Identify a Vehicle and its Driver from the Vehicle's Maneuver Data

Mehrdad Salehi and Ding Zhao

Abstract

In this project, we applied supervised learning algorithms to detect drivers based on their driving behavior. Using labeled driving data (time, speed, acceleration, heading and gas usage), we created a set of features such as maximum of speed, standard deviation of acceleration as well as additional complex features like variation of heading and variation of MPG over time. In addition, we applied PCA to convert a set of possibly correlated features into a set of values of linearly uncorrelated variables and optimize our algorithm performance. After applying the PCA, the accuracy results were improved from 86.0% to 87.7% and the running time was also decreased (using SVM model). We applied SVM, Random Forest, Decision Tree, Naïve Bayes, and Nearest Neighbor algorithms to the data and compared the accuracy and running time of the results for different features sets. Adding complex features greatly improved the classification results for all algorithms. Among the above mentioned algorithms, SVM optimized for its parameters based on a grid search provided the best accuracy of 96%. However, the running time of SVM was the longest.

1 Introduction

Lots of identification methods have been developed and widely used in security agencies and business companies. Most of existing methods are based on biomedical identification, such as fingerprint, face and retina. However, behavior signatures can be more valuable because they contain much more information about a person. We hypothesize that an individual's driving behavior is a unique signature that can be used in identification (Huihuan et al., 2010). We collect the driving data from sensors that measure the instantaneous speed, heading, acceleration and gas MPG. The objective of this project is to create a machine-learning algorithm to identify a car and its driver from detailed driving data. Below we describe how we model this problem and the algorithms we use to tackle it.

2 Task Definition

The task is to build a system that can identify a driver from detailed driving data. Here we describe the mechanics, training and test process of the program:

2.1 Problem Mechanics

We hypothesize that everyone has her own unique driving habit. We may be able to extract enough information from her driving data to create a unique signature that can be used as an identification method. More specifically, we want to build a program that is able to learn an individual's driving signature and correctly find the ownership of a given driving data.

2.2 Inputs & Outputs

The input for the program is the driving data from sensors that measure the instantaneous speed, heading, acceleration and gas MPG. The data for these sensors is collected every second. The data is collected from 17 cars with duration that vary from one week to several months. The datasets are labeled, i.e., we know the vehicle for which each data file belongs to. The program is supposed to learn enough information from the data and when given a piece of unknown driving data, it is able to output the ownership of the data.

3 Model

We treat driver identification as a classification problem. First, we process the raw data to let it be more appropriate for feature engineering. Then, we extract features from the processed data, and at the end we apply supervised learning. Below, we will describe the model for the problem.

3.1 Data Processing

3.1.1 Trip Extraction

The raw data includes instantaneous speed, heading, acceleration along the three dimensions x, y, and z and gas MPG that are measured every second for each vehicle. If we treat data as it is provided (i.e., consider data only for 17 vehicles), we have so few data and any model we create will have a high chance of over-fitting. In order to avoid such a problem, we create "trips" for each vehicle. A trip is defined as a set of vehicle data that starts from one zero speed and ends to the next zero speed. In some cases, the vehicle is stationary in one location for several seconds. In order to avoid adding noise to trips in such cases, we removed multiple stationary points from the input data. Using this pre-processing approach, we avoid overfitting by creating tens of thousands of trips. From the raw data, we were able to generate around 80,000 trips.

3.1.2 Bad Reads Recovery

Occasionally sensors can generate invalid reads, and these reads are discarded in the raw data set. In order to feed the program with consistent and continuous data, we need to recover these bad reads. Speed, acceleration, and heading are all continuous properties, and therefore we can use neighboring reads to recover these bad reads. Suppose that the last valid read is at time t_{last}, and the next valid read is at time t_{next}. Since sensors generate read every second, if $t_{last} - t_{next} > 1\ sec$, we know there are some bad reads in between. Let $r_1, r_2, \dots r_n$ be the bad reads between t_{last} and t_{next}. Also let r_{last} and r_{next} be the last valid read and the next valid read. We approximate the change of read as linear and recover bad reads using the following formula:

$$r_i = (r_{last} + r_{next}) \frac{i}{2(n+1)}$$

3.2 Feature Engineering

Once the trips are created, we can attribute features to them. Feature engineering is a critical part of this project as the more relevant features are selected the better the algorithm learns and hence will provide more accurate classification results.

