1 Introduction

An auto dealership often purchases a used car at an auto auction. This used car might have serious issues that cannot sell to customers. The auto community calls these unfortunate purchases "kicks." The purpose of this project is to build a model to predict if the car purchased at the Auction is a Kick (bad buy). All project data come from the Kaggle. I propose to use logistic regression as the main algorithm to create the classifier. After training and tuning hyperparameter of the classifier, the area under the curve (AUC) of my classifier is 0.73. The F1 score of my classifier is 0.35. The accuracy of the prediction on the testing dataset is 95%. I also provide a well-documented IPython file on Github to explain how to step-by-step analyze this project.

In this project, I will follow steps from Hands-On Machine Learning with Scikit-Learn and TensorFlow. This book provides an excellent strategy how to do an end-to-end machine learning project. I highly recommend this strategy for the machine learning project. I summary general processes for a machine learning project in this report, the process can split into seven steps: **understand the problem, get the data, data exploration, data preprocessing, model training, and model tuning and evaluations.**

2 Understand the Problem

The project target is to predict if the car purchased at the Auction is a Kick (bad buy).Therefore, I frame this project as a supervised classification problem. Furthermore, this project is a binary classification problem because it predicts help a dealership to identify the purchases is a bad buy or a good buy.

For the binary classification problem, there are three standard methods to evaluate the performance of the model. First, I evaluate the performance of a classifier using F1 score, because the F1 score is the combination of precision and recall, which is the harmonic mean of precision and recall. Also, I evaluate the performance of classifier use AUC of receiver operating characteristic (ROC) curve, because the ROC curve is another common tool used with binary classifiers. Finally, I evaluate the prediction accuracy of the testing dataset.

To sum up, after framing the problem (make an assumption) and select methods to evaluate the model, I can start to get data and to look data information.

3 Get the data

All data come from Kaggle, I download data and use Pandas to load data. However, before directly using Pandas, I write a small class with some loading function to load data (the codes are in IPython file).

Before splitting data into the training and test datasets, I need understand what exactly the data it is. First, I check original data and see the header of data in figure 1.

	RefId	IsBadBuy	PurchDate	Auction	VehYear	VehicleAge	Make	Model	Trim	SubModel	...	MMRCurrentRetailAveragePrice	MMRCurrentRetailClea
0	1	0	12/7/2009	ADESA	2006	3	MAZDA	MAZDA3	i	4D SEDAN I	...	11597.0	1
1	2	0	12/7/2009	ADESA	2004	5	DODGE	1500 RAM PICKUP 2WD	ST	QUAD CAB 4.7L SLT	...	11374.0	1
2	3	0	12/7/2009	ADESA	2005	4	DODGE	STRATUS V6	SXT	4D SEDAN SXT FFV	...	7146.0	
3	4	0	12/7/2009	ADESA	2004	5	DODGE	NEON	SXT	4D SEDAN	...	4375.0	
4	5	0	12/7/2009	ADESA	2005	4	FORD	FOCUS	ZX3	2D COUPE ZX3	...	6739.0	

5 rows × 34 columns

Figure 1 shows the top 5 rows and header of the data.

Here, we can see some feature of data, but it is not enough. Next, I want to know more data details in figure 2 such as total rows of data, numbers of features of data, the type of each feature of data.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 72983 entries, 0 to 72982
Data columns (total 34 columns):
RefId                                72983 non-null int64
IsBadBuy                             72983 non-null int64
PurchDate                            72983 non-null object
Auction                              72983 non-null object
VehYear                              72983 non-null int64
VehicleAge                           72983 non-null int64
Make                                 72983 non-null object
Model                                72983 non-null object
Trim                                 70623 non-null object
SubModel                             72975 non-null object
Color                                72975 non-null object
Transmission                         72974 non-null object
WheelTypeID                          69814 non-null float64
WheelType                            69809 non-null object
VehOdo                               72983 non-null int64
Nationality                          72978 non-null object
Size                                 72978 non-null object
TopThreeAmericanName                 72978 non-null object
MMRAcquisitionAuctionAveragePrice    72965 non-null float64
MMRAcquisitionAuctionCleanPrice      72965 non-null float64
MMRAcquisitionRetailAveragePrice     72965 non-null float64
MMRAcquisitonRetailCleanPrice        72965 non-null float64
MMRCurrentAuctionAveragePrice        72668 non-null float64
MMRCurrentAuctionCleanPrice          72668 non-null float64
MMRCurrentRetailAveragePrice         72668 non-null float64
MMRCurrentRetailCleanPrice           72668 non-null float64
PRIMEUNIT                            3419 non-null object
AUCGUART                             3419 non-null object
BYRNO                                72983 non-null int64
VNZIP1                               72983 non-null int64
VNST                                 72983 non-null object
VehBCost                             72983 non-null float64
IsOnlineSale                         72983 non-null int64
WarrantyCost                         72983 non-null int64
dtypes: float64(10), int64(9), object(15)
memory usage: 18.9+ MB
```

Figure 2 shows details information about this data.

From the output, I can see this data has total 34 features, 72983 rows of data, 19 numeric data, and 16 text data. Then, in figure 3, I know each text data feature, and majority label is "true" for the data.

