Stay Alert!: Creating a Classifier to Predict Driver Alertness in Real-time

Aditya Sarkar, Julien Kawawa-Beaudan, Quentin Perrot

Friday, December 11, 2014

1 Problem Definition

Driving while drowsy inevitably leads to road accidents. There are about 100,000 crashes every year attributed to drowsy driving in the US, with an associated 12.5b in annual losses. In fact, as many as one third of fatal car accidents are linked to drowsy driving. It would be extremely useful if cars could determine whether the driver is alert or not in real-time. The objective of this investigation, then, is to create a classifier that will determine if a driver is alert or not alert.

1.1 Describing the Ford Dataset

The dataset was provided by Ford, and shows the result of many observations of driver behavior. For each driver, there are 2 minutes of sequential data, recorded every 100ms. There are samples from 100 drivers from diverse backgrounds. For each observation, an output IsAlert is labeled. IsAlert = 1 indicates that the driver is alert, while IsAlert = 0 indicates that the driver is not alert or drwosy. For each observation, the data has 30 different features values; 8 of these are physiological (features start with 'P'), 11 are environmental (features start with 'E') and 11 are vehicular (features start with 'V').

In addition to the training data, we were also provided data to test with. Not only did this give us more data to work with, it also took away the decision on how to split the data into training and testing datasets. This pre-divided testing set eventually turned out to be problematic, as will be discussed later.

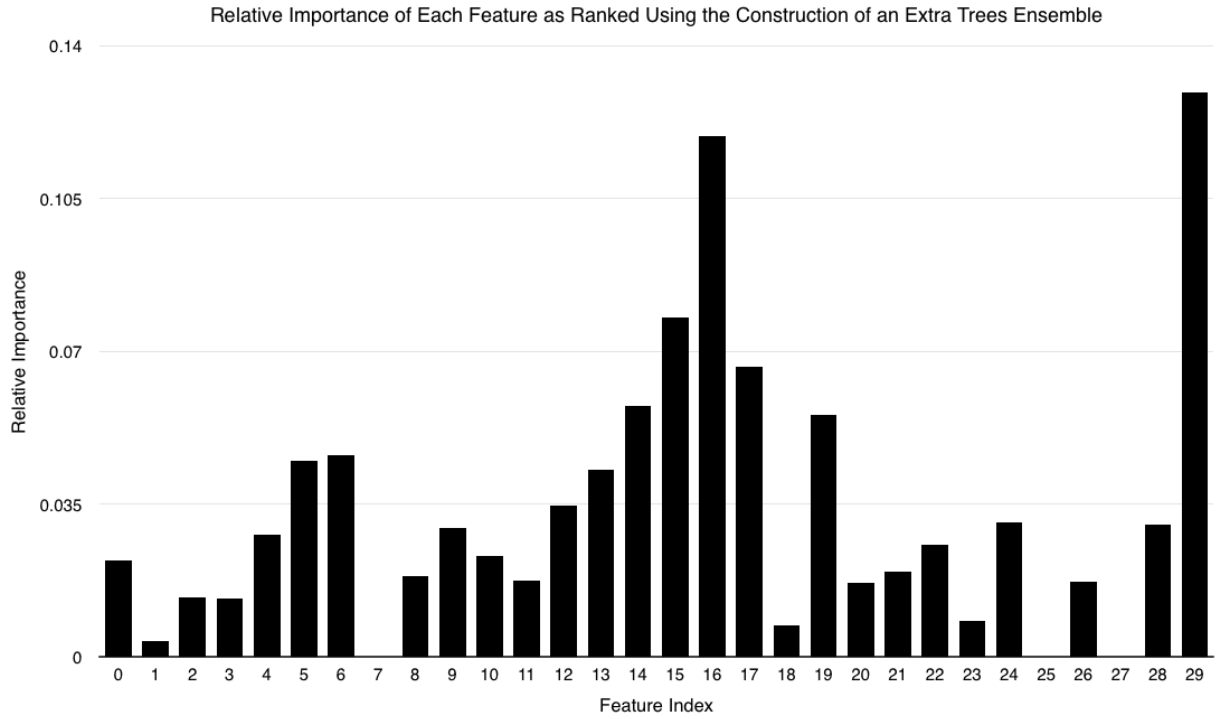
1.2 Key Challenges

One challenge to creating a classifier was that we could not use intuition in our predictions about which features to use, since the features are only labelled 'P', 'E', or 'V'. If we knew what each feature described, we could predict, for example, that it might be useful to include the feature for "time of day". Instead, we had to perform some statistical analysis of the data to determine which features had the largest difference between when IsAlert = 1 and IsAlert = 0. We also challenged ourselves to use as few physiological features as possible. This makes sense, since it is difficult to observe the driver's physiological features - like heart rate, for instance - in normal cars. Finally, the data contains a mixture of continuous and discrete features. For example, 'P1' is continuous, while 'E9' is discrete, and has a range of 0 to 9. Instead of discretizing the data, we decided to assume all the features were roughly normally distributed for certain models, such as the Gaussian Naive Bayes.

