Extracting Useful Information from Vehicle Identification Numbers with Machine Learning

Brendan McSweeney*, Nathan Rich*, Seth Percy*, and Professor Hady Ahmady Phoulady
Computer Science, University of Southern Maine

***Abstract*— Vehicle identification numbers ("VINs") are required by law to be associated with all vehicles sold in the United States. These numbers are not simply serial numbers but contain many points of data about the vehicle. Our objective was to create a classifier capable of determining the vehicle year, make, and model given only the VIN for any vehicle produced in the past ten years. We began by collecting data on cars for sale on the internet using a data scraping tool. We then trained and tested the accuracy of several different types of classifiers provided by SciKit-Learn for each of the three label sets independently (vehicle year, make, and model). After finding that the decision tree classifier had high accuracy across all three label sets when examined separately, we created a single decision tree classifier to predict all three label sets (known as a multi-label multi-class classifier). This multi-label multi-class classifier was ultimately found to have low accuracy when predicting vehicle model compared to its independent (multi-class, single-label) counterpart. Finally, an accurate predictor was created by stitching together the three independent decision trees.**

***Index Terms*— Machine Learning, Vehicle Identification Numbers, VINs, Python, SciKit-Learn, Data Scrapping Tool, Year, Make, Model, Decision Tree, Multi-Label, Multi-Class**

I. Introduction

Vehicle identification numbers in the United States follow a structured format with certain pieces of information contained in certain placements of their 17 characters. A visual breakdown of VIN structure is shown in figure I-1.

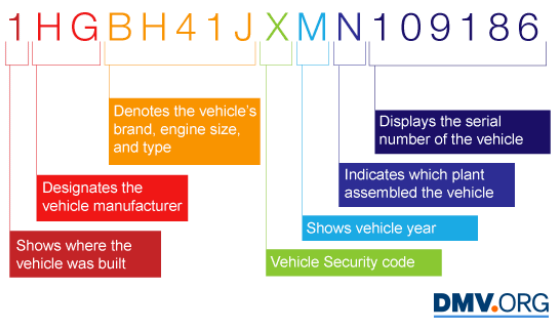

Figure I-1. Breakdown of information contained in VIN. [1]

Given the consistent structure of vehicle identification numbers, it can be expected that a classifier algorithm should be successful in discerning, at least, the vehicle year, make, and model when given the VIN. This was our objective. This type of problem, having multiple sets of potential labels which can be applied to a single input, is known as a multi-label multi-class problem. As the number of vehicle makes and models across all years is quite high, we chose to focus on only the last ten years. While only some of the 17 characters of the VIN are relevant to the data points we wished to predict, we still wished to use the entire VIN as the input to our models - thus allowing our models to learn their own features. We decided that we would test several classification algorithms, and select the most accurate for further experimentation.

As no relevant data set was found to be publicly available, we set out to create our own. This was done by means of an extension for the Chrome web browser which was used to automate the collection of data from a car sales website.

II. Data Collection and Normalization

The new and used vehicle sales website Cars.com contains a very large number of cars for sale, as well as complete information for most vehicles listed, presented in a consistently structured format. This made Cars.com ideal for automated data scraping. We employed an extension for Google's Chrome browser known as Web Scraper [3] which permitted us to select the data we wished to scrape from each page and collect it into a comma-separated values file quite rapidly. We needed only to instruct the scraper of which pages to visit and what data to scrape. The former problem was easy to solve since the URLs of cars for sale on Cars.com contained a numerical identifier. Thus, we instructed the scraper to progressively increment the ID contained within the URL, allowing it to rapidly locate information on several different cars. The desired data to extract was specified using a combination of CSS selectors and regular expressions. The scraper configuration file is included with this report. It can be imported into any machine having the Chrome extension installed, allowing for further data collection. The raw data collected by the scraper is also included with the report.

