Detecting Distracted Driving with Neural Nets

Joseph Blubaugh

30 October 2016

Overview

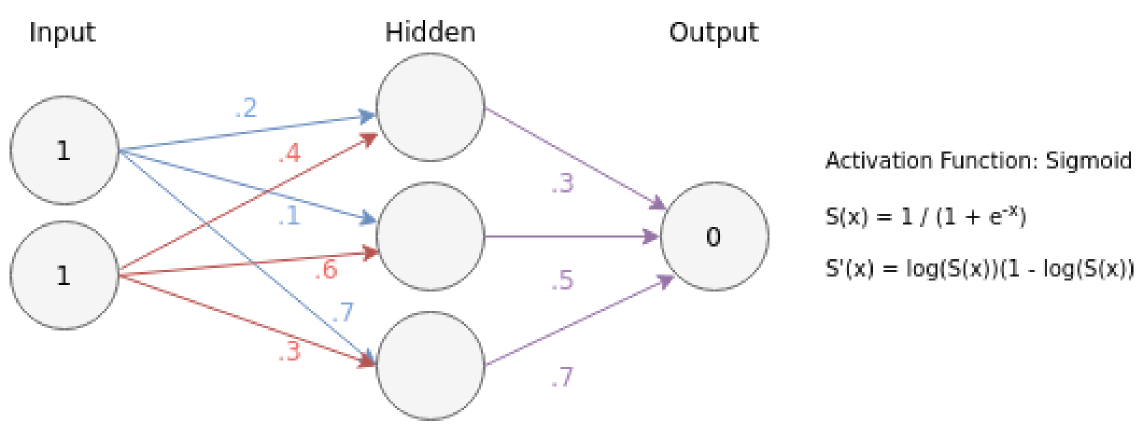
The first pass analysis used summarized statistical measures to fit both a random forest and neural network model to the data. The results primarily showed that aggregating the data over an entire event prevented the two algorithms from detecting texting events. Also, because the data was aggregated, there were very few data points to model between testing and training sets. The random forest algorithm was a better choice than the neural network because of the large number of predictor variables relative to the number of observations. Both models however, were guilty of overfitting.

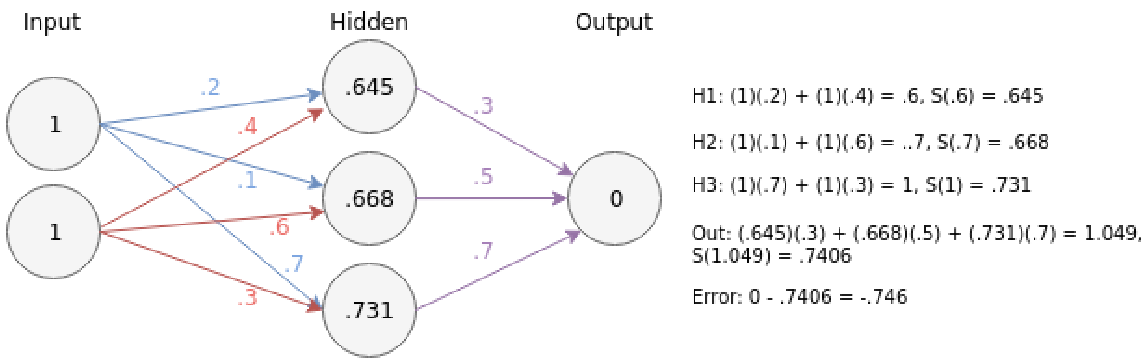
For the next phase of this analysis, I am focussing on looking at the data in its original form. I have chosen to focus on the neural network rather than the random forest mainly because of the data structure. In the previous analysis the aggregated data set contained 119 observations with 40 predictor variables. Since I am using the original data with no statistical measure the data set contains more than 1 million rows on 12 predictor variables.

Basic Neural Net

Neural networks are a class of statistical learning models that are well suited for large datasets of continuous variables. They are a essentially a series of interconnected nodes and weights that iterate on training data to update weights and minimize training errors. There are several types of neural nets and the type that I have used so far is the feed forward neural net, meaning that data is fed into the model through the input nodes, through the hidden layers, to the output nodes. Other more complex models may have a recursive function that feeds errors back through the model.

