Udacity Machine Learning Nanodegree
Capstone Project

Carsten Krüger

June 30, 2018

1 Definition

1.1 Project Overview

Machine learning is used in a wide variety of fields today. Luca Talenti et al. [1] for example used a classification model to predict the severity criteria in imported malaria. In this project, machine learning will be used to build a model that can decide based on the role information of an employee whether that employee shall have access to a specific resource.

An employee that has to use a computer in order to fulfill their tasks, needs access to certain areas of software programs or access rights to execute actions such as read, write or delete a document. While working, employees may encounter that they don't have a concrete access right required to perform the task at hand. In those situations a supervisor or an administrator has to grant them access. The process of discovering that a certain access right is missing and removing that obstacle is both time-consuming and costly. A model that can predict which access rights are needed based on the current role of an employee is therefore relevant.

1.2 Problem Statement

The problem stems from the *Amazon.com Employee Access Challenge Kaggle Competition* [2] and is there described as follows:

"The objective of this competition is to build a model, learned using historical data, that will determine an employee's access needs, such that manual access transactions (grants and revokes) are minimized as the employee's attributes change over time. The model will take an employee's role information and a resource code and will return whether or not access should be granted."

This is a supervised learning problem because the dataset is labeled. Anticipated solution:

1. Explore data in order to gain insights.

2. Train many different binary classification models using standard parameters.

3. Apply transformations or regularizations.

4. Compare plain models and transformed models.

5. Pick the three best models based on the performance metric.

6. Tweak the chosen models in order to improve model performance.

7. Evaluate the tweaked models on the test set.

8. Conclusion

1.3 Metrics

To quantify model performance, the area under the ROC curve will be used. This metric is appropriate for this type of project because it works well even if the classes are not balanced. Moreover it was the metric of choice in the herein before mentioned Kaggle competition. The metric is derived by first constructing the ROC curve and then calculating the area under that curve.

"The ROC curve is created by plotting the true positive rate against the false positive rate at various threshold settings" [3],

where the threshold is a value between 0 and 1 that determines how sure the model needs to be in order to classify a data entry as positive (access granted in the problem at hand). For example if the threshold was 0.7 the model would have to have calculated a probability of at least 70 % to classify a data entry as positive.

2 Analysis

2.1 Data Exploration

There are 32769 entries in the dataset with no missing values. Figure 1 shows the first five rows in the dataset.

```
   ACTION  RESOURCE  MGR_ID  ROLE_ROLLUP_1  ROLE_ROLLUP_2  ROLE_DEPTNAME  \
0       1     39353   85475         117961         118300         123472
1       1     17183    1540         117961         118343         123125
2       1     36724   14457         118219         118220         117884
3       1     36135    5396         117961         118343         119993
4       1     42680    5905         117929         117930         119569

   ROLE_TITLE  ROLE_FAMILY_DESC  ROLE_FAMILY  ROLE_CODE
0      117905            117906       290919     117908
1      118536            118536       308574     118539
2      117879            267952        19721     117880
3      118321            240983       290919     118322
4      119323            123932        19793     119325
```

Figure 1: Top five rows in the dataset

The dataset has ten attributes. All attributes are categorial. One attribute called `RESOURCE` holds the ID of the resource for which the access has been granted or denied. There are 7518 different resources in the dataset. The target attribute is called `ACTION`. The other eight columns provide role information for an employee[1]:

- `MGR_ID` - ID of the manager of employee (4243 different values)

- `ROLE_ROLLUP_1` - Role ID of employee (128 different values)

- `ROLE_ROLLUP_2` - Second role ID of employee (177 different values)

- `ROLE_DEPTNAME` - Role department description (449 different values)

- `ROLE_TITLE` - Role business title (343 different values)

- `ROLE_FAMILY` - Role family description (67 different values)

- `ROLE_FAMILY_DESC` - Extended role family description (2358 different values)

- `ROLE_CODE` - Company role code; this code is unique to each role (343 different values)

---

[1]`https://www.kaggle.com/c/amazon-employee-access-challenge/data`

Because the eight attributes that describe the role of an employee and the resource attribute are categorial, they will be vectorized. This is achieved by using one-hot encoding. For example the ROLE_FAMILY-attribute has 67 different categories. Each row in the dataset that has been one-hot encoded will contain one entry for each category. The entry will be 1 (hot) only for the entry corresponding to the current role family and 0 (cold) for the 66 other role families. This example shows that the number of input attributes will increase tremendously.