Once we had our data, we needed to clean and normalize it. This was done using a Python script, which is included with this report. The cleaning script evolved as we did our training

in order to extract and vectorize features efficiently, and that process is described in the section on training methods. However, many aspects of the data cleaner remained unchanged from the beginning. First, we needed to drop any collected samples that lacked a VIN number. Next, we needed to remove filler words used in car sales marketing such as "New", "Used", "Certified Pre-owned", and the like. Next, we needed to separate the year, make, and model into separate data columns, as the structure of Cars.com forced us to scrape them as one whole string. In order to normalize the data, we dropped any non-alphanumeric characters (such as hyphens) from the make and model names and transformed all our data to uppercase. We also addressed special cases where the vehicle maker name may have included a space by removing the space in those cases (e.g., converting "LAND ROVER" to "LANDROVER"). Finally, we needed to drop any duplicate records (using the VIN as a unique ID) since multiple runs of the scraper had created some overlap.

III. TRAINING METHOD

Once our data was collected and cleaned, we began by attempting to train a classifier to find the model year of the vehicle given its VIN. This was done by using a SciKit-Learn pipeline consisting of a HashingVectorizer and a OneVsRestClassifier using logistic regression. This proved unsuccessful. In all cases, our model would predict that the vehicle's model year was 2019. Upon examination, it became apparent that the classifier was arriving at this decision simply because 2019 was most common among training examples, and our method of feature extraction by hashing the entire VIN was faulty - the result was a unique string as the feature for each training example, which taught our model nothing.

In order to remedy this, we took a new approach to feature extraction. We split each character in the VIN into its own feature, and where the character was nonnumeric, the character was transformed to a number by a simple mapping (A = 10, B = 11, C = 12, and so on). This was done in the data cleaning step. Thus, we were able to dispense with the use of the HashingVectorizer as our data was already vectorized once it had been cleaned. A stratified shuffle split, as implemented by SciKit-Learn, was employed to split our data into train and test sets such that labels were proportionally represented in each set.

Using this new set of feature data, we tested several classifiers provided by SciKit-Learn. Some of these classifiers threw warnings that the sample size of some certain labels was too low. Upon inspection, we found that certain model years had only one or two samples. All such years were found to be greater than ten years in the past. This was due to the nature of Cars.com, the website from which we collected our data - the site is heavily biased towards the sale of new and late-model used cars. Thus, we dropped all samples before the model year 2009. This allowed our classifiers to proceed without issue in learning about the years we were interested in. The distribution of this newly cleaned model year data is shown in figure M-1. The test results with the accuracy of each classifier are shown below, in figure M-2.

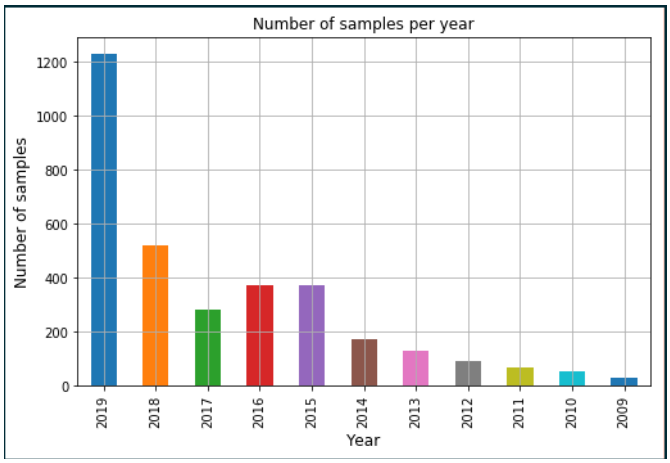

Figure M-1. Number of samples per model year (after dropping pre-2009 samples), showing our data weighs heavily towards recent years.

```
VIN to YEAR
==============================
KNeighborsClassifier
****Results****
Accuracy: 50.0750%
==============================
SVC
****Results****
Accuracy: 37.0315%
==============================
DecisionTreeClassifier
****Results****
Accuracy: 100.0000%
==============================
RandomForestClassifier
****Results****
Accuracy: 89.2054%
==============================
AdaBoostClassifier
****Results****
Accuracy: 66.4168%
==============================
GradientBoostingClassifier
****Results****
Accuracy: 100.0000%
==============================
GaussianNB
****Results****
Accuracy: 99.8501%
==============================
LinearDiscriminantAnalysis
****Results****
Accuracy: 93.8531%
```

Figure M-2. Results of initial test of multiple classifiers using 17 features (one for each character in the VIN) and one set of labels (the model year).