```
CHEVROLET       17248
DODGE           12912
FORD            11305
CHRYSLER         8844
PONTIAC          4258
KIA              2484
SATURN           2163
NISSAN           2085
HYUNDAI          1811
JEEP             1644
SUZUKI           1328
TOYOTA           1144
MITSUBISHI       1030
MAZDA             979
MERCURY           913
BUICK             720
GMC               649
HONDA             497
OLDSMOBILE        243
ISUZU             134
VOLKSWAGEN        134
SCION             129
LINCOLN            97
INFINITI           42
VOLVO              37
ACURA              33
CADILLAC           33
LEXUS              31
SUBARU             28
MINI               24
PLYMOUTH            2
TOYOTA SCION        1
HUMMER              1
Name: Make, dtype: int64
```

```
0    64007
1     8976
Name: IsBadBuy, dtype: int64
```

Figure 3 shows each column's data distribution.

In figure 4, I want to know more statistical attributes like the mean of each numeric data or the standard deviation of each numeric data.

	RefId	IsBadBuy	VehYear	VehicleAge	WheelTypeID	VehOdo	MMRAcquisitionAuctionAveragePrice	MMRAcquisitionAuctionClea
count	72983.000000	72983.000000	72983.000000	72983.000000	69814.000000	72983.000000	72965.000000	72965.
mean	36511.428497	0.122988	2005.343052	4.176644	1.494299	71499.995917	6128.909217	7373.
std	21077.241302	0.328425	1.731252	1.712210	0.521290	14578.913128	2461.992768	2722.
min	1.000000	0.000000	2001.000000	0.000000	0.000000	4825.000000	0.000000	0.
25%	18257.500000	0.000000	2004.000000	3.000000	1.000000	61837.000000	4273.000000	5406.
50%	36514.000000	0.000000	2005.000000	4.000000	1.000000	73361.000000	6097.000000	7303.
75%	54764.500000	0.000000	2007.000000	5.000000	2.000000	82436.000000	7765.000000	9021.
max	73014.000000	1.000000	2010.000000	9.000000	3.000000	115717.000000	35722.000000	36859.

Figure 4 shows each column's data statistical attributes.

Sometimes, people may prefer to visualize the data distribution, for example, in figure 5. Now I write a class to plot data and save graphs into the target location. The MatPlotlib already have the plot function, but it is a good habit to write a function to combination plot and label, which can reuse later.

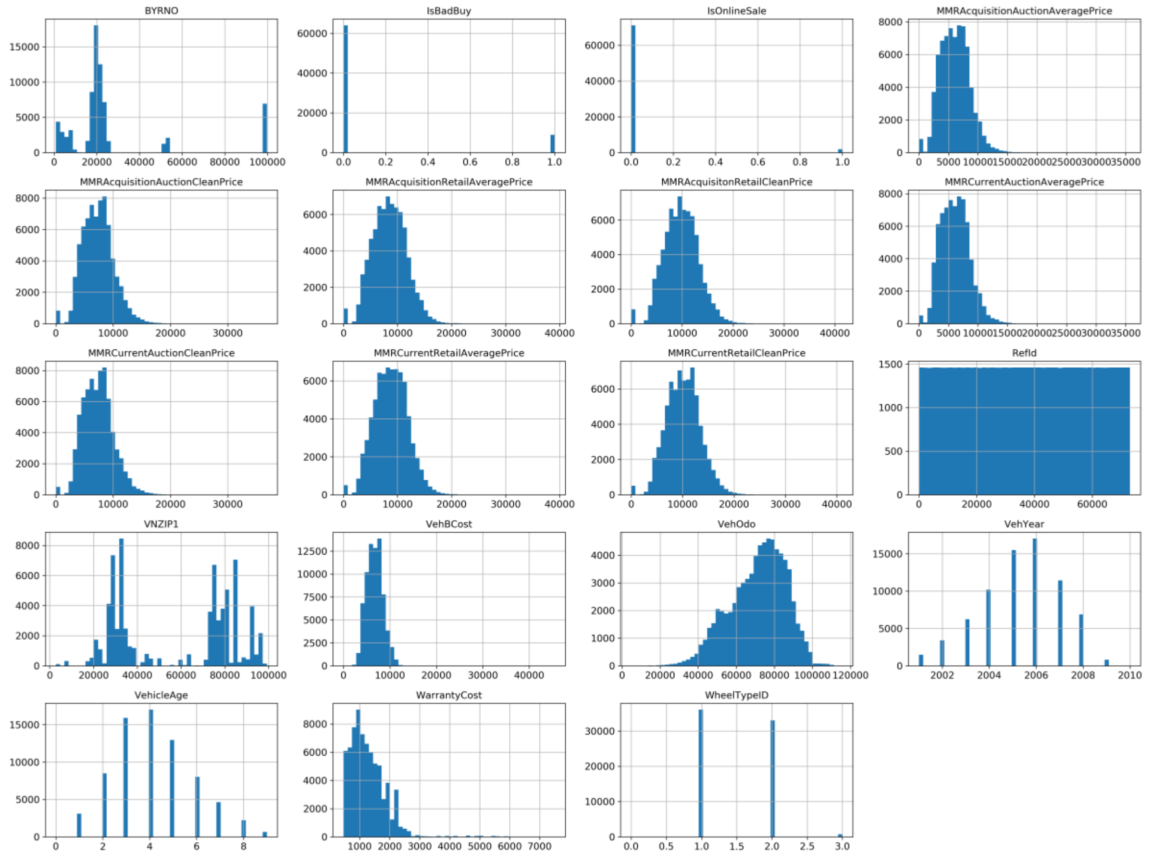

Figure 5 provides diagrams to display data distribution of all numeric columns.

From the “IsBadBuy” diagram, I can also see the majority label is “true,” which is good for us making classification on the test dataset. Also, I see the “VehYear” and “VehicleAge” diagrams have the normal distribution, which means that I can use those two features as the critical features to make a model (have more eigenvalue to make more accuracy model).

To summary, after I understand much information about the data, I can split the data into training and test dataset. However, I need not split datasets because Kaggle already provides training (60%) and test (40%) dataset. If the data need to split, the Scikit-Learn provide a few functions to split datasets into multiple subsets in various ways.)

4 Data Exploration

So far I have only taken a quick glance at the data to get a general understanding of the kind of data I am manipulating. Now, I need to go deep to check more about data. First, I need put the test dataset aside. Since the training dataset is not too large, I can efficiently compute the standard correlation coefficient (also called Pearson's r) between every pair of attributes, in Figure 6.