2.2 Exploratory Visualization

Figure 2 shows that the ACTION-attribute is highly unbalanced. In fact more than 94% of the rows have an ACTION-attribute of 1 (access granted) whereas only roughly 6% have a 0 (access denied). The accuracy metric is therefore inadequate for this dataset because even a dumb model that always predicts 1 would have a very high accuracy.
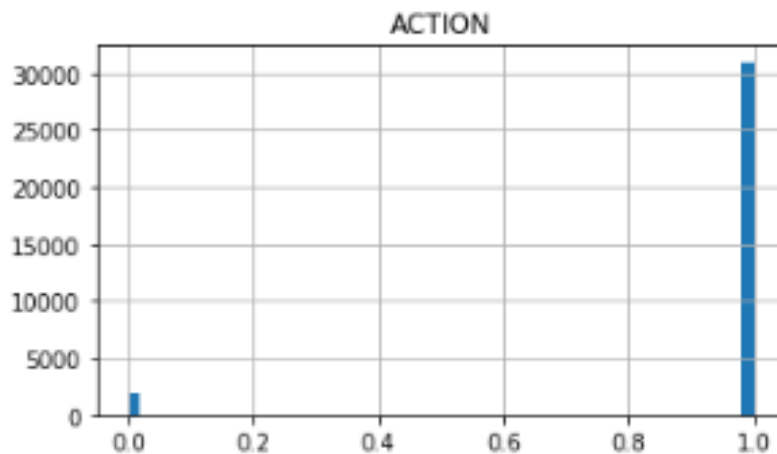

Figure 2: Histogram for target attribute

2.3 Algorithms and Techniques

First of all it is important to put aside a test set. This set will not be touched until the very end when all models have been trained and optimized. To generate the test set stratified shuffle split[2] will be used which preserves the percentage of samples for each class. A preprocessing pipeline will be created, that selects role attributes and applies one-hot encoding to them. This step is important because, as mentioned before, the attributes are categorial and two values that are close to each other are not more similar than two values with a larger distance. Different binary classification models with standard parameters will be created. The classifiers trained in this project are Logistic Regression, Decision Tree, SVM, Random Forest, AdaBoost and XGBoost. They will be initialized with a fixed random seed in order to make the results reproducable. There performance will be measured using scikit learn's cross validation score[3] function in order to circumvent overfitting. The best models will be tweaked by trying different sets of parameters. The goal is to furhter increase their performance. The better set of hyperparameters for the different models will be determined by doing a grid search[4]. Grid search will evaluate all combinations of a specified amount of parameters and values. This is far more convenient than the tedious task of trying out a bunch of hyperparameters manually. Finally the previously generated test set will be used to determine whether the final model generalizes well.

---

[2]http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html
[3]http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html
[4]http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

2.4 Benchmark

An out of the box logistic regression model will be used as the benchmark for this project, because the model is fast, simple to implement and to interpret and should give far better results than random guessing for the problem at hand.

Because the problems stems from a Kaggle competition, as a secondary benchmark, the result of the final solution will be compared to the result of the solution of the team that won the competition. The submissions to the competition were judged on the *area under the ROC curve (auc)* metric. Therefore this metric will be used to compare the results. The winning team got an *auc* value of 0.92360, which is an excellent result.

3 Methodology

3.1 Data Preprocessing

Compared to the samples with a positive class (access granted) the dataset contains very few samples with a negative class (access denied). The first preprocessing step is therefore to put aside a test set that preserves the percentage of samples for each class. As mentioned before this is achieved by using a stratified shuffle split. If a standard random split had been used to split the dataset into a training and testing set, it would be possible that for example the training set would not contain a single negative sample which would have certainly a negative impact on the performance of the classifiers. The second preprocessing step is to one-hot encode the predictive attributes. This is done by using sklearn's *OneHotEncoder*[5]. This transforms all the categorial attributes to binary attributes and fixes the issue that the models would assume that two nearby values of a categorial attribute are more similar than two distant values.

3.2 Implementation

The first step is to make some imports and to read the dataset:

```
import numpy as np
import pandas as pd
random_state = 42
df = pd.read_csv('../data/train.csv')
```

The second step is to create the stratified training and test sets with a test size of 25%:

```
from sklearn.model_selection import StratifiedShuffleSplit
sss = StratifiedShuffleSplit(n_splits=1, test_size=0.25, random_state=random_state)
for train_index, test_index in sss.split(df, df['ACTION']):
    train_set, test_set = df.loc[train_index], df.loc[test_index]
```

The third step is to extract the labels and the attributes for the training set:

```
access = train_set.drop('ACTION', axis=1)
access_labels = train_set['ACTION'].copy()
```

