Self-Organizing Maps, XgBoost, Naïve Bayes, Random Forest, KNN

Lemon Map – Unsupervised Predictions For Car Buying

“Using Self-Organizing Maps to classify junk cars”
Volak Sin, August 31, 2017

I Introduction

The goal of this project is to implement an unsupervised deep learning algorithm and compare it to other machine learning algorithms. In this case, we used the data set from the “Don’t get kicked,” Kaggle Competition. The goal of this competition is to decrease the odds of choosing a “bad” vehicles; non running cars that must be sold at a substantial discount to market valuations.

Looking at the competition results from years ago, most competitors used traditional machine learning algorithms such as logistic regression and Support Vector Machines to predict whether a vehicle is considered a “bad buy.” We will also run traditional Machine Learning algorithms and compare them to the accuracy of running a Self-Organizing map on the data set. We will also compare our results to the Human Error rate; the rate of error in the entire dataset.

We split this project into nine sections. The first section is preprocessing the data in order to run the machine learning algorithms. This includes doing exploratory analysis to determine what to do with missing variables. Sections 2 through 8 are dedicated to running more traditional machine learning algorithms; including some of the latest algorithms such as XgBoost.

II Data Processing

The dataset contains both categorical and continuous data for a total of 32 features. The data types of these features are: 10 are float64, 7 are int64 and 15 are objects. Some of the features have missing values while other features seem to be redundant. In looking at the entire dataset, we notice that 12.30% of the data are “bad buys.” We will use this as the human error rate. Normally, we would assume that the human error rate would be a good approximation of Bayes’ irreducible error but since cars are fairly complicated systems, we will assume then it is lower than the human error rate.

Missing Data

Almost half of the features have missing data. Some of the features are missing less than 1% of their data while other features are missing almost 95% of their values. For continuous features that have a small number of values missing, we impute the missing values with their mean. Fortunately, there were no continuous features that had a large amount of the values missing. For the categorical features, that have a small number of the values missing, we imputed the most common value. If the categorical feature had a large number of values missing, we would label these missing values as unknown. In doing this, we give those categorical values that are present a chance to influence our model.

Redundancy and Multicollinearity

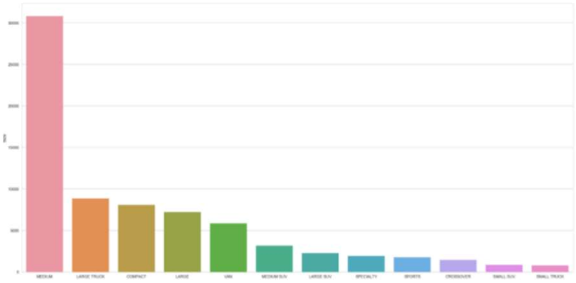
We noticed that some of the features did not contain any new information and in some cases mirrored the information presented by other features. In order to minimize the effects of multicollinearity, we decided to drop these features. The feature “WheelTypeID” presented the same information as “WheelType, ” but in numerical form. Since we will create dummy variables with categorical features, these two features will eventually become identical and therefore we dropped it. We also dropped the feature, “PurchDate,” since we assumed that the date in which a vehicle was purchase has little to no effect on the condition of the vehicle. Depending, on the results of our analysis, we could revise this assumption.

Creating Dummy Variables

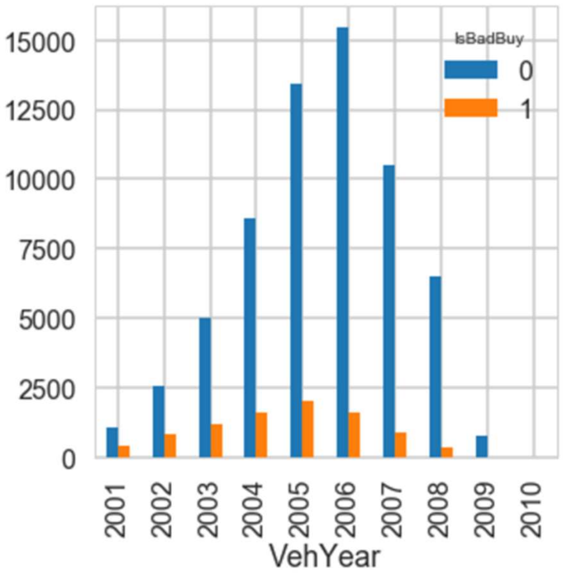
Almost half of the features are categorical. In order for some of our algorithms to work, we must convert these categorical variables to dummy variables. We will use the standard practice of creating dummy variables for categorical features. This practice takes each subcategory and creates a column with the values 1 or 0 representing whether the variable was present for a particular instance.