2 Preliminary Processing

2.1 First Round: Feature Selection using 30 Features

Unsurprisingly, using all the features provided did not improve the performance of our Naive Bayes or linear classifier. Instead, we experimented with using subsets of the 30 features provided to us. To select the best features, we did two things. The first thing we did was to compare the distribution of each features when the driver was drowsy and when the driver was alert. In particular, we assumed that many of the features had a normal distribution, and that we could compare by simply calculating the difference in the mean of each feature divided by the total variance. This proved extremely useful in narrowing down which features to use; we tried to include features with a ratio of difference in mean to total variance greater than 0.2, and ignored the others, which narrowed the down the feature set to 8 features. We experimented with using different subsets of these features, naturally.



The second thing we tried was to feed the data through feature selection algorithms. The first was a univariate feature selection, which selects the best features based on univariate statistical tests. In particular, we used "SelectKBest" which is part of the scikit-learn package. We also used L1-based feature selection, which uses a transform method to reduce the dimensionality of the data by selecting the non-zero coefficients in the sparse solution. The culmination of all these feature selection algorithms is represented in the image above where we used ensembles of a decision tree (here we used an Extra Tree) to compute the relative importance of each attribute. The three attributes that were ranked to have the highest importance were the ones we used in order to get our lowest testing error. As you can see from the graph, there are certain stand out features that we used across different feature sets.

2.2 Second Round: Feature Selection using New Features

The data, as explained above, already contained 30 features about the driver's physiological, environmental, and vehicular behavior at each time step. We also chose to create features describing each feature over time: the minimum and maximum value, as well as the size of the range, of each feature over a given time span. Simply processing (not mention training any models on) this many features took an extraordinary amount of time and memory so we had to narrow down which features to use.

As with the first round features, we manually selected features by comparing the ratio of difference in means (between drowsy and alert observations) to total variance. This narrowed down the original 120 features to only 24 features, which was much more tractable.

2.3 Third Round: Feature Selection Dependent on Model

Although this is not a preprocessing step, we feel like it is important to discuss this last round of feature selection here. Both the first and second rounds of feature selection described above look at the data from a higher-level, and then attempt to create superior sets of features. However, we have found that different features perform better or worse depending on the model used. Hence, our third round of feature selection is an iterative process where we use intuition built from our first two rounds and create new sets of features that are adapted to each model.

3 Models and Results

Using the sets of features developed using our various feature selection techniques, we looked to train our learner on the Ford training dataset. To do this, we explored various models, including Stochastic Gradient Descent and Randomized Decision Trees. For each model, we trained the learner on 5 different feature sets:

- A basic set: this set was arrived at using our first round, where we looked at features with the largest difference in drowsy and alert mean to variance ratios. This set has 9 features, with 1 physiological, 4 vehicular and 4 environmental.
- Univariate set: this set has a total of only 3 features, where none are physiological (as desired by our challenges). Two of the features in this set overlap with the basic set.
- L1-based set: this feature selection method was less successful at picking out features and had a total of 27 features (only 3 eliminated).
- Time-change Features: these features were arrived at from our second round. There are a total of 24 features.
- Method-dependent Set: inherent in the name, this set varied across models. For SGD, for example, our method-dependent set has 3 features (two environmental and 1 vehicular).

Training our learner on each set yielded some very striking results. For relevance, the table below only shows the best training and test errors and specifies which feature set yielded these results.

Method	Feature selection	Train Error	Test Error	Train False Pos.	Test False Pos.
SGD w/ Hinge Loss	Model dependent	31.71	11.79	97	98.97
SGD w/ Logistic Loss	Model dependent	30.55	11.8	95.95	98.89
Naive Bayes	Time-change	22.94	25.03	82.58	34.78
Naive Bayes	L1-based	35.19	12.26	15.84	98.8
Randomized Trees	L1-based	0.006	25.38	42.5	36.85

Next, we will describe each model, its implementation and the relevant results.

3.1 Stochastic Gradient Descent

We implemented a standard stochastic gradient descent algorithm, using the features selected above. We experimented with two different losses: a hinge loss, and a logistic loss function. We eventually used an implementation provided in the scikit library.

