dontGetKicked_JA

August 6, 2017

1 "Don't Get Kicked": a classification practice

Jamal Alikhani
jamal.alikhani@gmail.com
July 2017

Dealerships buy used cars in large scale to get benefit after reselling them to new customers. However, they sometimes mistakenly buy cars with major issues that prevent them from reselling these cars to customers, which is slanged as "kicked" cars.

The goal of this practice is to develop a predictive model that helps dealerships to detect "kicked" cars before purchasing them. For this reason, data are downloaded from kaggle.com competition called "Don't get kicked" as same as this report's title.

The dataset is uploaded to the Python3 environment, and after preprocessing and data cleaning phases, the scikit-learn tool has been used to provide three classification models based on logistic regression, bagging ensemble, and boosting ensemble approaches. The Python codes plus a short description is documented as below.

```
In [2]: import pandas as pd
        import numpy as np

        # classification metrics
        from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

        # Selected classifiers
        from sklearn.linear_model import LogisticRegression
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.ensemble import AdaBoostClassifier

        # Cross validation models
        from sklearn.preprocessing import LabelEncoder, normalize, OneHotEncoder
        from sklearn.pipeline import Pipeline
        from sklearn.model_selection import GridSearchCV, train_test_split, KFold, cross_val_sco
```

2 Data Preparation

2.1 Loading the training dataset

```
In [3]: train = pd.read_csv("training.csv")
        print("Origional size of the training set: ", train.shape)
```

```
Origional size of the training set:  (72983, 34)
```

```
In [4]: pd.set_option('display.max_columns', 50)
        print("A preview of training dataset prior data processing:")
        train.head()
```

```
A preview of training dataset prior data processing:
```

```
Out[4]:    RefId  IsBadBuy  PurchDate Auction  VehYear  VehicleAge   Make  \
        0      1         0  12/7/2009   ADESA     2006           3  MAZDA
        1      2         0  12/7/2009   ADESA     2004           5  DODGE
        2      3         0  12/7/2009   ADESA     2005           4  DODGE
        3      4         0  12/7/2009   ADESA     2004           5  DODGE
        4      5         0  12/7/2009   ADESA     2005           4   FORD

                        Model Trim           SubModel   Color Transmission  \
        0              MAZDA3    i        4D SEDAN I     RED         AUTO
        1  1500 RAM PICKUP 2WD   ST  QUAD CAB 4.7L SLT   WHITE         AUTO
        2          STRATUS V6  SXT    4D SEDAN SXT FFV  MAROON         AUTO
        3                NEON  SXT          4D SEDAN  SILVER         AUTO
        4               FOCUS  ZX3       2D COUPE ZX3  SILVER       MANUAL

           WheelTypeID WheelType  VehOdo  Nationality         Size  \
        0            1     Alloy   89046  OTHER ASIAN       MEDIUM
        1            1     Alloy   93593     AMERICAN  LARGE TRUCK
        2            2    Covers   73807     AMERICAN       MEDIUM
        3            1     Alloy   65617     AMERICAN      COMPACT
        4            2    Covers   69367     AMERICAN      COMPACT

          TopThreeAmericanName  MMRAcquisitionAuctionAveragePrice  \
        0                OTHER                               8155
        1             CHRYSLER                               6854
        2             CHRYSLER                               3202
        3             CHRYSLER                               1893
        4                 FORD                               3913

           MMRAcquisitionAuctionCleanPrice  MMRAcquisitionRetailAveragePrice  \
        0                             9829                             11636
        1                             8383                             10897
        2                             4760                              6943
        3                             2675                              4658
        4                             5054                              7723

           MMRAcquisitonRetailCleanPrice  MMRCurrentAuctionAveragePrice  \
        0                          13600                           7451
        1                          12572                           7456
```

```
2                        8457                          4035
3                        5690                          1844
4                        8707                          3247

   MMRCurrentAuctionCleanPrice  MMRCurrentRetailAveragePrice  \
0                         8552                         11597
1                         9222                         11374
2                         5557                          7146
3                         2646                          4375
4                         4384                          6739

   MMRCurrentRetailCleanPrice PRIMEUNIT AUCGUART  BYRNO  VNZIP1 VNST  \
0                       12409       NaN      NaN  21973   33619   FL
1                       12791       NaN      NaN  19638   33619   FL
2                        8702       NaN      NaN  19638   33619   FL
3                        5518       NaN      NaN  19638   33619   FL
4                        7911       NaN      NaN  19638   33619   FL

   VehBCost  IsOnlineSale  WarrantyCost
0      7100             0          1113
1      7600             0          1053
2      4900             0          1389
3      4100             0           630
4      4000             0          1020
```