```
IsBadBuy                             1.000000
VehicleAge                           0.167164
VehOdo                               0.082560
WarrantyCost                         0.052319
RefId                                0.023973
VNZIP1                               0.005796
IsOnlineSale                        -0.003697
WheelTypeID                         -0.044620
BYRNO                               -0.061488
MMRAcquisitonRetailCleanPrice       -0.083665
MMRAcquisitionRetailAveragePrice    -0.087418
VehBCost                            -0.099911
MMRCurrentRetailCleanPrice          -0.100245
MMRAcquisitionAuctionCleanPrice     -0.102954
MMRCurrentRetailAveragePrice        -0.103914
MMRCurrentAuctionCleanPrice         -0.104020
MMRCurrentAuctionAveragePrice       -0.109112
MMRAcquisitionAuctionAveragePrice   -0.109252
VehYear                             -0.158886
Name: IsBadBuy, dtype: float64
```

Figure 6 shows standard correlation coefficient.

From the correlation, I see the “VehYear” and “VehicleAge” is the strongest relation with “IsBadBuy.”

One last thing I want to do before actually preparing the data for machine learning algorithms is to try out various attribute combinations. In my opinion, the more miles per year driver drive; the more likely driver will more cause a kick car. Thus, I can create new attribute “miles per year,” and I recheck the standard correlation coefficient in figure 7.

