in order to further improve the performance in terms of accuracy.

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 categories of 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 different 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. In order to take advantage of a few methods, we chose to combine a bunch of sub-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 boosting. 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 Forests 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.

Naive combination 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:

$$P_{\text{combined}} = \frac{P_{\text{LR}} + P_{\text{NB}} + P_{\text{RF}} + P_{\text{XT}} + P_{\text{GB}}}{5} ,$$

where P_{LR}, P_{NB}, P_{RF}, P_{XT}, and P_{GB} are the outputs of logistic regression, naive Bayes, random forests, extremely randomized trees, and gradient boosting respectively. 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 [1].

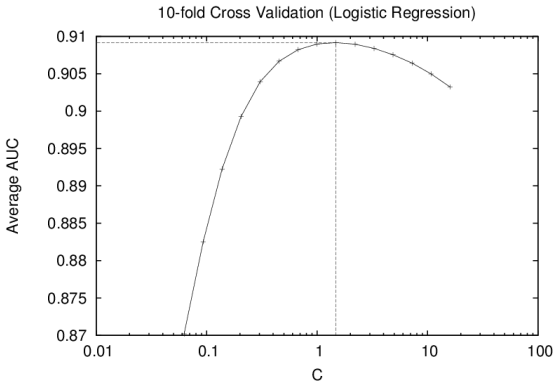
4 Results

We implemented some classifiers as well as used some existing libraries to do the model selection. 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.

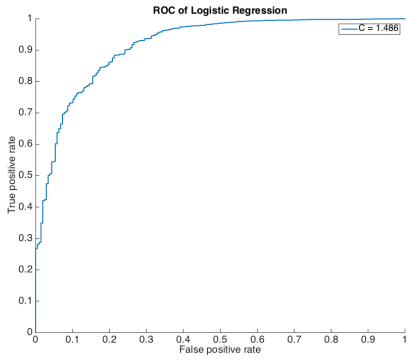
In order to compare the performance of those classifiers, we employ a metric called receiver operating characteristic curve, i.e. ROC curve (Figure 4.1 (b)). The curve is created by plotting the true positive rate against the false positive rate at various thresholds. Then we calculate the area under 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).

[1] We made a mistake by using the number of trees as the hyper parameter for those tree-based models. But we now know that hyper parameter will never cause overfitting. So we are going to do cross validation over the depth of the trees in the next half of the project.