Dropping redundant and none-informative attributes: The none informative attributes like IDs and dates, redundant attributes like zipcode and "VehYear" while "VehicleAge" is provided with the same information, and attributes with none-atomic contents like "Model" and "SubModel" are dropped.

It should be pointed out that attributes with none-atomic members can be kept and their information can be retrieved by natural language processing techniques (like *nltk*). However, for this practice, they are just simply dropped out.

```
In [5]: # dropping none informative (redundant) attributes
        train.drop(['RefId', 'PurchDate', 'VehYear', 'WheelType', 'VNZIP1','SubModel', 'Model'],
```

Checking the number of missing data in each attribute:

```
In [6]: print('Number of NA/NaNs in each attribute:')
        train.isnull().sum(axis=0)
```

```
Number of NA/NaNs in each attribute:
```

```
Out[6]: IsBadBuy                            0
        Auction                             0
        VehicleAge                          0
        Make                                0
```

```
Trim                                  2360
Color                                    8
Transmission                             9
WheelTypeID                           3169
VehOdo                                   0
Nationality                              5
Size                                     5
TopThreeAmericanName                     5
MMRAcquisitionAuctionAveragePrice       18
MMRAcquisitionAuctionCleanPrice         18
MMRAcquisitionRetailAveragePrice        18
MMRAcquisitonRetailCleanPrice           18
MMRCurrentAuctionAveragePrice          315
MMRCurrentAuctionCleanPrice            315
MMRCurrentRetailAveragePrice           315
MMRCurrentRetailCleanPrice             315
PRIMEUNIT                            69564
AUCGUART                             69564
BYRNO                                    0
VNST                                     0
VehBCost                                 0
IsOnlineSale                             0
WarrantyCost                             0
dtype: int64
```

Dropping the attributes with high number of missing values:

```
In [7]: train.drop(['PRIMEUNIT', 'AUCGUART'], axis=1, inplace =True)
```

The missing cells of 'WheelTypeID' and 'Trim' attributes are filled with some out-of-range values to keep around 3,200 rows. The classification algorithm can distinguish this out-of-range values in their model. For the rest of the missing value, the rows associated with each missing value is droppd.

```
In [8]: train['WheelTypeID'].fillna(-99, axis=0, inplace=True)
        train['Trim'].fillna('?', axis=0, inplace=True)
        train.dropna(axis=0, inplace=True)
        print("Size of training set after trimming: ", train.shape)

Size of training set after trimming:  (72658, 25)
```

2.1.1 One-hot encoding of categorical data to dymmy features:

```
In [14]: dt = train['Auction'].dtypes
         for col in train.columns:
             if str(train[col].dtypes) == 'object':
                 train = pd.get_dummies(train, columns=[col])
```

In [15]:
```python
print("A preview of training dataset after data processing:")
train.head()
```

```
A preview of training dataset after data processing:
```