```
IsBadBuy                             1.000000
VehicleAge                           0.167164
VehOdo                               0.082560
WarrantyCost                         0.052319
RefId                                0.023973
VNZIP1                               0.005796
IsOnlineSale                        -0.003697
WheelTypeID                         -0.044620
BYRNO                               -0.061488
MMRAcquisitonRetailCleanPrice       -0.083665
MMRAcquisitionRetailAveragePrice    -0.087418
VehBCost                            -0.099911
MMRCurrentRetailCleanPrice          -0.100245
MMRAcquisitionAuctionCleanPrice     -0.102954
MMRCurrentRetailAveragePrice        -0.103914
MMRCurrentAuctionCleanPrice         -0.104020
MMRCurrentAuctionAveragePrice       -0.109112
MMRAcquisitionAuctionAveragePrice   -0.109252
miles_per_year                      -0.112894
VehYear                             -0.158886
Name: IsBadBuy, dtype: float64
```

Figure 7 shows standard correlation coefficient after adding a custom attribute.

Now, it’s time to prepare the data for my machine learning algorithms.

5 Data Preprocessing

The best way to prepare data is to use Scikit-Learn’s Pipeline to do it. However, before I create the full pipeline, I need figure out how to create transformer for each part. It includes five steps: data cleaning, handing test and categorical attributes, custom transformer, feature scaling, and transformation pipelines.

Data cleaning

Many times, I noticed that some attributes have some missing values. I can accomplish these efficiently using Pandas' DataFrame to drop miss data. However, I believed that it would be a waste. I decided to implement the following rules: if the feature includes a continuous value, I will replace the missing value with the average of the feature over the other samples, and if the feature includes a discrete value, I will create a new value specifically to identify missing data. Here, Scikit-Learn provides a handy class to take care of missing values: Imputer.

Handing test and categorical attributes

Earlier I left out many categorical attributes because those are a text attribute so I cannot compute its median. Most machine learning algorithms prefer to work with numbers, so let's convert these text labels to numbers. The author of Hands-On Machine Learning with Scikit-Learn and TensorFlow provides a method to encoder these text labels. I can use this method to get one-hot encoding easily.

Custom transformer

In previous, I need to add the custom attribute to the training dataset. Therefore, I want to create a custom transformer class to add this step to the pipeline.

Feature scaling

Machine learning algorithms do not perform well when the numerical input attributes have very different scales. Therefore, I can use Scikit-Learn’s StandarScaler for standardization.

Transformation pipelines

Finally, many data transformation steps need to execute in the right order. I can now create a full pipeline to prepare use Scikit-Learn’s Pipeline.

6 Model Training

Since our data is ready, I will start training models now. I will use three algorithms: Logistic Regression, Radom Forests, and Nonlinear SVM. First, I train model use Logistic Regression without tuning it. Moreover, I use 5-fold cross-validation to get the evaluation. For Logistics Regression classifier without tuning, the F1 score is 0.37 and AUC is 0.72. The ROC curve is in the Figure 8.

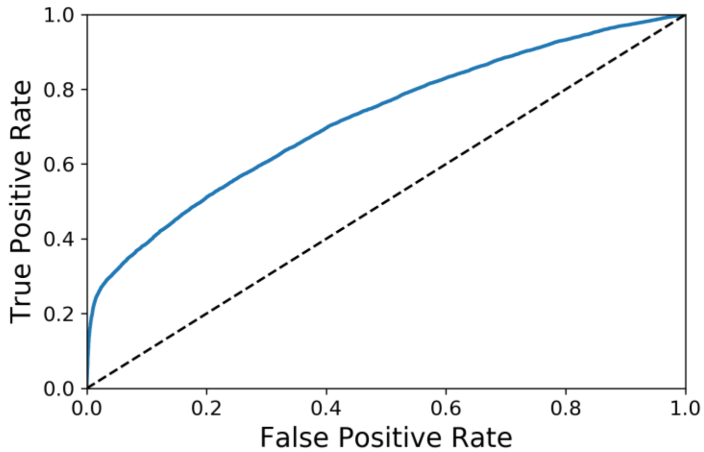

Figure 8 plots a ROC curve for the Logistics Regression classifier without tuning.

7 Model Tuning and Evaluations

Once Logistics Regression trains the model, I first evaluate the model's performance. Next, I can use Scikit-Learn's GridSearchCV to search the best hyperparameters for me. The result is in figure 9.