Early on, we found that Stochastic Gradient Descent on the hinge-loss function showed a lot of promise. As a result, we iterated a lot on this model using parameter tuning and choosing our features carefully. Our best results came with a set of 3 with features 'E8', 'E9' and 'V11'. Our test error was a highly satisfying 11.79% - the lowest test error in all our results. However, the learner developed had a surprisingly high training error of 31.71%. This is strikingly high, and is related to the distribution of both training and testing data sets: in training data, there were more drowsy outputs and in test data there were less. As a result, our predictor predicted alert for a very large percentage of samples and this led to a high number of false positives (predicting alert but driver is in fact drowsy). In fact, out of all our wrong predictions, 98.97% were false positives. These high false-positive rates occurred across different feature sets. In our analysis, we will delve into this further. When we used a logistic regression instead, results were indeed very similar; test error was higher (11.8%) but training error was lower (30.55%).

3.2 Naive Bayes Classifier

The Naïve Bayes model assumes that all 24 features (selected as described above) are dependent on the alert or drowsy variable, and are independent of each other. The model was trained by calculating the distribution of each feature (separated by alert or drowsy observations). A major assumption that we made because most of the features were continuous was that the distribution of each feature was normal. So, the distribution of each feature was described by its mean and variance, and the probability of any given value of a feature was calculated using the probability density function for that features' normal distribution. Classifying was done by simply choosing the assignment (drowsy vs. alert) which had the higher probability of giving the observed values of the features.

Naive Bayes offers an alternate way of learning. Unlike the surprising results found in SGD - with very high false positive rates across different feature sets - Naive Bayes had varying rates with different feature sets. Our best test error occurred using our L1-based feature set. This was generally surprising because this set contained a total of 27 features, and we expected overfitting. Our test error on L1-based feature set was 12.26% with once again a high training error of 35.19%. A more balanced result came using our Time-change feature set which

yielded a test error of 25.04% and a training error of 22.94%. For the L1-based set, there were high false-positive rates (98.8%) whereas in the time-change feature set the false-negative rate was high (65.22%).

3.3 Ensemble Methods: Randomized Decision Trees

Using the scikit-learn package, we were able to implement an averaging algorithm based on randomized decision trees called the RandomForest algorithm. This algorithm creates a diverse set of classifiers by introducing randomness in the classifier construction. Then, prediction is done by averaging the prediciton of the individual classifiers. This has the benefit of reducing the variance of the base estimator and thereby reducing overfitting.

Unlike other models, these decision trees yielded very low training error. This is a characteristic of this model because the learner can create overly complex trees that do not generalize the data well - effectively overfitting. Low training error, however, did not translate into low testing error. The best performing randomized tree occurred on time-change feature set with 16.83% test error and a very low 0.24% training error. False-positive and negative rates in this model were more evenly distributed.

3.4 Neural Networks

We were actually very excited about implementing neural networks. The set up of our problem, in particular the unlabeled features, made the concept of using neural networks extremely attractive to us. In addition to this, we always suspected that if we combined some of our features together, we would get a more realistic model of the situation that we were trying to mimic, while we could not use our intuition to put these features together, the promise of neural networks was alluring and hence we implemented it with great anticipation. The results however were less than satisfactory. With a 34.8% training error and a 44.2% testing error, this was not sufficiently accurate to even make it to our summarized table of results. Retrospectively, we should have taken more time setting the initial weights vector for the algorithm but it was obviously hard without the knowledge of what each feature represented.

4 Analysis and Discussion

4.1 Distribution of Training and Test Data

Our high training error was something that was extremely alarming to us. At many points we were sure we were implementing our algorithms wrong in some way so after triple checking each line of our code to ensure correctness, we were curious as to why this was the case. We tried many different learning algorithms and often received a similar anomalous pattern: our training error was at times up to a factor of three higher than our test error.

We decided to splice the dataset that we were using as our training set into two parts, one that would continue to be used as training data while the other would be used as test data in place of the dataset we were given. We finally started getting more expected results where the testing error and training error were similar. After further statistical analysis of the training and testing datasets, we realised that these had a significantly different distribution from one another: training had less alert outputs than testing.

This was a relief for us in multiple ways. Apart from assuring us that our algorithms were indeed not to blame, it also explained why our feature extraction, and subsequent testing on our models, showed that the best results achieved were when we used very few of the features we had available to us. As mentioned, we received the best results with only three out of the thirty features. This can be explained by the difference in distribution. The more features we used, the more our model was fitting to the distribution of the training set which of course was very different from that of the testing dataset.