Following our successful test of the accuracy of different classifiers in predicting the model year, we decided to use the same testing format for predictions of vehicle make and model (separately). The methodology used for make and model

differed only slightly from the methodology used for predicting the year. First, the labels needed to be encoded, as they were textual, rather than numerical, as was the case when predicting the year. Additionally, a function was employed at the data-reading stage (rather than the data-cleaning stage) to remove labels with low numbers of samples. Upon testing, removing labels with fewer than six examples proved effective. The data distribution and accuracy of vehicle make prediction are shown below in figures M-3 and M-4, respectively.

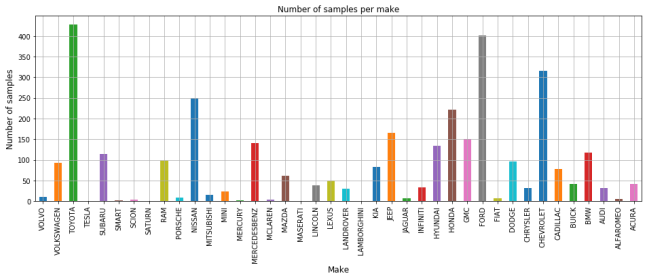

Figure M-3. Number of samples per manufacturer, before dropping underrepresented labels.

```
VIN to MAKE
==============================
KNeighborsClassifier
****Results****
Accuracy: 92.9217%
==============================
SVC
****Results****
Accuracy: 12.9518%
==============================
DecisionTreeClassifier
****Results****
Accuracy: 97.4398%
==============================
RandomForestClassifier
****Results****
Accuracy: 98.0422%
==============================
AdaBoostClassifier
****Results****
Accuracy: 30.2711%
==============================
GradientBoostingClassifier
****Results****
Accuracy: 99.0964%
==============================
GaussianNB
****Results****
Accuracy: 71.2349%
==============================
LinearDiscriminantAnalysis
****Results****
Accuracy: 68.2229%
```

Figure M-4. Training accuracy of multiple classifiers in predicting the vehicle make based on VIN.

Vehicle model classification was much more labored than other classifications, likely due to the very large number of potential labels. While all classifiers could be tested in under 30 seconds for the year and make, this same process took several minutes for the vehicle models. The accuracy of the classifiers in predicting vehicle model is shown below in figure M-5. Due to the very large number of labels, a data distribution graph would be extremely large and so is not shown.

```
VIN to CAR MODEL
==============================
KNeighborsClassifier
****Results****
Accuracy: 87.0279%
==============================
SVC
****Results****
Accuracy: 2.9557%
==============================
DecisionTreeClassifier
****Results****
Accuracy: 95.0739%
==============================
RandomForestClassifier
****Results****
Accuracy: 97.3727%
==============================
AdaBoostClassifier
****Results****
Accuracy: 3.2841%
==============================
GradientBoostingClassifier
****Results****
Accuracy: 3.1199%
==============================
GaussianNB
****Results****
Accuracy: 94.4171%
==============================
LinearDiscriminantAnalysis
****Results****
Accuracy: 80.9524%
```

Figure M-5. Training accuracy of multiple classifiers in predicting the vehicle model based on VIN.

Examining the results of each classifier test, we note that only one classifier performs with greater than 90% accuracy in all three test cases - the decision tree classifier. This classifier performed with accuracies of 100%, 97.4%, and 95% when predicting vehicle year, make, and model, respectively. Thus, we elected to do further examination of this type of classifier.

SciKit-Learn's implementation of the decision tree classifier includes the ability to output a visualization of the decision tree used in the trained model. This information is extremely useful to see the trained model in a human-readable fashion. We used this feature to output decision trees for each of our three test cases (vehicle year, make, and model). Upon inspection of the decision tree used to determine vehicle year, we could see that the classifier was making its decisions mostly based on character 10 of the VIN. This matches our expectations since this is the character that is officially used to denote the model year [1]. Since all 17 characters of the VIN were used as inputs and the classifier generally used only the

expected feature, this demonstrated that the classifier was successfully learning its own features. The decision trees for vehicle make and vehicle model were substantially more complex, but again they generally chose to use features that met with our expectations, with characters two and three being used to determine the vehicle make, and characters four through eight used in concert with two and three to determine the model. These, too, match with the official government specification of the VIN [1]. A sample of this decision tree visualization is shown below in figure M-6. Complete decision tree visualizations for all three test cases are included as appendices to this paper.