```
GridSearchCV(cv=5, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid=[{'max_iter': [150, 200], 'penalty': ['l1', 'l2'], 'C': [0.1, 0.2]}],
       pre_dispatch=4, refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
--------------------------------------------------------------
0.749443145843 {'C': 0.1, 'max_iter': 150, 'penalty': 'l1'}
0.745462984891 {'C': 0.1, 'max_iter': 150, 'penalty': 'l2'}
0.749424447875 {'C': 0.1, 'max_iter': 200, 'penalty': 'l1'}
0.745462984891 {'C': 0.1, 'max_iter': 200, 'penalty': 'l2'}
0.745977905642 {'C': 0.2, 'max_iter': 150, 'penalty': 'l1'}
0.741127827695 {'C': 0.2, 'max_iter': 150, 'penalty': 'l2'}
0.745950515301 {'C': 0.2, 'max_iter': 200, 'penalty': 'l1'}
0.741127827695 {'C': 0.2, 'max_iter': 200, 'penalty': 'l2'}
--------------------------------------------------------------
LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=150, multi_class='ovr', n_jobs=1,
          penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

Figure 9 shows model tuning setup and result.

Furthermore, I need to also look for variable importance, i.e., which variables have proved to be significant in determining the target variable. The result is in figure 10.

```
[(0.65272396579348313, 'VehicleAge'),
 (0.38696211723842838, 'MMRAcquisitionRetailAveragePrice'),
 (0.30998676228786726, 'MMRCurrentAuctionAveragePrice'),
 (0.19299381519601849, 'MMRCurrentAuctionCleanPrice'),
 (0.11727835725328331, 'VehOdo'),
 (0.081296408552790675, 'WarrantyCost'),
 (-0.1490488326221881, 'BYRNO'),
 (-0.18549831230862585, 'MMRCurrentRetailCleanPrice'),
 (-0.19789512015029803, '1/6/2009'),
 (-0.25030456432963977, '4/14/2009'),
 (-0.25317249794078478, 'VehBCost'),
 (-0.2552259744960812, '2/16/2009'),
 (-0.25978717446446681, 'MMRCurrentRetailAveragePrice'),
 (-0.27959205770010548, 'WheelTypeID'),
 (-0.31558471284033057, 'MMRAcquisitionAuctionAveragePrice'),
```

Figure 10 shows variables' importance.

Moreover, accordingly, I can shortlist the best variables and train the model again. Finally, I use different algorithms to train the model. In figure 11, I plot three ROC curves for different algorithms: Logistic Regression, Random Forests, and Nonlinear SVM.

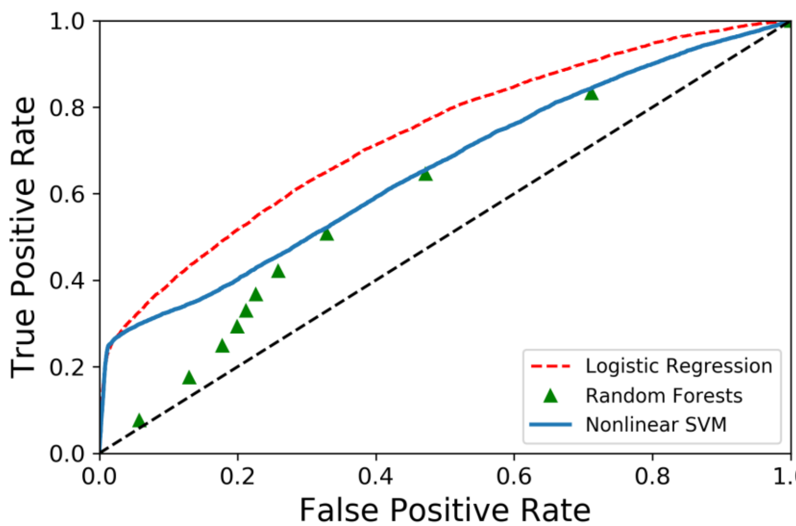

Figure 11 plots three ROC curves of different classifier with tuning after modifying data features.

In the end, I will use those classifiers on test dataset to generate final classification. The result is the figure 12.

Algorithms	Accuracy(%)	AUC	F1 Score
Logistic Regression	95.77	0.733	0.357
Random Forests	96.12	0.609	0.241
Nonlinear SVM	96.79	0.656	0.253

Figure 12 shows a table to compare result for three classifiers.

Reference

1. Aurélien Géron. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. Sebastopol, CA: O'Reilly Media, Inc.
2. Don't Get Kicked! (2012) training & test [Data file]. Retrieved from https://www.kaggle.com/c/DontGetKicked/data
3. Albert, H., Robert R, & Xin, A.W. (2012). *Don't Get Kicked - Machine Learning Predictions for Car Buying.* Retrieved from http://cs229.stanford.edu/proj2012/HoRomanoWu-KickedCarPrediction.pdf