The following diagrams are of a basic neural net with 2 predictor variables (input), 3 hidden nodes, and 1 output node. Hidden layer weights are calculated using the sum product of the input values and the starting weights which are chosen randomly.





Neural networks require an activation function to transform the input into the appropriate output. In the case of this analysis I am trying to predict texting events so I only want predictions between 0 and 1. An appropriate activation function for this is sigmoid (S(x)).

Weights are initially chosen randomly so neural nets have an iterative process to minimize error by feeding the training error back through the model. The change for each iteration is calculated by multiplying the first derivative of the activation function by the most recent training error.

The following steps complete the update for a single training iteration:

```
# Calculate the weight change for the weights between the hidden nodes and output node:
# S'(1.049) = S(1.049)(1 - S(1.049)) * -.7406 = -.1619

# Update the Weights between the Hidden and Output Nodes:
# (-.1619) / [.645, .668, .732] = [-.251, -.242, -.221]

# W7: .645 - .251 = .394
# W8: .668 - .242 = .426
# W9: .731 - .221 = .510

# Calculate the change of the hidden weights
# Divide the weight change by the original [w7, w8, w9] weights times S'(w3, w4, w5)
# -.1619 / [.3, .5, .7] * S'([.6, .7, .1]) = [-.1234, -.0717, -.0454]

# Update the weights between the input nodes and hidden nodes
# [-.1234, -.0717, -.0454] / [1, 1]
```

The diagram below shows the updated neural net for one iteration of the training cycle. It shows that the updated error is slightly smaller than the original training error using random weights.

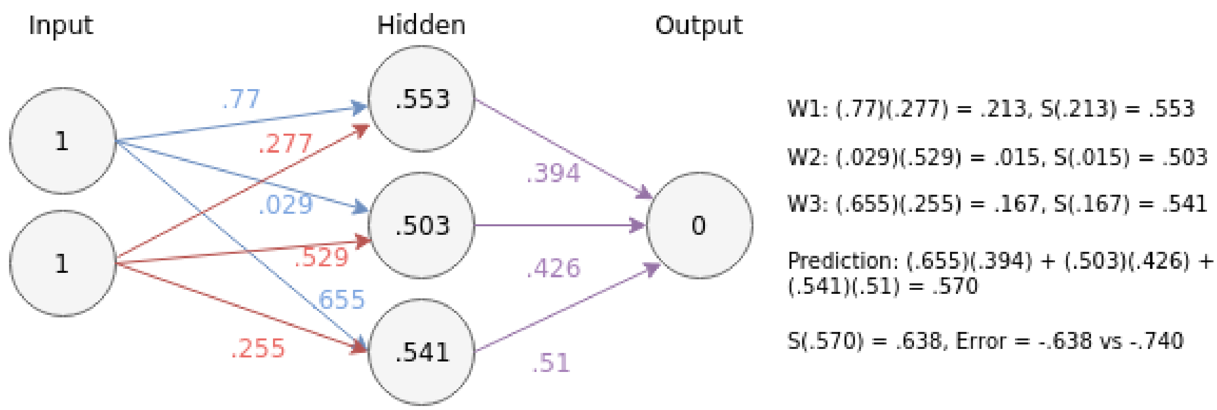


Figure 1: Hidden Weights

Neural Nets applied to Distracted Driving

In order to standardize the results of my analysis, I have elected to use the R package caret to train all of my models (http://topepo.github.io/caret/index.html). Caret is a modeling framework that allows you to run many different types of models with different combinations of parameter sets at the same time. It can easily use models from other popular packages and offers a rich set of validation tests and diagnostic plots. Another benefit to using this package is enabling parallel computing of cross validation tasks.