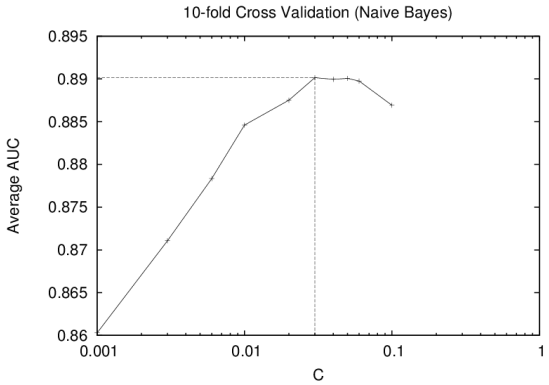


(a) Cross Validation

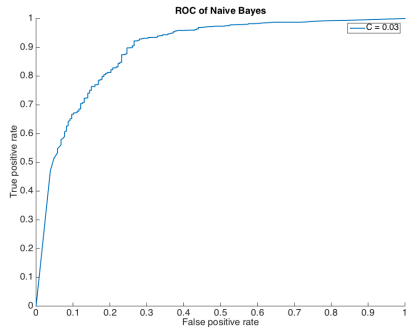


(b) ROC

Figure 4.1: Logistic Regression



(a) Cross Validation



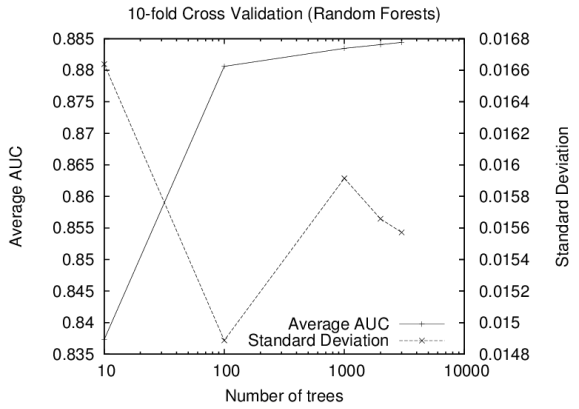
(b) ROC

Figure 4.2: Naive Bayes

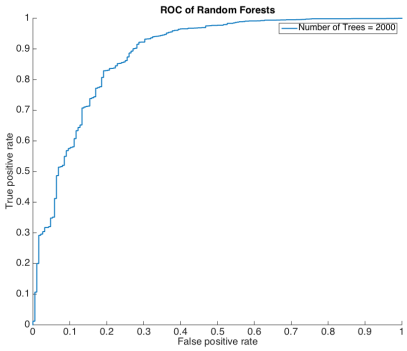
In terms of tree-based models, we decide to vary the number of trees [2] to achieve better performance. In the results of random forests, there are two lines in the left graph. The solid line represents the relationship between average AUC and the number of trees. While the dotted line stands for the standard deviation of the cross validation results. It is obvious that the more trees we use, the better results we get. Considering both the performance and the computational cost, however, we settled on 2000. Shown on the right is the corresponding ROC curve. Similarly, we also draw the cross validation result for extremely randomized trees (using the ExtraTrees package [3]) classifier and set the number of trees to 2000.

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 4.5, the right-most error-bar stands for it. And our following work is to find a better

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

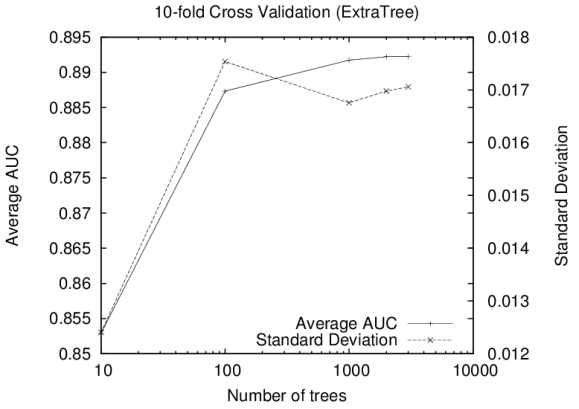


(a) Cross Validation

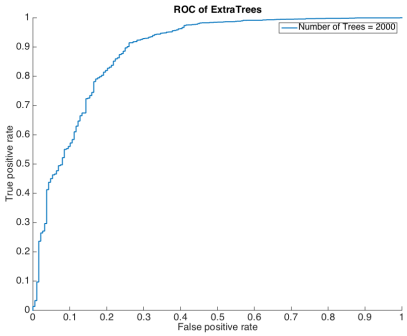


(b) ROC

Figure 4.3: Random Forests



(a) Cross Validation



(b) ROC

Figure 4.4: ExtraTrees

way to combine individual models. For example, we would like to learn the linear combination weights based on their performance rather than hardcode them.

5 Future Work

Tweak individual models As mentioned above, in some cross validations, we chose the incorrect hyper parameters, such as the number of trees in random forests and extremely random trees. So, we will keep tweaking our sub-models. Besides, we will implement as many classifiers as possible by ourselves.

Complicated combination form The linear combination of sub-models with equal weights is very naive, resulting in only slightly accuracy improvement. We will try to further improve the accuracy by applying complicated combination forms, such as logistic regression.

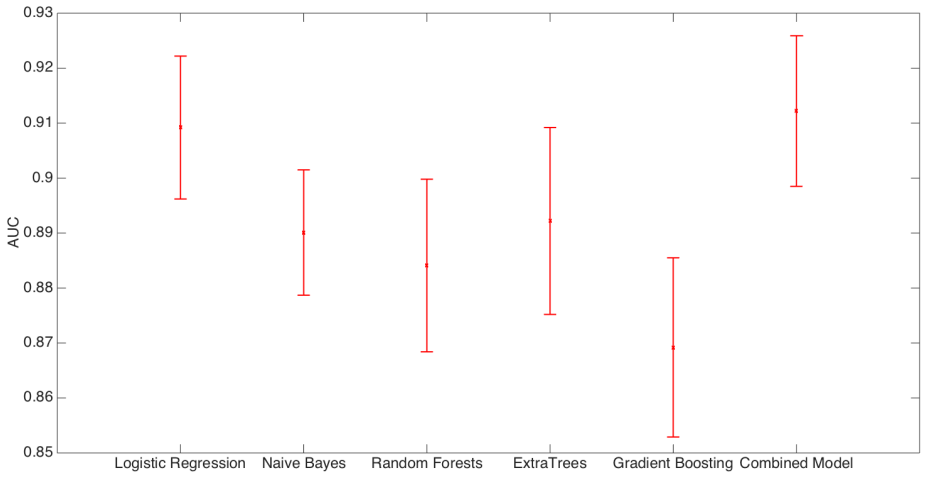


Figure 4.5: Comparison of Individual Models and the Combined Model

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