3.2.1 Feature Extraction

Speed

For each trip, we use the max, the mean, and standard deviation of speed as our features because they imply the type of road (city or highway) the driver usually drives in. Also, they can reflect whether the driver tends to overspeed.

Acceleration

Acceleration read depends on how the accelerometers are mounted on a car. Since the accelerometers are mounted differently in different cars, i.e., the X direction is not consistently defined for all the cars, we will have totally different acceleration vectors for different cars. If we try to align the axis of acceleration vector with respect to the car (forward, side, and upward), we will lose a large amount of precision. Because acceleration is the first derivative of speed with respect to time, we use the difference of consecutive speed reads over time as the acceleration. We use the average and standard deviation of the acceleration as our features.

Heading

The heading of a vehicle is the direction where it is traveling. The heading data is in the range of 0 degree to 360 degree, with 0 degree denoting the north. Since the absolute value of the heading just mean the direction the driver is facing, it makes no sense for us to use mean or maximum values of heading. However, some drivers may tend to change the car's heading left and right while going in a straight line. Therefore, we use standard deviation of heading as an estimate.

Miles Per Gallon

Miles per Gallon is a metric to evaluate the energy efficiency of a vehicle. We could not use the absolute value of MPG because absolute value of MPG depends only on the make of the car. However, we could take the standard deviation of the MPG to see how driver's behavior change fuel efficiency.

Complex Features

In addition to applying pure statistical methods to our data, we extracted some additional features and were able to achieve much better results. We call these features that are not directly measured as complex features. Below is the list of our complex features:

1. More from heading changes: How fast a driver change car's heading indicates how wide he makes turns. We need to use the first derivative of heading to represent such feature. However, one thing to note that the heading changes over time actually is related to the acceleration of the car, as the centripetal acceleration has a component in the forward direction. To remove such relationship, we extract the sideward component of the heading changes using speed and heading changes. After applying the knowledge of centripetal acceleration and Newton's Law, we found that $Side\ component \propto v \cdot \Delta\ heading$.

2. More MPG measurement: We found that only taking the standard deviation of MPG does not fully represent the fuel efficiency changes. We further make use of the MPG changes by taking the derivative of MPG values.

3.2.2 Scaling

We scale each column of the $m \times n$ feature matrix X to have zero mean and unit variance by the formula:

$$X_{ij} = \frac{X_{ij} - mean(X_{1j}, X_{2j}, \dots, X_{mj})}{stdv(X_{1j}, X_{2j}, \dots, X_{mj})}$$

where

$$mean(X_{1j}, X_{2j}, \dots, X_{mj}) = \frac{1}{m} \sum_{i=1}^{m} X_{ij}$$

$$stdv(X_{1j}, X_{2j}, \dots, X_{mj}) = \sqrt{\frac{\sum_{i=1}^{m} (X_{ij} - mean(X_{1j}, X_{2j}, \dots, X_{mj}))^2}{m}}$$

3.3.3 Principal Component Analysis and Feature Space Reduction

Principal component analysis (PCA) is a statistical procedure that utilizes orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables (principal components). PCA was leveraged to reduce the dimension of features, automatically detect and eliminate noise in the feature set, minimize the complexity of the hypothesis class considered, avoid overfitting, and optimize our algorithm's performance. In conjunction with PCA, we applied a whitening transformation to ensure that our feature vector contained unit component-wise variances.

3.3 Supervised Learning

After generating features, the next step is to apply a supervised learning algorithm. For this project, we tried the algorithms below and compared their accuracies and run time.

3.3.1 Support Vector Machine

A support vector machine (SVM) constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space to maximize the distance to the nearest training data points of any class. Since an SVM is only capable of binary classification, the "one-against-one" approach will be applied toward multi-class classification. As we have five classes, $5 * \frac{5-1}{2} = 10$ classifiers were constructed and each trained data from two classes. Each SVM is assigned a weight of 1, and predictions are made based on the largest number of votes the SVMs collectively assign to a particular class.