4.2 False-Positive and False-Negative Rates

Consider the following table that describes our findings under the lens of our wrong predictions.

Method	Feature selection	Train False Pos.	Train False Neg.	Test False Pos.	Test False Neg.
SGD w/ Hinge Loss	Model dependent	97	2.99	98.97	1.02
SGD w/ Logistic Loss	Model dependent	95.95	4.05	98.89	1.11
Naive Bayes	Time-change	82.58	17.42	34.78	65.22
Naive Bayes	L1-based	15.84	84.15	98.8	1.18
Randomized Trees	L1-based	42.5	57.5	36.85	63.14

In our search for the reason behind the appalling performance of our training set to the models we were creating, we started looking at where the errors were coming from. Were we predicting drivers as drowsy when they were actually alert or vice versa? In the above table, we determine false positive and false negatives for each of our models. A false positive rate is defined as the percentage of errors that were predicted as alert but were actually drowsy. Similarly, false negative rate is the percentage of errors that were predicted drowsy but were in fact alert. What we found was while looking at linear classification with both loss models, we were achieving a similar rate of false positives and false negatives. Upon further probing we realised that the training set indeed had a much lower proportion of positive labels. In fact, the training data had fit the model in a way that it would almost always predict that the driver was alert. This of course was advantageous to the test data where around 75% of the examples represented an alert driver. Hence, we see that our predictor yields high false positives because it often predicts 'alert'. This leads to a high training error because the training data only has 54% of samples as alert.

4.3 Usefulness of time-variation features

After our first experiments with the regular feature set, we were disappointed by the high training and testing error of our models, especially with stochastic gradient descent with hinge loss and logistic loss functions. At the time, we weren't sure why the linear classifier did poorly, and we were even more surprised that the physiological features were bad at predicting whether the driver was alert. We expected, when we began the project, that physiological features would be the most effective at predicting driver alertness, because intuitively, alert people have higher heart rates (for example) than drowsy people.

One explanation for the poor predicting power of physiological features was that the "normal" value of each feature might vary drastically depending on the features; for example, one person's resting heart rate might be a high heart rate for someone else. So, instead of using the instantaneous value of each feature, we decided to include other features, such as the minimum, maximum, and total range of each feature over a certain time frame, usually 5 seconds. We expected the range of a feature over a time period to be especially useful, since intuitively, a change in alertness in a driver should reflect a greater variation in physiological features over time.

Surprisingly, then, the time-variation features were not extremely useful. After running the same analysis on these time-variation features as on the original features, we found that only a few differed significantly between alert and drowsy observations. Even more surprisingly, the features were significant were not physiological features: the useful range features were 'E2', 'E5', 'E7', 'E8', and 'V3', and both 'E7' and 'E8' were already present in the set of normal useful features. Similarly, the useful minimum features were 'V10', 'E6', 'E9' (only 'E6' was not in the set of normal features), and the useful maximum features were 'E1', 'E2', 'E3', 'E8', 'E9', 'V6', and 'V8' (only 'E2', 'E3', and 'V8' were not in the set of normal features). So, perhaps it is not surprising that using these time-variation features did not affect the training and testing error very much.

However, the time-variation features did help in some cases. For example, our best Naive-Bayes model using only features in the original dataset only achieved 30% training error and 24.3% testing error, while the using time-variation features reduced the error to 22.9% training error, while the testing error stayed the same, 25.0%.

5 Future Direction

Another model we would have liked to explore, with a lot more time, would have been a hidden markov model which calculates the probability of transitioning from alert to drowsy or vice versa, given the value of the features. This model would have been interesting because we had several positive results with our time-change features. Furthermore, although we have already tried, we plan on contacting Ford so that they can give us labels to the features. One focus of ours was to create the classifier so that it can run in real-time - that is why we tried to ignore phsyiological features as much as possible. In a car environment, the data passed to the learner would have labels for the features. For example, data about the speed of the vehicle would be labeled "Vehicle Speed." It would be very interesting to be able to see what the features actually referred to - especially our optimal feature set of 'E8', 'E9' and 'V11' for SGD. With this intuition, perhaps we could better combine features leading to superior predictions.

6 References

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011. "Stochastic Gradient Descent.Ẅikipedia. Wikimedia Foundation, 12 Nov. 2014. Web. 18 Nov. 2014. "Naive Bayes Classifier.Ẅikipedia. Wikimedia Foundation, 12 Nov. 2014. Web. 19 Nov. 2014.