Out[15]:
```
   IsBadBuy  VehicleAge  WheelTypeID  VehOdo  \
0         0           3            1   89046
1         0           5            1   93593
2         0           4            2   73807
3         0           5            1   65617
4         0           4            2   69367

   MMRAcquisitionAuctionAveragePrice  MMRAcquisitionAuctionCleanPrice  \
0                               8155                             9829
1                               6854                             8383
2                               3202                             4760
3                               1893                             2675
4                               3913                             5054

   MMRAcquisitionRetailAveragePrice  MMRAcquisitonRetailCleanPrice  \
0                             11636                          13600
1                             10897                          12572
2                              6943                           8457
3                              4658                           5690
4                              7723                           8707

   MMRCurrentAuctionAveragePrice  MMRCurrentAuctionCleanPrice  \
0                           7451                         8552
1                           7456                         9222
2                           4035                         5557
3                           1844                         2646
4                           3247                         4384

   MMRCurrentRetailAveragePrice  MMRCurrentRetailCleanPrice  BYRNO  VehBCost  \
0                         11597                       12409  21973      7100
1                         11374                       12791  19638      7600
2                          7146                        8702  19638      4900
3                          4375                        5518  19638      4100
4                          6739                        7911  19638      4000

   IsOnlineSale  WarrantyCost  Auction_ADESA  Auction_MANHEIM  Auction_OTHER  \
0             0          1113              1                0              0
1             0          1053              1                0              0
2             0          1389              1                0              0
3             0           630              1                0              0
4             0          1020              1                0              0
```

```
   Make_ACURA  Make_BUICK  Make_CADILLAC  Make_CHEVROLET  Make_CHRYSLER  \
0           0           0              0               0              0
1           0           0              0               0              0
2           0           0              0               0              0
3           0           0              0               0              0
4           0           0              0               0              0

   Make_DODGE   ...     VNST_LA  VNST_MA  VNST_MD  VNST_MI  VNST_MN  VNST_MO  \
0           0   ...           0        0        0        0        0        0
1           1   ...           0        0        0        0        0        0
2           1   ...           0        0        0        0        0        0
3           1   ...           0        0        0        0        0        0
4           0   ...           0        0        0        0        0        0

   VNST_MS  VNST_NC  VNST_NE  VNST_NH  VNST_NJ  VNST_NM  VNST_NV  VNST_NY  \
0        0        0        0        0        0        0        0        0
1        0        0        0        0        0        0        0        0
2        0        0        0        0        0        0        0        0
3        0        0        0        0        0        0        0        0
4        0        0        0        0        0        0        0        0

   VNST_OH  VNST_OK  VNST_OR  VNST_PA  VNST_SC  VNST_TN  VNST_TX  VNST_UT  \
0        0        0        0        0        0        0        0        0
1        0        0        0        0        0        0        0        0
2        0        0        0        0        0        0        0        0
3        0        0        0        0        0        0        0        0
4        0        0        0        0        0        0        0        0

   VNST_VA  VNST_WA  VNST_WV
0        0        0        0
1        0        0        0
2        0        0        0
3        0        0        0
4        0        0        0

[5 rows x 263 columns]
```

2.1.2 Splitting label attribute from training set

In [16]:
```
train = np.array(train,dtype='float')
X = train[:,1:]
y = train[:,0]
seed = 7
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=
```

3 Classification

In this practice, we are facing with an unbalanced labeled dataset where class "0" are 87.7% and class "1" (kicks) are just 12.3%! This makes the classification training phase tricky because the accuracy of random selection is already a high value (87.7%). Therefore, accuracy is not a proper metric for this case to be evaluated.

The important part of the model prediction is to reduce the number of kicked cars. Therefore, we have to increase the rate of true negative (TN) while decreasing the false negative (FN) rates. By looking at *precision*, *recall*, *F1-score*, and *Area under the ROC curve (ROC-AUC)* we can get a better insight into the classification performance.

```
In [17]: print("Ratio of class 1 to class 0 in the training set:")
         np.round(np.mean(y), 3)
```

```
Ratio of class 1 to class 0 in the training set:
```

```
Out[17]: 0.123
```

3.0.1 *k*-fold cross validation model for classification training score

```
In [18]: kfold = KFold(n_splits=10, shuffle=True, random_state=seed)
```