Grid Search and K-fold Cross Validation

The kernel type *K*, the error term *C* and kernel coefficient γ, were three important SVM input parameters requiring adjustment. We selected a Gaussian kernel as our kernel function, since it constructs a hyper-plane in an infinite dimensional space and is reputed to be among the most powerful kernel functions available. *C* balances between the dual goals of minimizing$||w||^2$ and of ensuring that most examples possess a functional margin of at least 1.0. Larger values of γ correspond to worse balanced classification, which suggests that the deflective classifier model is causing lower accuracy on one side of the classification destination and higher accuracy on the opposite side. We applied a two-step grid-search method that tests every combination of possible C and γ values ($C = 10^{-5}, 10^{-4}, \dots, 10^{5}$, $\gamma = 10^{-10}, 10^{-9}, \dots, 10^{1}$) to find the optimal combination. After conducting a search with a coarse grid to locate a better region, the method constructs a finer grid within that region.

To evaluate the accuracy of each combination, we used a 3-fold cross validation on the data. We found that $C = 8192$, $\gamma = 0078125$ gives us optimal solution.

3.3.2 Decision Tree and some other Machine Learning Methods

In addition to SVM, we also apply other machine learning methods to solve our problem:

- *Decision Tree:* A non-parametric supervised learning method that constructs a decision tree that maps features vectors to labels.
- *Random Forests*: Fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting
- *Naïve Bayes:* A simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions.
- *Nearest Neighbor:* A classification model that assigns to observations the label of the class of training samples whose mean (centroid) is closest to the observation.

4 Test Results and Analysis

4.1 Summary

We tested our models using the driving data from 5 drivers. Therefore, it is a 5-class classification problem, i.e. the model predicts driver's identity given any piece of driving data from 5 drivers. In order to validate and improve the results, we did a 3 fold cross validation on the 5000 data points (1000 for each driver). We measured the accuracy of our classifier as number of correct classifications divided by total number of test data. We achieved 96.0% accuracy using parameter-optimized SVM with 5-class classification. In next sections, we will include an analysis of the effects of each method we used and also a comparison of different machine learning algorithms.

4.2 The Effect of Scaling Features

Enormous improvement was achieved by scaling features — from 27.3% accuracy in 10.9 seconds to 87.7% accuracy in 2.07 seconds if we use SVM with default parameters. This was due to the fact that scaling disallows attributes in greater numeric ranges (i.e. maximum speed) from dominating values within a smaller numeric range (i.e. standard deviation of acceleration). In fact, the introduction of scaling resulted in accelerated calculations and the avoidance of computational difficulties, such as overflow/underflow caused by the kernel values' dependence on the inner products of feature vectors.

4.3 Principal Component Analysis and Feature Space Reduction

Principal component analysis was one technique applied to reduce the feature space. To identify the optimal number of features to incorporate, we evaluated our model's performance while varying the number of components retained. Our results appear below.

