Stay Alert! The Ford Challenge

Tayyab bin Tariq, Allen Chen
CS 229 Fall 2012

Problem Description

Driving while drowsy, intoxicated, or otherwise distracted can have life changing consequences. Activities that divert the driver's attention from the road ahead, such as engaging in a conversation with other passengers in the car, making or receiving phone calls, sending or receiving text messages, eating while driving or events outside the car may cause driver distraction. Fatigue and drowsiness can result from driving long hours or from lack of sleep. Given a set of physiological, environmental and vehicular features taken from real world testing, the goal of our project was to predict whether or not the driver of a car is alert. The units and source of these features have been removed, therefore preventing us from using any domain and/or feature specific methods to the problem, and presenting a binary classification problem. Setting the top leaderboard scores on Kaggle as our benchmark, we sought to create a competitive classifier using the techniques we have learned in CS 229.

For this competition, Kaggle scored test results based on Receiver Operating Characteristic Area Under Curve (ROC AUC) [1]. This rewards a high $\frac{true\ positive\ rate}{false\ positive\ rate}$ratio, which is important for driving situations, since a system that triggers many false positives will likely be very intrusive to the driver, or simply turned off altogether. For classifier development, we used F1 score and assumed that it was a good proxy of the AUC score on the training set [2].

Data Description

We obtained the data for this problem from the Kaggle challenge website [3]. The website provides two data files - one for training and one for testing. The training data file (hereafter called S_{train}) consists of 604,329 data points. Each data point belongs to a one of 500 separate trials. The test file (hereafter called S_{test}) contains 120,840 data points belonging to 100 trials. While each data point in S_{train} comes with a labels in {0,1}, S_{test} has no labels. The intention is that we test our classifier on the test data and submit our predictions via Kaggle's online submission process. Each sample contains 8 physiological, 12 environmental and 10 vehicular features. As previously mentioned, the actual identity and units of the features has been stripped away from the data, making it more difficult to intuit which features might be most relevant to classification.

For classifier development, we held back 20% of S_{train}, hereby referred to as S_{dev}, in order to evaluate our models before submission to Kaggle for scoring on S_{test}. However, we soon found that when we tried to train Neural Networks, SVMs and random forests on this data, the turnaround time required to perform substantial testing (i.e. for parameter search) was too long given the project timeline, or we just ran out of available memory. As a workaround, we selected the first 100,000 measurements from S_{train} and divided them into S_{train}' and S_{dev}', again using an 80/20 split. Once we had tuned our final model, we trained it S_{test}. In the following sections all the results are from training on S_{train}' and testing on S_{dev}' unless otherwise stated.

Baseline Methods

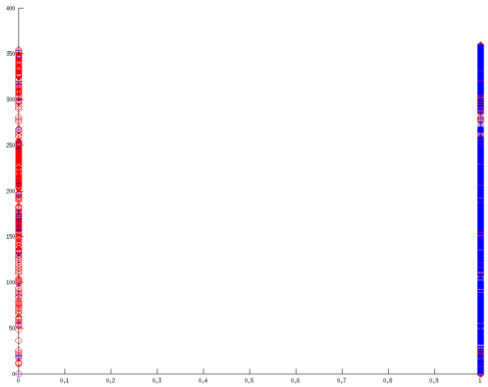
For the milestone, we explored three methods to establish baseline performance upon which to build during the rest of the project.

Note that the Kaggle score is calculated based on the area under the Receiver Operating Curve (ROC), referred to from here on as AUC.

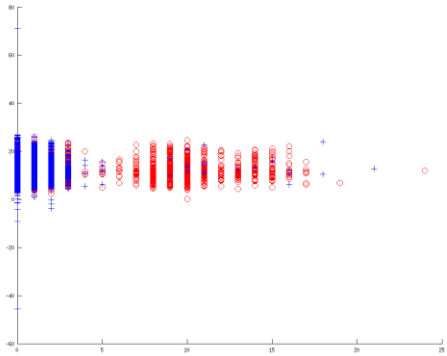
Method	Description	F1 on S_{train}	Kaggle score (AUC)
Linear decision boundary on single feature	See “Plotting Data” section below	77.21	57.725
Logistic regression	Used glm in MATLAB to run logistic regression on S_{train}	97.06	72.25
SVM	Liblinear package, linear kernel	96.99	72.747
Naive Bayes	Discretized each feature across all training samples, using 100 bins	84.61	69.51

Plotting Data

We plotted the data to look for interesting patterns. We plotted each of the 30 variables as a scatter plot against the labels and also plotted combinations of variables. We noted a couple of interesting patterns. We noted that whenever the E7 (environmental feature 7) was greater than 5.5 the labels were mostly negative. We used this intuition to come with the simplest possible classifier and that was just 1{E7 < 5.5}. This extremely simple classifier gave us an F1/P/R of 77.21/63.06/99.55 on the S and AUC of .5775 on the test set (as evaluated by Kaggle’s online evaluation system). This is a very reasonable result for a classifier as simple as a comparison operator. We also noted interesting shapes behavior in other variables. For instance P3 was strongly correlated with an alert label whenever it was an even multiple of 100. The same behavior was noted by other Kaggle participants [NT3]. We also noted that P5 looked largely bimodal. Later on we use these intuitions to transform the data to get better results.