3.1 Logistic Regression

To overrcome the unbalanced lables, the "balanced" mode is activated for equally weighting the binary classes in the calculation of the loss function. The balanced mode automatically adjusts weights inversely proportional to class frequencies in the input data as:
`n_samples / (n_classes * np.bincount(y))`

```
In [19]: clf = LogisticRegression(class_weight='balanced')

         scoring = 'f1'
         results = cross_val_score(clf, X, y, cv=kfold, scoring=scoring, n_jobs=-1)
         print("Logistic Regression,",scoring, ": {0:.3}, (std: {1:.3})".format(results.mean(),
```

```
Logistic Regression, f1 : 0.366, (std: 0.00957)
```

```
In [20]: scoring = 'roc_auc'
         results = cross_val_score(clf, X, y, cv=kfold, scoring=scoring, n_jobs=-1)
         print("Logistic Regression,",scoring, ": {0:.3}, (std: {1:.3})".format(results.mean(),
```

```
Logistic Regression, roc_auc : 0.755, (std: 0.00815)
```

```
In [21]: clf.fit(X_train, y_train)
         y_pred = clf.predict(X_test)

         conf_mat = confusion_matrix(y_test, y_pred)
         print("Logistic Regression, confusion matrix: ")
         print(conf_mat)
```

```
Logistic Regression, confusion matrix:
[[16068  4962]
 [ 1167  1781]]
```

```
In [22]: acc = accuracy_score(y_test, y_pred)
         print("Logistic Regression, accuracy: {0:.1%}".format(acc))
```

```
Logistic Regression, accuracy: 74.4%
```

```
In [23]: print("Logistic Regression, summary report: ")
         print(classification_report(y_test, y_pred))
```

```
Logistic Regression, summary report:
             precision    recall  f1-score   support

        0.0       0.93      0.76      0.84     21030
        1.0       0.26      0.60      0.37      2948

avg / total       0.85      0.74      0.78     23978
```

f1-score and ROC-AUC metrics show reasonably good results for the first try, regardless of seemingly low accuracy of 75.5%! It is already mentioned that accuracy is not a good measure for this case. Values in the confusion matrix show that model was able to correctly predict the 58% of cars with major issues.

3.2 Random Forest Classifier

Random forest classifier uses bagging technique to construct many parallel decision trees over bootstrapped resampling of the training dataset to reduce the variance. Presence of several categorial attributes (like "Transmission" and "WheelTypeID") makes it suitable to apply decision tree as the base classifier.

To overcome the overfitting in the decision tree, a kfold cross validation technique over a grid search is applied to find the optimum depth of the trees. It is shown that 7 is the optimum depth.

```
In [24]: # Random Forest Classifier
         n_depth = [2, 5, 7, 10, 15, 20]

         pipeline_estimator = Pipeline([('clf', RandomForestClassifier(class_weight='balanced'))

         params = [{'clf__max_depth': n_depth}]

         grid = GridSearchCV(estimator=pipeline_estimator, param_grid=params, scoring='f1', cv=3

         grid.fit(X, y)
         mean_scores = np.array(grid.cv_results_['mean_test_score'])
```

```
reults_df = pd.DataFrame()
reults_df['n_depth'] = n_depth
reults_df['f1_score'] = mean_scores
print("Random Forest Classifier, Grid Search result:")
print(reults_df)
```

```
Random Forest Classifier, Grid Search result:
   n_depth  f1_score
0        2  0.287530
1        5  0.310161
2        7  0.332633
3       10  0.339482
4       15  0.348165
5       20  0.329425
```

In [25]:
```
opt_depth = n_depth[np.argmax(mean_scores)]
clf = RandomForestClassifier(max_depth=opt_depth, n_estimators=100, class_weight='balan

scoring = 'roc_auc'
results = cross_val_score(clf, X, y, cv=kfold, scoring=scoring, n_jobs=-1)
print("Random Forest Classifier,",scoring, ": {0:.3}, (std: {1:.3})".format(results.mea
```

```
Random Forest Classifier, roc_auc : 0.761, (std: 0.00557)
```

In [26]:
```
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

conf_mat = confusion_matrix(y_test, y_pred)
print("Random Forest Classifier, confusion matrix: ")
print(conf_mat)
```

```
Random Forest Classifier, confusion matrix:
[[18711  2319]
 [ 1664  1284]]
```

In [27]:
```
acc = accuracy_score(y_test, y_pred)
print("Random Forest Classifier, accuracy: {0:.1%}".format(acc))
```

```
Random Forest Classifier, accuracy: 83.4%
```

In [28]:
```
print("Random Forest Classifier, summary report: ")
print(classification_report(y_test, y_pred))
```

```
Random Forest Classifier, summary report:
             precision    recall  f1-score   support
```

```
        0.0       0.92      0.89      0.90     21030
        1.0       0.36      0.44      0.39      2948

avg / total       0.85      0.83      0.84     23978
```

Both the ROC-AUC factor (0.761) and overal f1-score (0.82) are slightly better than the results obtained from logistic regression. However, just by looking at the TN rate (recall) from the confusion matrix, logistic regression finds a higher value of 58% compared to the 51% obtained by the Random Forest.