The fourth step uses a `DataFrameSelector`:

```
# DataFrameSelector class, taken from "Hands-On Machine Learning with Scikit-Learn &
# Tensorflow by Aurlion Gron (O'Reilly). Copyright 2017 Aurlion Gron, 978-1-491-96229-9,
    Page 41
from sklearn.base import BaseEstimator, TransformerMixin
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
```

---

[5]http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

```
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

Using the `DataFrameSelector` a pipeline is built that one-hot encodes the attributes of the training set:

```python
# get attributes
attributes = access.columns.values.tolist()

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder

# one-hot encode the categorial attributes
pipeline = Pipeline([
    ('selector', DataFrameSelector(attributes)),
    ('encoder', OneHotEncoder())
])
access_1hot = pipeline.fit_transform(access)
```

The sixth step is to create a function that takes a trained model and an integer number of folds as an input and displays the cross validation scores based on the area under the ROC curve metric (`auc`) for the *n*-folds, their mean and their standard deviation:

```python
def display_auc_scores(model, folds=10):
    scores = cross_val_score(model, access_1hot, access_labels, scoring='roc_auc', cv=folds)
    print('Scores: ', scores)
    print('Mean: ', scores.mean())
    print('Std: ', scores.std())
```

The final step is to actually create and train the different models. The following listing shows the creation and training of a logistic regression classifier:

```python
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(random_state=random_state)
log_reg.fit(access_1hot, access_labels)
display_auc_scores(log_reg, folds=10)
```

3.3 Refinement

The initial results for the **logistic regression** benchmark model on the training-set were the following:

```
('Scores: ', array([ 0.86783489, 0.79915939, 0.79383764, 0.87851276, 0.85076963,
    0.86988866, 0.86555532, 0.85809023, 0.84325891, 0.86433091]))
('Mean: ', 0.84912383267572067)
('Std: ', 0.027958452541170693)
```

The initial results for all models ordered by descending mean *auc* score on the training-set were the following:

Model	Mean auc	Standard Deviation
Logistic Regression	0.84912	0.02796
Random Forest	0.82307	0.02451
XGBoost	0.75264	0.02680
SVM	0.75006	0.03236
AdaBoost	0.74251	0.02921
Decision Tree	0.70193	0.02216

To refine the initial results the best three models were chosen to being tweaked with the exception of the logistic regression model that was used as the benchmark. For each hyperparameter that has been modified the default value was the basis. Values that are near this default value were tested as well as some values that are farther away. In the end the best parameters according to the *auc* scores obtained by doing a grid search cross validation were chosen.

It was first tried to improve the random forest model by discovering a good value for the `n_estimators`-parameter corresponding to the number of trees in the forest. The default value is 10 and values of $5, 10, 20, 30, 40$ and 50 have been tested. The best number of trees was discovered to be 50. With this parameter-change the model's mean *auc* score on the training-set increased to 0.85387. Secondly the number of minimum required samples to split an internal node has been examined. The default value is 2 and values of $2, 5$ and 10 have been tested. The best value has discovered to be 5. With this additional change the mean *auc* score on the training-set increased to 0.85687. By the setting the max depth to 500 the mean *auc* score got up to 0.85706. Changing the `min_samples_leaf` parameter from the default value of 1 to $2, 3, 4, 10, 50, 100$ brought no improvement. Finally the maximum number of features to consider when looking for the best split has been tried to improve. The values *sqrt*, $5, 10, 20$ and 50 have been tried. Among those values, 10 has been discovered to be the best and with that the mean auc score on the training-set increased to 0.86371.

Secondly the SVM model has been improved by doing a grid-search with the values $[0.01, 0.1, 1, 10]$ for the parameter `gamma` and $[0.1, 1, 10, 100]$ for the parameter `C`. The best values were 1 for `gamma` and 10 for `C`. This search took several hours. Even the score calculation of the refined SVM model took a long time. Because of the extremely high computational costs, other parameter settings for the SVM model were not tried. The refined SVM got a mean *auc* score on the training-set of 0.85768.

The last model being refined was the XGBoost model. The first grid-search consisted of the parameters `max_depth` and `min_child_width` for which the values $[5, 6, 7, 8, 9, 10]$ and $[1, 2, 3, 4, 5]$ were used respectively. Out of these parameter-sets the best values were 10 for the max depth and 1 for the other parameter. With this parameter-setting the mean *auc* score on the training-set inccreased to 0.81301. Trying other values such as $[6, 7, 8, 9, 10, 50, 100$ for the `min_child_width` parameter did not lead to better results. For the `gamma` parameter the value 0.5 was found to give the best results on the training set. The default value is 0.0 and values of $0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6$ and 1.0 have been tried. After this parameter twist has been applied the score got up to 0.81489. The best learning rate was found to be 0.4. With this little change the score increased to 0.82793. Next, I've tried to find a good value for the number of boosted trees to fit. After trying many different values I discovered that the best value seems to be 170. Using this value for the parameter led to a mean score of 0.83418. After that I've tried different values for the subsample ratio of the training instance and the subsample ratio of columns when each each tree is constructed. This are the parameters `subsample` and `colsample_bytree`. The best value for the `subsample` parameter seems to be the default value of 1.0. For the `colsample_bytree` parameter I found a value of 0.9 to be better. With this adjustment the score increased to 0.83470. I was not able to find better values for the regularization parameters `reg_alpha` and `reg_lambda` than the defaults. The same holds true for the parameter that is used for balancing of positive and negative weights. After that I again tried some values for the number of boosted trees and discovered that much higher numbers led to somewhat improved scores on the training set. With a value of 1000 the score increased to 0.83897. But the change comes with a cost the computation time increased perceptibly. Therefore higher numbers were not tested for this parameter.

4   Results

4.1   Model Evaluation and Validation

To evaluate the model the previously set aside test-set was used.

4.2 Justification

On the test-set the benchmark logistic regression model got an *auc* score of 0.56946. The three models that have been used for further examination were all found slightly stronger than the benchmark model. The final random forest model had a score of 0.57810, the final SVM model a score of 0.57243 and the final XGBoost model got a score of 0.65055. Figure 3 shows the ROC curves of the final model compared to the benchmark.
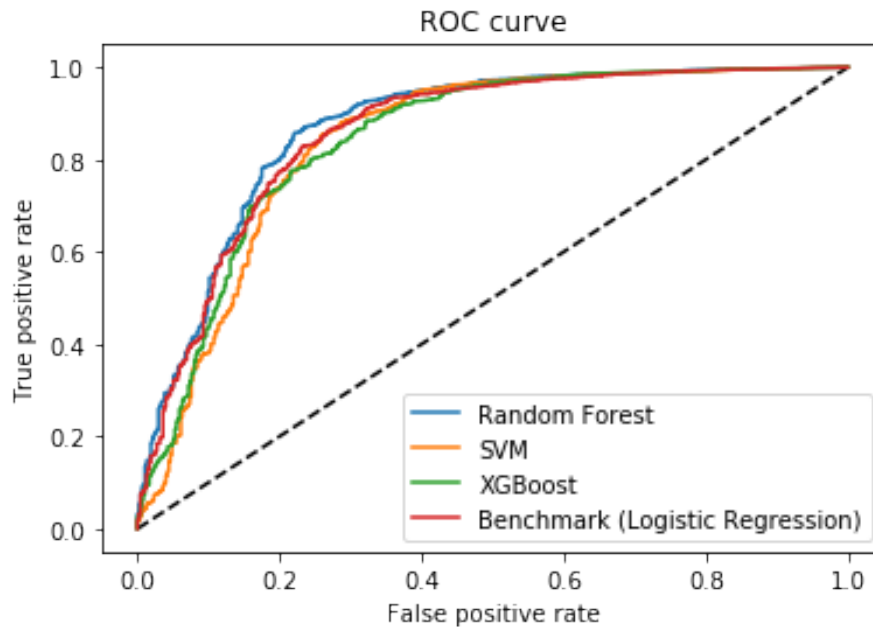

Figure 3: ROC curves for different models

So the XGBoost model is the one that can handle unseen data better than the other models. The overall score is not that good. As mentioned before the winning team of the Kaggle competition managed to get a score of 0.92360. Because of that I would argue that the final solution is not significant enough to have solved the problem.

5 Conclusion

5.1 Free-Form Visualization

5.2 Reflection

5.3 Improvement

Probably the number of trees in the random forest model could be increased to improve the performance. But this also increases the computation time and has therefore be neglected.

References

[1] L1 logistic regression as a feature selection step for training stable classification trees for the prediction of severity criteria in imported malaria

Luca Talenti, Margaux Luck, Anastasia Yartseva, Nicolas Argy, Sandrine Houz, Cecilia Damon

arXiv:1511.06663 [cs.LG]

[2] Amazon.com – Employee Access Challenge

Predict an employee's access needs, given his/her job role.

https://www.kaggle.com/c/amazon-employee-access-challenge,

[3] Receiver operating characteristic

https://en.wikipedia.org/wiki/Receiver_operating_characteristic