Feature 17 vs Feature 10



Feature 15 vs Feature 2

Naive Bayes

For Naive Bayes, we discretized each column into a number of bins, defined globally. We then carried out a multinomial model for binary classification similar to the one used on problem set 2 for spam email classification, with Laplace smoothing applied. We saw optimal performance around 100 bins.

Logistic Regression

We then moved on to use a logistic regression as a means of predicting the driver's alertness. We trained the model on S_{train}' and used S_{dev}' as the validation set. This gave us an F1/P/R of 97.06/94.3/100 on the validation set and an AUC of 0.7225 on S_{test}.

SVMs

We also tried using SVMs to build a classifier for driver alertness. As a first attempt, we used the raw data and trained a simple SVM using a linear kernel with L1 regularization to make predictions. We trained it on S_{train}' and tested it on S_{dev}' to get a F1/P/R of 96.99/94.18/99.98 and an AUC of .7274 on S_{test} as evaluated by the Kaggle online evaluation system.

Neural Networks

The first more advanced method we applied was neural networks. Using a feedforward neural network with two hidden layers of size 30 and 100 nodes, we obtained an F1 score of 79.77 and a Kaggle AUC of 76.513 while training on S_{test}'. These results caused us to focus our efforts on trying to improve neural network performance through tweaking parameters and various other methods.

We made several observations regarding the data that led us to apply various transforms on the training set. First, we noted that the individual observations were on widely varying scales, which led us to believe that scaling the data to a standard range [0,1] would both improve performance and reduce possibilities of computational error (i.e. failure of iterative methods to converge consistently). While this was not the case, when we attempted to scale data on a trial by trial basis, we saw improved results compared to scaling based on all trials combined. This suggests that the trials are not drawn from the same distribution. We address this fact later in the paper.

We also tried running neural network on a binned version of S_{train}', using 10, 50 and 100 bins. A larger number of bins decreased performance, with 10 bins seeing performance slightly below our initial runs. However, discretization caused our training error to fall dramatically, producing F1 scores of over 96 percent, indicating overfitting to the training set. We also attempted less complicated neural networks, using a single hidden layer with 15 to 30 nodes. Again, we witnessed that training error decreased, while test error increased compared to the initial settings.

Due to the dynamic nature of the problem, one approach that we tried while working with neural networks was using a measure of change in the features as a feature. For example, we created a feature that represented the difference between the current value of each feature and the mean of the previous 100 values. However, this did not seem to increase performance above the baseline.

Random Forests

Another type of classifier we attempted to use was Random Forest. We used the randomForest package in R to see if it would offer improved AUC performance on the test data. Similar to neural networks, random forests are quite computationally intensive to train, so we used S_{train}' and S_{dev}'. Again, we saw scores on S_{dev}' of above 90 percent, but the Kaggle AUC results were around 76 to 76.5 percent, very similar to the neural network. Experimenting with both the node size and number of trees in the forest did not yield significant improvements, and attempting to train on a larger subset of the training data required too much memory.

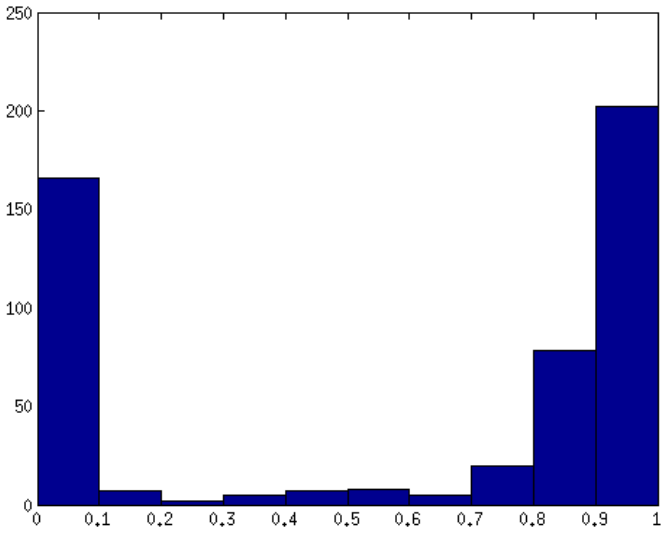
Further Data Analysis

As we proceeded with more advanced techniques beyond our baseline analyses, we saw a pattern of our algorithms returning a much higher AUC when run on the held back subset of the training data compared with the AUC we were receiving on S_{test}. This implies that S_{test} and S_{dev} are drawn from different distributions, and therefore we should extrapolate using a simpler model. We decided to step

back from the more advanced “black box” techniques and try a simpler approach, using logistic regression and selecting a smaller subset of features to avoid overfitting.