I am using k=10 cross validation for all models as well as several combinations of paramters. Training and Testing sets are all approximately the same size 50/50 split. Each model run

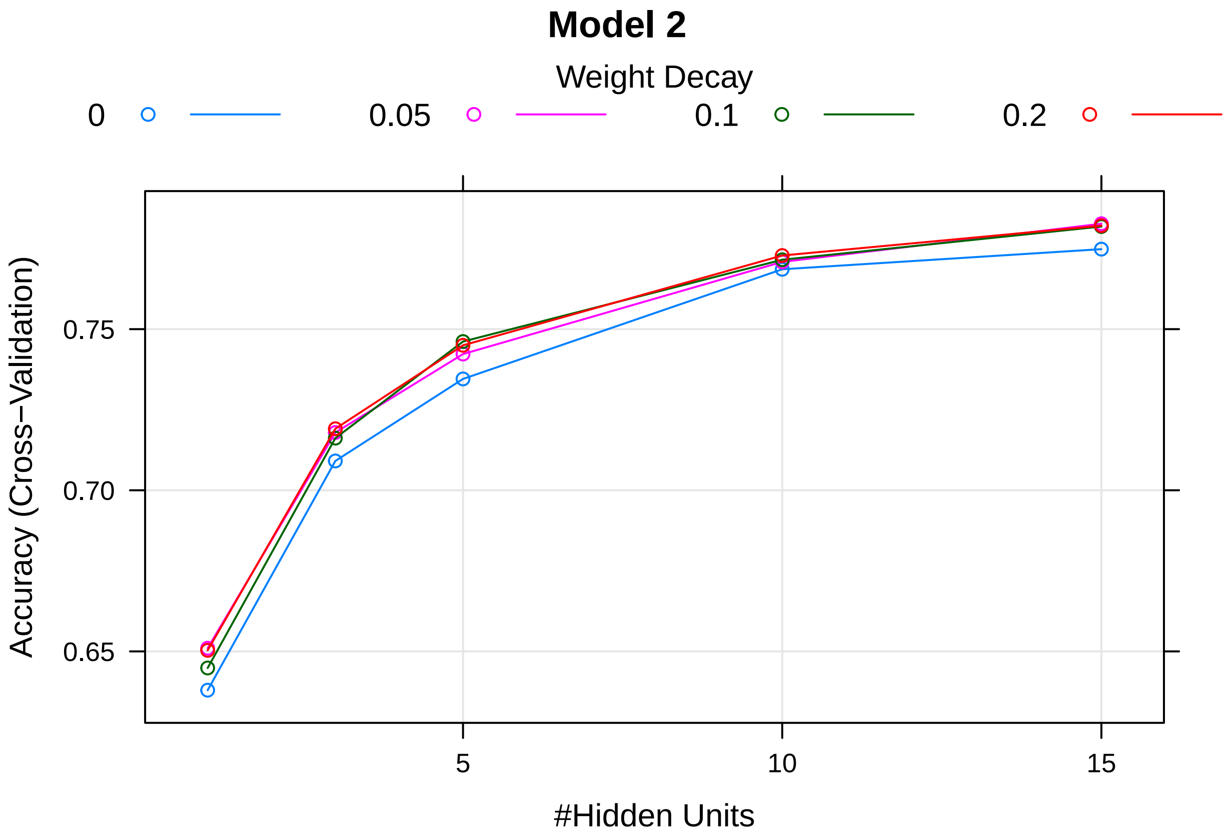
Current Results:

Model	Data	Testing	Training
Model 1:	Training and Testing data split at the 365 second	.78	.69
Model 2:	Training set was sampled from the entire simulation	.75	.75
Model 3:	Differencing based on Model 1 training data split		
Model 4:	Random sample from differenced of the entire simulation		
Model 5:	Moving average of all predictors then split data at 365 seconds		
Model 6:	Differencing based on the moving average over entire simulation		