3.3 AdaBoost Classifier

AdaBoost is an adaptive boosting ensemble that collects many weak classifiers into a strong classifier. Similar to the Random Forest, a decision tree is again used as the base classifier.

Since in the scikit-learn package this algorithm does not support the weighted classes, the majority class is "down-sampled" to get a size equal to the minority class. Afterward, the balanced subsamples of the training data is fed into the model training phase of the AdaBoost.

```
In [29]: Xt_c1 = X_train[y_train==1]
         yt_c1 = y_train[y_train==1]

         Xt_c0 = X_train[y_train==0]
         yt_c0 = y_train[y_train==0]
         Xt_c0 = Xt_c0[:Xt_c1.shape[0],:]
         yt_c0 = yt_c0[:Xt_c1.shape[0]]

         X_train_balanced = np.concatenate((Xt_c0, Xt_c1), axis=0)
         y_train_balanced = np.concatenate((yt_c0, yt_c1), axis=0)
```

```
In [30]: clf = AdaBoostClassifier(n_estimators=100, random_state=seed)

         scoring = 'f1'
         results = cross_val_score(clf, X_train_balanced, y_train_balanced, cv=kfold, scoring=sc
         print("AdaBoost Classifier,",scoring, ": {0:.3}, (std: {1:.3})".format(results.mean(),
```

```
AdaBoost Classifier, f1 : 0.66, (std: 0.0171)
```

```
In [31]: scoring = 'roc_auc'
         results = cross_val_score(clf, X_train_balanced, y_train_balanced, cv=kfold, scoring=sc
         print("AdaBoost Classifier,",scoring, ": {0:.3}, (std: {1:.3})".format(results.mean(),
```

```
AdaBoost Classifier, roc_auc : 0.754, (std: 0.0118)
```

```
In [32]: clf.fit(X_train_balanced, y_train_balanced)
         y_pred = clf.predict(X_test)

         conf_mat = confusion_matrix(y_test, y_pred)
         print("AdaBoost Classifier, confusion matrix: ")
         print(conf_mat)
```

```
AdaBoost Classifier, confusion matrix:
[[15340  5690]
 [ 1115  1833]]
```

```
In [33]: acc = accuracy_score(y_test, y_pred)
         print("AdaBoost Classifier, accuracy: {0:.1%}".format(acc))
```

```
AdaBoost Classifier, accuracy: 71.6%
```

```
In [34]: print("AdaBoost Classifier, summary report: ")
         print(classification_report(y_test, y_pred))
```

```
AdaBoost Classifier, summary report:
             precision    recall  f1-score   support

        0.0       0.93      0.73      0.82     21030
        1.0       0.24      0.62      0.35      2948

avg / total       0.85      0.72      0.76     23978
```

The AdaBoost performs better in recall percentage compare to other two clasiifiers. The f1-score and the ROC-AUC are just slightly lower than random forest.

4 Conclusion

Three classification models are applied to predict the cars with major issues referred to as kicked cars:

classifier	recall over kicked cars	overall f1-score
Logistic Regression	0.60	0.78
Random Forest	0.44	0.84
AdaBoost	0.62	0.76

It is concluded that the accuracy is a poor metric to evaluate the model performance due to the heavily unbalanced labels in the binary classes. True negative classification rate or "Recall" is considered to be a target model performance. The TN prevents car dealerships from purchasing a kicked. On the other hand, there is a trade-off between the recall and the precision, as when the rate of false negative (FN) increases, the company loses the opportunity to purchase a good car that could potentially benefit the company. The financial objective function can be constructed as:

```
total_profit = (TP-FN)*benefit + (FP-TN)*Loss
```

where *benefit* and *loss* are the positive and negative benefit from each good and bad car, respectively. By maximizing this objective function, the optimum trade-off between the FN and TN rates can be obtained and then a best predictive model can be selected and trained.

Other classifications can also be tested and may perform better in this case.