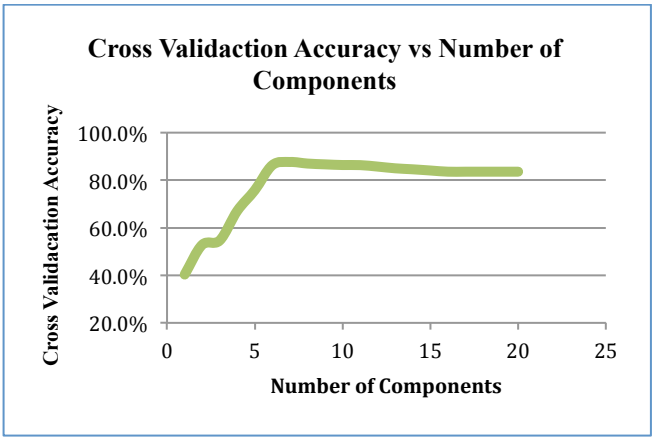

Figure 1 Cross Validaction Accuracy vs Number of Components

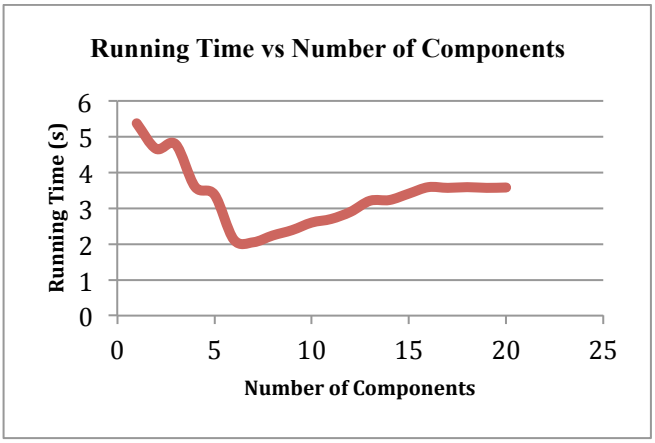

Figure 2 Running Time vs Number of Components

Initially, as the number of components increases, predictive accuracy dramatically improves (from 40.3% to 87.7). This result can be explained by the increased preservation of useful features. However, when the number of features retained exceeds 7, noise is subsequently introduced and we begin incorporating irrelevant features or those that overlap with existing features. Therefore, we think that 7 is the best number of components to keep with PCA transformation. Compared to the accuracy of 86.0% without PCA, we get an accuracy of 87.7% with PCA optimization using 7 components.

4.4 The Effect of Individual Features

As we gradually include the features of Speed, Heading, MPG, Acceleration, Time, Complicated MPG, Complicated Acceleration, the cross validation accuracy of our SVM model increases. This result is illustrated in the figures below.

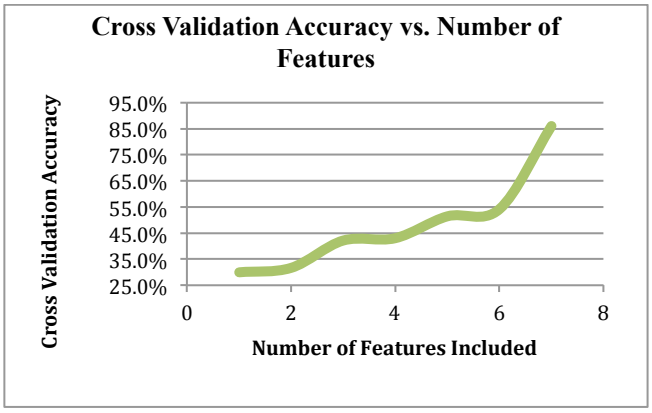

Figure 3 Cross Validation Accuracy vs. Number of Features

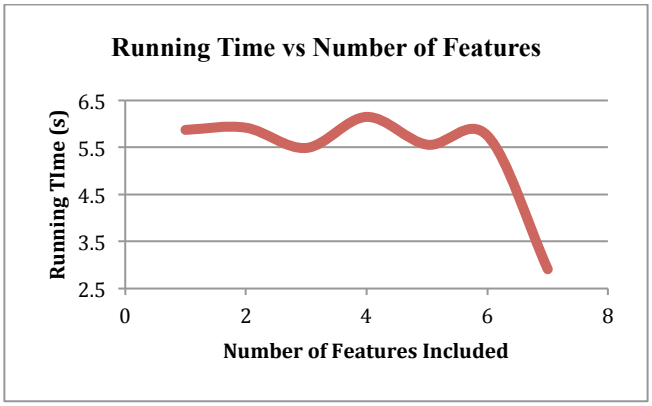

Figure 4 Running Time vs Number of Features

From the graph, we can see that every feature we introduce is useful for our learning algorithm. The two last features from complex features give us the greatest improvement. The reason is that they capture very realistic driver behaviors. It is a little surprising that as we add the last two features, the running time also decreases. This is mainly due to faster convergence in SVM algorithms when we introduce important features.