General Model Form: nnet(Texting ~ Subject + Age + Gender + Anger + Contempt + Disgust + Fear + Joy + Sad + Surprise + Neutral)

```
#####################################################################
## Code for running Model 2
## Compute time: 12 hours over 6 CPU

## Set Cross Validation
fit.control = trainControl(method = "cv", number = 10)

## Create combination of model parameters to train on
search.grid = expand.grid(decay = c(0, .05, .1, .2),
                          size = c(1, 3, 5, 10, 15))

## Limit the iterations and weights each model can run
maxIt = 500; maxWt = 10000

fit = train(Texting ~ . - Time, mdl.02.train, method = "nnet",
            trControl = fit.control,
            tuneGrid = search.grid,
            MaxNWts = maxWt,
            maxit = maxIt)

Neural Network

542550 samples
12 predictors
2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 488295, 488296, 488295, 488294, 488296, 488295, ...
Resampling results across tuning parameters:

decay  size  Accuracy   Kappa
0.00    1    0.6379560  0.2186901
0.00    3    0.7090977  0.3627473
0.00    5    0.7345479  0.4244812
0.00   10    0.7685799  0.5083622
0.00   15    0.7748299  0.5226515
0.05    1    0.6509813  0.2211743
0.05    3    0.7179190  0.3914567
0.05    5    0.7422744  0.4481362
0.05   10    0.7708690  0.5133975
0.05   15    0.7827131  0.5405813 ** Best Model **
0.10    1    0.6448476  0.2514474
0.10    3    0.7161829  0.3820061
0.10    5    0.7461580  0.4562864
0.10   10    0.7715399  0.5151954
0.10   15    0.7818469  0.5390179
0.20    1    0.6503197  0.2436473
```

```
0.20    3     0.7190932  0.3910539
0.20    5     0.7449894  0.4524052
0.20   10     0.7728504  0.5181559
0.20   15     0.7821215  0.5392592
```




```
Loading required package: nnet

-------------------------------------------------------------------

Neural Network Confusion Matrix

--------------------------------------------------------------------

      Predicted
Actual      0      1
     0 281242  39644
     1  82515 139149

(Training Set) Overall Performance: 0.752101

      Predicted
Actual      0      1
     0 280817  39752
     1  82577 139405

(Testing Set) Neural Net Overall Performance 0.7519984
```