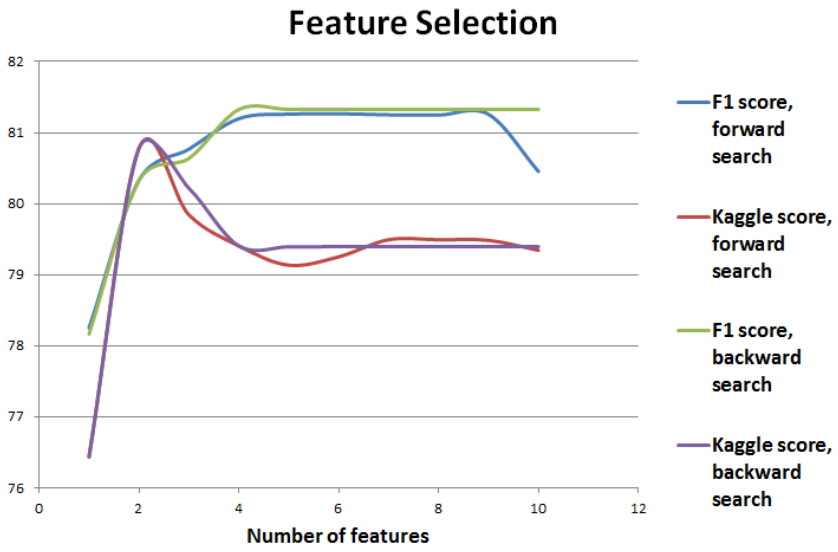
An analysis of the mean isAlert value across trials reveals that there is a bimodal distribution (i.e. in each trial, the driver is mostly alert or mostly not alert):

Histogram of mean driver alertness per trial



This observation suggests that we should use aggregate data from each trial as an additional feature. We chose to add the mean and standard deviation across all observations of a trial as additional features, expanding our feature space from 30 dimensions to 90. Furthermore, we tossed out correlated variables, such as P7 (correlated with P6) and P4 (correlated with P3), transformed P5 into a 1/0 indicator variable, and discretized E7 and E8 into 4 bins, following suggestions from competitors on the Kaggle forums[4]. (Note: “P” = physiological feature, “E” = environmental, “V” = vehicular)

We then carried out both forward and backward search, using logistic regression F1 score and Kaggle score to optimize the number of features:



Results and Discussion

Method	**S_{dev} F1 score**	**S_{test} AUC (via Kaggle)**
Logistic regression	97.06	72.25
SVM	96.99	72.747
Naïve Bayes	84.61	75.665
Neural Network, 2 layers	79.77	76.513
Neural Network (10 bins)	96.35	75.703
Random Forest, 200 trees	90.41	76.118
Logistic regression with feature selection	80.767	80.779

Over the course of the project, we found that black box methods came with several disadvantages - parameter tuning took a long time, especially since training on large data sets with neural networks and random forests is computationally intensive. Random forest also has the disadvantage of not converging to a certain result with each run, which means we saw variation in AUC results of around 1 percent between runs on the same training and test sets. Furthermore, increasing the complexity of neural networks and random forest consistently overfit the test set.

Feature selection showed that maximum AUC on S_{test} appeared to occur at just 2 or 3 features using logistic regression (which provided the best results). This is likely due to the fact that S_{train} and S_{test} differ in some way, as seemed to be the consensus among other Kaggle competitors. Also note that logistic regression with feature selection results in F1 and AUC scores that are very close, suggesting that we are *not* overfitting the data.

In the end, our simple logistic regression using 2 or 3 features selected via forward search presented the best results, with AUC around **81 percent** and putting us in the **top 15** on the Kaggle leaderboard.

One issue that we did not address was that our mean and standard deviation features were taken from all data across a trial, meaning it could not run in real time as a true driver alertness application would. However, implementation of an online learning version would be simple, and results from other Kaggle competitors suggest a very small difference in resulting AUC [5].

Furthermore, because of the way AUC is calculated, we found that the Kaggle scoring system consistently gave higher scores when we submitted a continuous range of values from classifiers that supported it, e.g. logistic regression.

[1] http://en.wikipedia.org/wiki/Receiver_operating_characteristic
[2] http://en.wikipedia.org/wiki/F1_score
[3] http://www.kaggle.com/c/stayalert/data
[4] http://www.kaggle.com/c/stayalert/forums/t/328/methods-tips-from-non-top-3-participants/8695#post8695
[5] http://blog.kaggle.com/wp-content/uploads/2011/03/ford.pdf