Histograms & Bar Plots

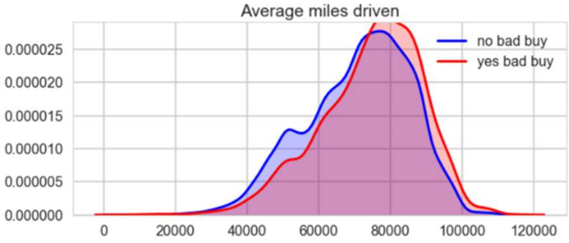
For categorical features, we will use bar plots to quickly visualize the data values. This helps bring attention to any major data oddities.



We see that there are 3 times as many “medium” size cars than any other types. From the vehicle year, we noticed that the year 2005 had disproportionately more bad cars than other years.



From the histogram below, we see that the distribution for “bad” vehicles is slightly shifted to the right when compared to normal cars. This is consistent with our intuition that more miles a vehicle has the more likely that it has mechanical failures and be classified as a bad buy.



III Machine Learning Algorithms

The first algorithm that was used on the dataset was the Logistic Regression. We then followed with the K-Nearest Neighbors, Support Vector Machines, Naive Bayes, Decision Tree, Random Forest and XgBoost.

i. Logistic Regression

Before running our Logistic Regression, we scaled our feature values. This will allow the logistic regression to run faster. Although interpretability becomes harder due to scaling, we can inverse transform our scaled values if we need to interpret our model. Since we are more focus on comparing algorithms than we are improving accuracy, we will also add a feature extraction before feeding data into our regression model. We chose a Linear Discriminant analysis to extract our features because we wanted to maintain the supervised nature of the algorithm. We limited the number of components to 10.

ii. K Nearest Neighbors

For our K Nearest Neighbor algorithm, we made the same preprocessing decisions as the Logistic Regression. We chose the number of neighbors to be 5 and set the distance parameter to be Euclidean.

iii. Support Vector Machine

For our Support Vector Machine, we ran our dataset twice through algorithm. The first run of the algorithm was with a linear kernel. For the second run, we chose a non linear kernel. There choice for non linear kernel was not prompted by any particular assumption about the data. Support Vector Machines requires the features to be scaled but the algorithm does not automatically do so. The Standard Scalar algorithm from SciKit Learn was used before feeding the data into the algorithm.

iv. Naive Bayes

For the Naive Bayes classifier, we used the default parameters. The classifier is also non-linear. The classifier works on the principles of Bayes theorem, but without the requirement of feature independence.

v. Decision Tree

The Decision Tree algorithm does not require that the features be scaled. For maintaining interpretability, the features are feed into the algorithm as is. Since we are only interested in comparing model accuracy, we will scale the features. We chose the standard evaluation criterion for Decision Trees; entropy.

vi. Random Forest

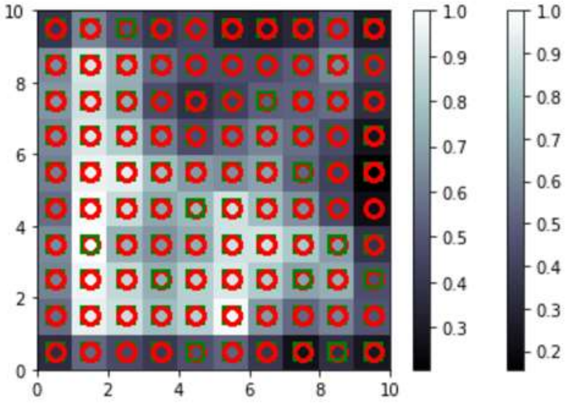
For the first ensemble method, we will chose to run a Random Forest. Since Random Forest are not easily interpretable, we also ran a feature scaling and extraction to further maximum model run time. We chose our number of estimators to be 10 with the same evaluation criteria as our Decision Tree.