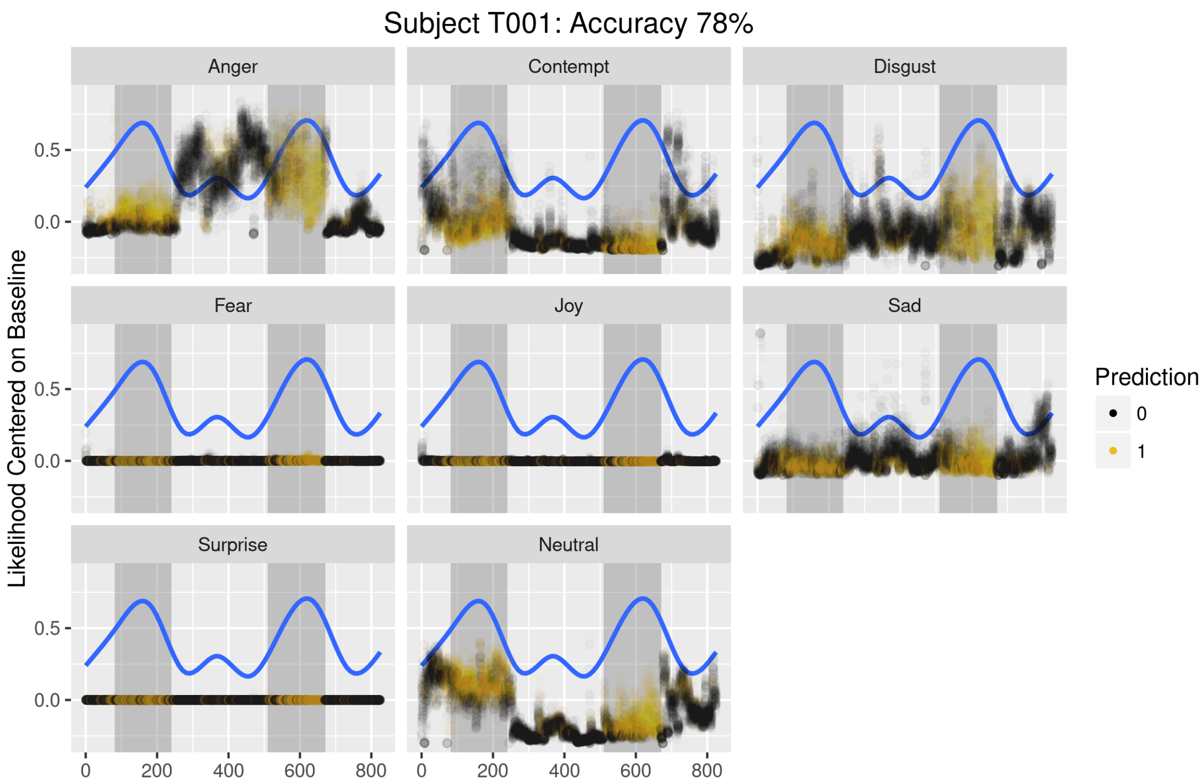


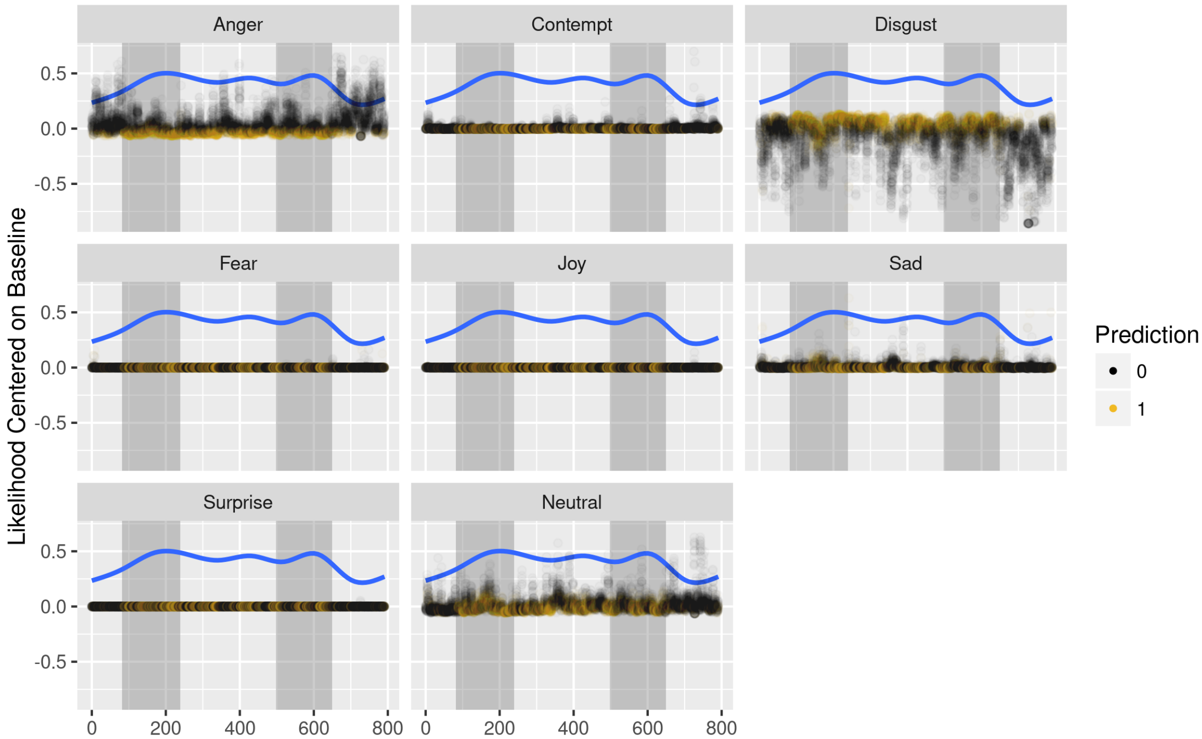
Figure 2:

Plots

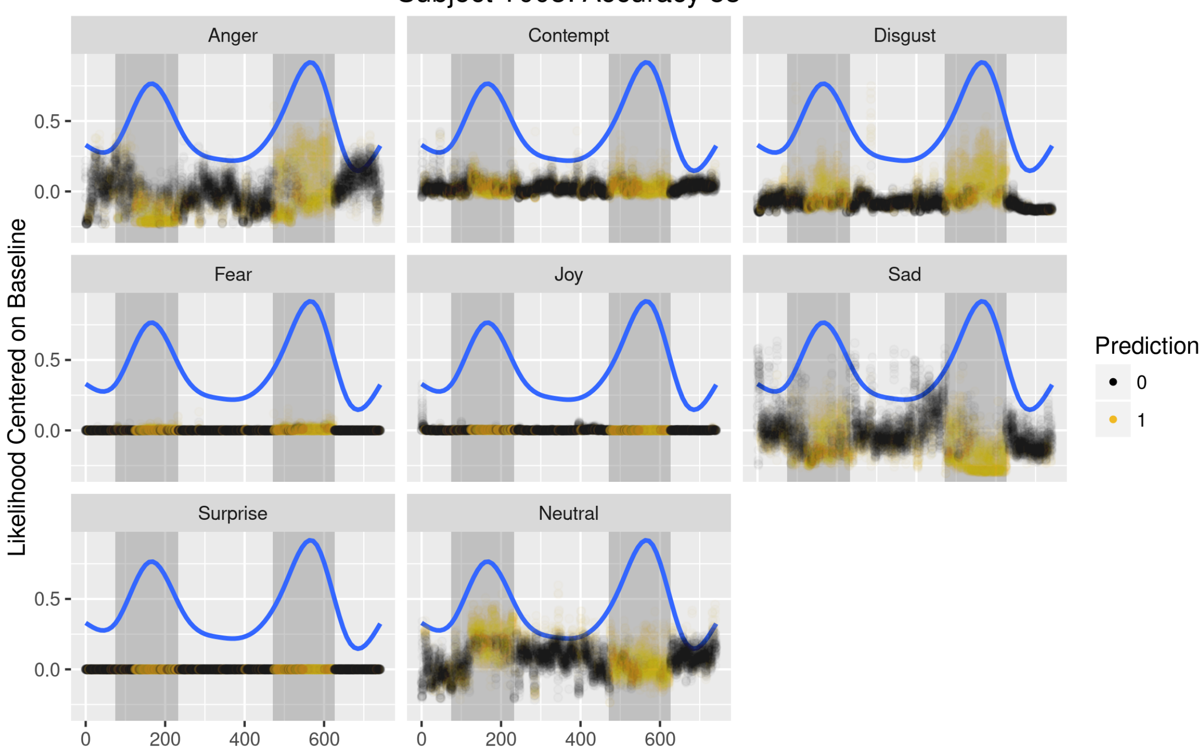
The following 3 plots overlay the prediction for model 2 against the test sets for the first 3 subjects. Yellow dots represent predictions for Texting events and the black dots are no texting events. The blue curve is a loess curve for the probability prediction. It shows the overall trend for the predictions by time. It is encouraging that for Subject 001, the probability tends to peak during the periods of texting which are highlighted in gray in the background.

Just as interesting though is the plot for Subject 002 where the model has a horrible fit overall and only predicts texting correctly about 60% of the time. Subject 003 has an excellent fit overall with a large majority of positive predictions occurring during the texting event window.

Subject T002: Accuracy 60%



Subject T003: Accuracy 83



Accuracy by Subject:

Table 2: Table continues below

Train	0.94	0.929	0.918	0.921	0.911	0.916	0.89	0.877	0.853	0.856	0.843
Test	0.943	0.93	0.926	0.92	0.92	0.903	0.894	0.878	0.862	0.861	0.837

Table 3: Table continues below

Train	0.833	0.827	0.834	0.832	0.824	0.804	0.812	0.815	0.815	0.804	0.803
Test	0.833	0.831	0.831	0.828	0.82	0.817	0.815	0.815	0.813	0.812	0.805

Table 4: Table continues below

Train	0.802	0.795	0.785	0.776	0.784	0.771	0.773	0.776	0.776	0.769	0.767
Test	0.802	0.801	0.787	0.786	0.785	0.779	0.779	0.776	0.773	0.768	0.765

Table 5: Table continues below

Train	0.754	0.753	0.742	0.75	0.736	0.731	0.745	0.738	0.726	0.709	0.704
Test	0.758	0.753	0.745	0.745	0.737	0.735	0.729	0.728	0.726	0.716	0.692

Table 6: Table continues below

Train	0.684	0.671	0.665	0.65	0.634	0.625	0.629	0.614	0.613	0.602	0.595
Test	0.681	0.669	0.664	0.656	0.634	0.629	0.618	0.616	0.609	0.597	0.594

Train	0.563	0.512	0.506	0.502
Test	0.563	0.512	0.504	0.503

Conclusions and Next Steps

So far the neural net models have shown encouraging results in the ability to detect texting events for some subjects based on facial expressions alone. It clearly helps model performance to consider each individual separately. In my next few sessions I would like to finsih building and testing the performance for the rest of the modeling scenarios I have proposed. I would also like to look into some additional neural net models that have recursive features. So far in the models I have trained I have ignored the time component which might be valuable.