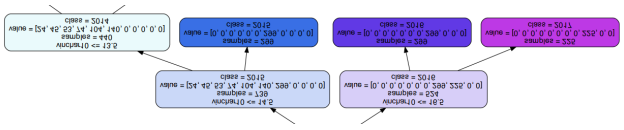

Figure M-6. A small sample of the visualized decision tree used by the decision tree classifier in predicting the vehicle year. In testing, this decision tree was 100% accurate.

Finally, having three separate decision tree classifiers with acceptable accuracy levels, we elected to train a single decision tree classifier that would be capable of predicting the year, make, and model altogether. This would be a multi-class, multi-label classifier. The implementation of decision tree classifier provided by SciKit-Learn supports the training of this type of model, however, some parts of our workflow needed to be modified. First, we discovered that the implementation of stratified shuffle split provided by SKLearn does not support multidimensional arrays, thus precluding its use. Therefore, we had to resort to a random train/test split. This is less than ideal, but by testing several different random data splits we could attempt to discern an optima. Additionally, the accuracy scoring function we had used in previous cases did not support multi-class multi-label algorithms. We discovered that by performing some relatively simple math, we could gain insight into our accuracy by calculating the Hamming loss - the fraction of labels that were predicted incorrectly. We were able to obtain the Hamming loss for each label category predicted by our classifier (year, make, and model). Finally, we discovered that the visual graph creation module of the decision tree classifier does not have the ability to show textual class names for multi-output algorithms - this is a documented limitation [2]. Thus, the labels are shown as vectors, which severely limits the insight they are able to provide. The decision tree visualization is quite large, and is provided as an appendix (mega-tree.pdf).

After several runs of our multi-class multi-label classifier using different randomized train/test splits, we found that our Hamming loss did not vary very much. The model year was predicted with no loss whatsoever in nearly all test runs. The make and model had typical losses of 4.6% and 32.5%, respectively. Through all test runs, the Hamming loss for the make and model varied by at most 0.2%, a relatively small variation. The output of a typical Hamming loss is shown in figure M-7.

```
Hamming losses:
year     0.000000
make     0.046099
model    0.325735
```

Figure M-7. Typical Hamming losses of the multi-class multi-label decision tree classifier.

Examining the Hamming losses, we note that the loss associated with the model year and make are very close to those exhibited by their respective individual (single-label) classifiers discussed earlier. However, the loss associated with the vehicle model is significantly higher than that of its single-label classifier.

IV. Conclusion

Given the high accuracy of our three individually-trained decision tree models (vehicle year, make, and model), it is clear that the use of these three models in concert is an excellent way to extract useful information from vehicle identification numbers. As the number of different vehicle models is quite high relative to the number of available training samples, we believe that accuracy could be improved even further by employing additional training data. Additional data would also permit us to expand our algorithm to include less common vehicle makes and models, some of which had to be dropped due to low sample numbers.

Given the high Hamming loss of the multi-label multi-class decision tree algorithm when predicting vehicle model, use of this model would not be effective. This may be due to the implementation of decision tree classifier in SciKit-Learn. Moving forward, it would be interesting to see whether other machine learning toolkits using other methodologies (such as a neural network produced using TensorFlow) would be able to address the multi-class multi-label problem more accurately.

Acknowledgments

The authors would like to extend their sincere appreciation to Professor Phoulady and the USM Computer Science Department for their guidance and support in developing this project.

References

[1] "Free VIN Decoder." DMV.ORG. Accessed December 02, 2018. https://www.dmv.org/vehicle-history/vin-decoder.php. "Sklearn.tree.export_graphviz." Scikit-learn 0.19.2

[2] Documentation. Accessed December 03, 2018. https://scikit-learn.org/stable/modules/generated/sklearn.tree.export_graphviz.html

[3] "Web Scraper." Google. Accessed December 03, 2018. https://chrome.google.com/webstore/detail/web-scraper/jnhgnonknehpejjnehehllkliplmbmhn