4.5 Comparison of Different Machine Learning Algorithms

We measured the accuracy and performance of different machine learning algorithms, and the results are shown below:

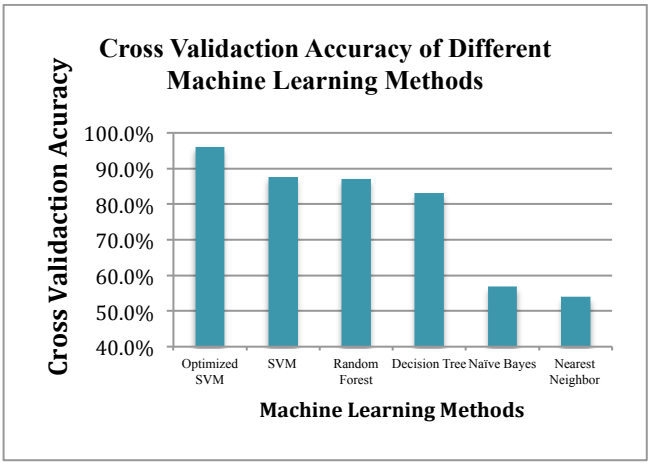

Figure 5 Cross Validation Accuracy of Different Machine Learning Algorithms

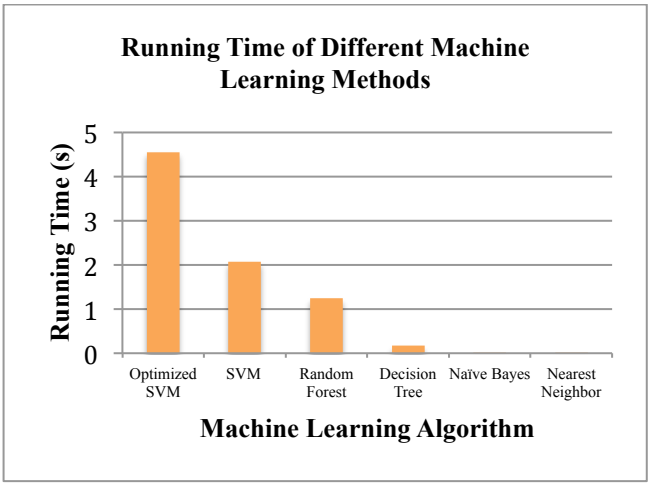

Figure 6 Running Time of Different Machine Learning Algorithms

We can see that there is actually a trade-off between performance and accuracy. Nearest Neighbor and Naïve Bayes are very simple algorithms. They run really fast and almost finish instantaneously. However, the accuracies they produce are not so good. These methods are very useful if we want to get a sense of how our model works and get results very quickly. SVM and random forest, on the other hand, give us very high accuracy but take a lot of time. They are more suitable if we have big computational power and memory space.

5 Conclusions

In this project, we proved that an individual's driving behavior is a unique signature that can be used in identification. We collected the driving data from sensors that measure the instantaneous speed, heading, acceleration and gas MPG. We successfully created a machine learning algorithm to identify a car and its driver from detailed driving data.

We first processed the data to make it appropriate for supervised learning. We avoided overfitting by creating trips from each vehicle data. A trip is defined as a set of vehicle data that starts from one zero speed and ends to the next zero speed. Also, we removed multiple stationary points from the input data to avoid adding noise to the data. The second step was feature engineering where we created different features for the learning algorithms. Since the acceleration data was collected in an inconsistent and noisy way, instead of using the collected acceleration, we derived it from speed and time. Also, we extracted a number of complex features such as the variation of heading over time, variation of MPG over time, and ratio of MPG to speed.

In addition, we applied PCA to convert a set of possibly correlated features into a set of values of linearly uncorrelated variables and optimize our algorithm performance. After applying the PCA, the accuracy results were improved from 86.0% to 87.7% and running time was decreased.

At last we applied supervised learning algorithms to detect drivers based on the driving behavior. We applied SVM, Random Forest, Decision Tree, Naïve Bayes, and Nearest Neighbor algorithms to the data and compared the accuracy and running time of the results for different features sets. Adding complex features improved the classification results for all algorithms dramatically. Among the above mentioned algorithms, SVM optimized for its parameters based on a grid search provided the best accuracy of 96% for 5-class classification. However, the SVM running time was the longest (4.55 seconds compared to 0.01 seconds using nearest neighbor).

References

Q. Huihuan, O.Yongsheng, W. Xinyu, M. Xiaoning, and X. Yangsheng (2010), *Support Vector Machine for Behavior-Based Driver Identification System*. Journal of Robotics, Volume 2010.

H. Chih-Wei, C. Chih-Chung, and L. Chih-Jen (2010), *Practical Guide to Support Vector Classification.* National Taiwan University, Taipei 106, Taiwan.