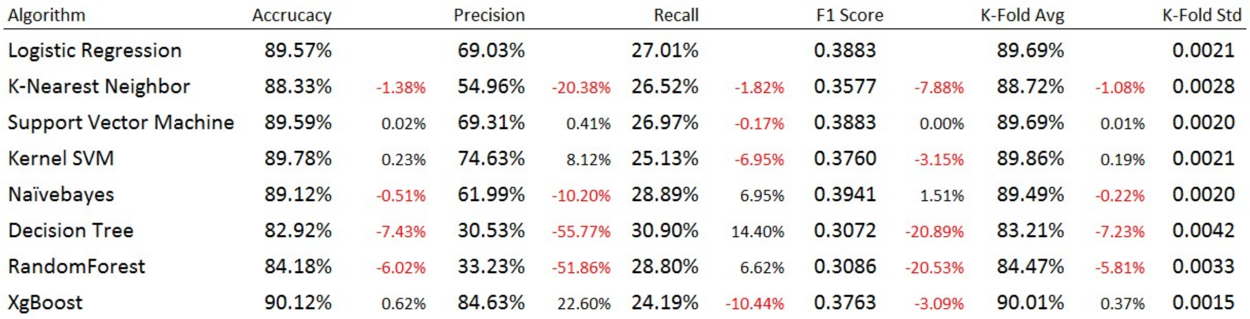
vii. XgBoost

XgBoost is one of the most popular models in machine learning. It is also the most powerful implementation of gradient boosting. One of the major advantages of Xgboost besides having high performance and fast execution speed is that you can keep the interpretation of the original problem.

IV Self Organizing Maps

Self Organizing Maps are an unsupervised deep learning algorithm used for feature detection. Specifically, SOMs are used for reducing dimensionality. They take a multidimensional dataset and reduce it to a 2 dimensional representation of your dataset. To create our initial SOM, we imported the MiniSom package. We chose a grid of 10 by 10. For our outlier detection we chose sigma to be 1. The learning rate is set to 0.5.





Algorithm	Accuracy		Precision		Recall		F1 Score		K-Fold Avg		K-Fold Std
Logistic Regression	89.57%		69.03%		27.01%		0.3883		89.69%		0.0021
K-Nearest Neighbor	88.33%	-1.38%	54.96%	-20.38%	26.52%	-1.82%	0.3577	-7.88%	88.72%	-1.08%	0.0028
Support Vector Machine	89.59%	0.02%	69.31%	0.41%	26.97%	-0.17%	0.3883	0.00%	89.69%	0.01%	0.0020
Kernel SVM	89.78%	0.23%	74.63%	8.12%	25.13%	-6.95%	0.3760	-3.15%	89.86%	0.19%	0.0021
Naïvebayes	89.12%	-0.51%	61.99%	-10.20%	28.89%	6.95%	0.3941	1.51%	89.49%	-0.22%	0.0020
Decision Tree	82.92%	-7.43%	30.53%	-55.77%	30.90%	14.40%	0.3072	-20.89%	83.21%	-7.23%	0.0042
RandomForest	84.18%	-6.02%	33.23%	-51.86%	28.80%	6.62%	0.3086	-20.53%	84.47%	-5.81%	0.0033
XgBoost	90.12%	0.62%	84.63%	22.60%	24.19%	-10.44%	0.3763	-3.09%	90.01%	0.37%	0.0015

V Results

Our baseline model, the Logistic Regression performed fairly well; outperforming half of the other algorithms. The Decision Tree and Random Forest not only did worse than the Logistic Regression it did worse than our human error of 12%. The accuracy of each algorithm after doing a k-Fold Cross validation were approximately the same as when running a train / test split. When comparing the F1 score, the harmonic mean of precision and recall, we see that the Logistic Regression performed the second best. The model that performed the best according to the F1 score was the Naïve Bayes Classifier.

Our original intent was to use the Self Organizing Map to classify whether a vehicle was a bad buy. The hopes was that the characteristics of bad purchases could be delineated by sigma separation of 1 or two. After trying both strategies, we were not able to adequately separate these bad purchases. The alternative strategy was to take the results of the SOM dimensionality reduction to feed into a neural network to predict the probability of being a bad purchase. The resulting algorithm can help the company avoid bad purchases by setting risk threshold.

VI Summary

Even though we were hoping that running a Unsupervised Deep Learning algorithm will produce similar results to our other algorithms, the unsupervised nature of the algorithm made its accuracy predictions less than ideal; unusable. We were able to achieve a 90% accuracy rate with XgBoost; which is greater than the human error rate and around the maximum of model accuracy. If we were even more accurate, that could be indicative of overfitting our model