Zach Fontenot

Machine Learning

Dr. Kawsar

April 22, 2020

Prediction Model Generator Framework: Phase II

1. Introduction

Phase two of development on the Prediction Model Generator focused on modularity, then expanded functionality. The goal for the final version of the Prediction Model Generator is a versatile tool that imports and cleans data, then produces and tests a prediction model with test data. In this phase, the first steps to achieving this goal are taken. The data used for testing during this phase comes in the form of pressure data from sensors located in a user’s shoe. The model generated from the pressure data then predicts which activity a user is performing. The processes of data cleaning and model generation will be detailed in this report, as well as measures taken to improve the accuracy of the model’s predictions.

1. Logistic Regression

Model generation was completed via logistic regression. The Numpy library allowed for simplified vector arithmetic, and the Scipy library provided the minimization function required for gradient descent. For multiclass logistic regression, a cost function, and one-versus-all wrapper over the minimization function were programmed. The cost function utilized the sigmoid function to determine whether each training example was part of a class or not part of that class. The Scipy “minimize” function took the data and the cost function as parameters and produced a model for each activity capable of calculating the likelihood that any given testing example fit into that class.

This initial model had a poor accuracy with the testing data (using a 70/30 split of training and testing data). Methods of improving the accuracy were brainstormed, implemented, and independently tested. These methods included better data cleaning, prediction bias, and elimination of the driving class.

1. Data Cleaning

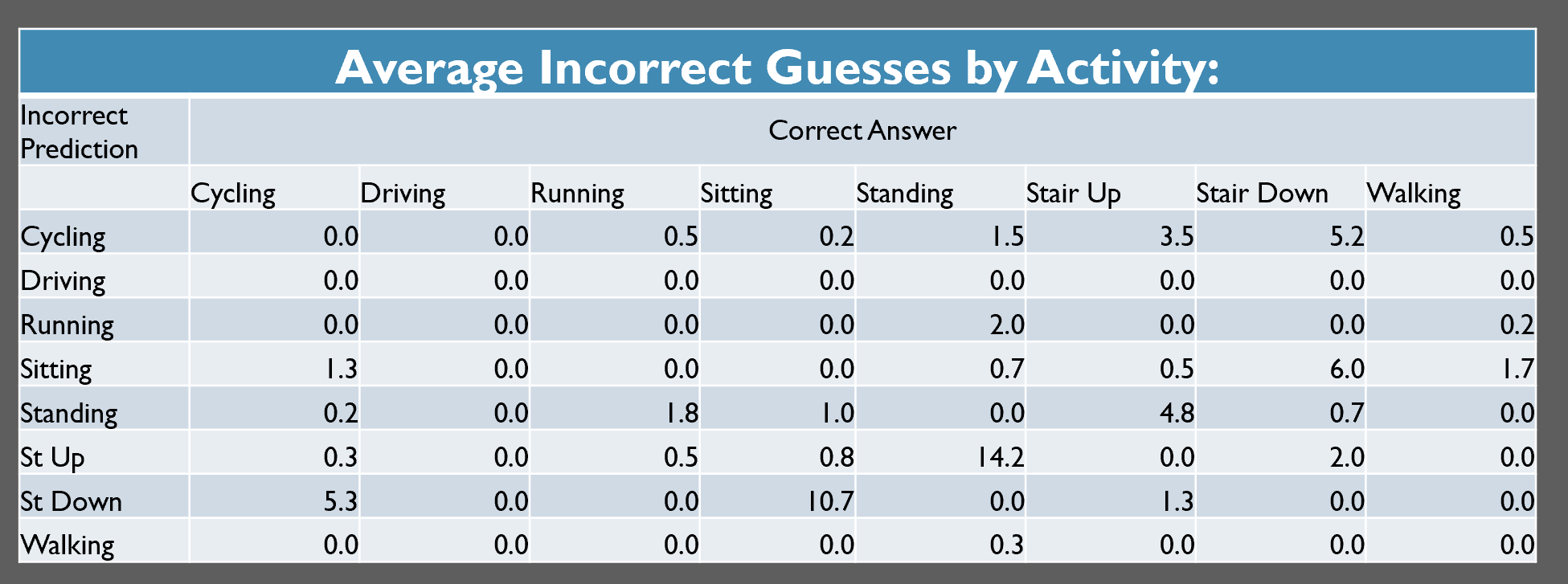
To better clean the data, outlier values were removed from each interval, ensuring that features were not generated using bad data. Outliers were defined by the IQR rule, a mathematical concept stating that any data point greater than the third quartile plus the interquartile range or less than the first quartile minus the interquartile range is an outlier. Originally, any two second interval of pressure data that contained an outlier was removed; however, this eliminated nearly every interval, as almost all of them had at least one outlier. Meanwhile, the approach of only eliminating outlier values and not entire intervals removed about 10% of the total dataset. Model accuracy improved by an average of 6% after ten test runs. The running activity was one of the few classes to see a decrease in accuracy, likely because the quick spikes of pressure with each step are seen as outliers to the cleaning algorithm. However, there was an overall improvement in model accuracy. Improvement derived from data cleaning via the IQR rule is visualized below in Figure 1.

*Figure 1*: The effect of removing outliers via the IQR rule can be seen on the prediction accuracy of each class e.g. cycling accuracy was 83% before applying the rule and 95% afterwards.

More work in data cleaning is certainly needed, as an extreme version of cleaning was performed on the data manually, where only the best datapoints were retained. The model that resulted from this “extreme cleaning” produced accuracies in the high 90% range, though it is guaranteed that this model is overfit to the smaller dataset. More testing will be needed to find the right amount of cleaning needed.

1. Feature Generation

New features added in this phase were designed to split apart commonly confused classes. In order to determine which classes were commonly confused, data was generating highlighting every wrong prediction made by the algorithm. This data was tabulated into the chart seen in Figure 2.





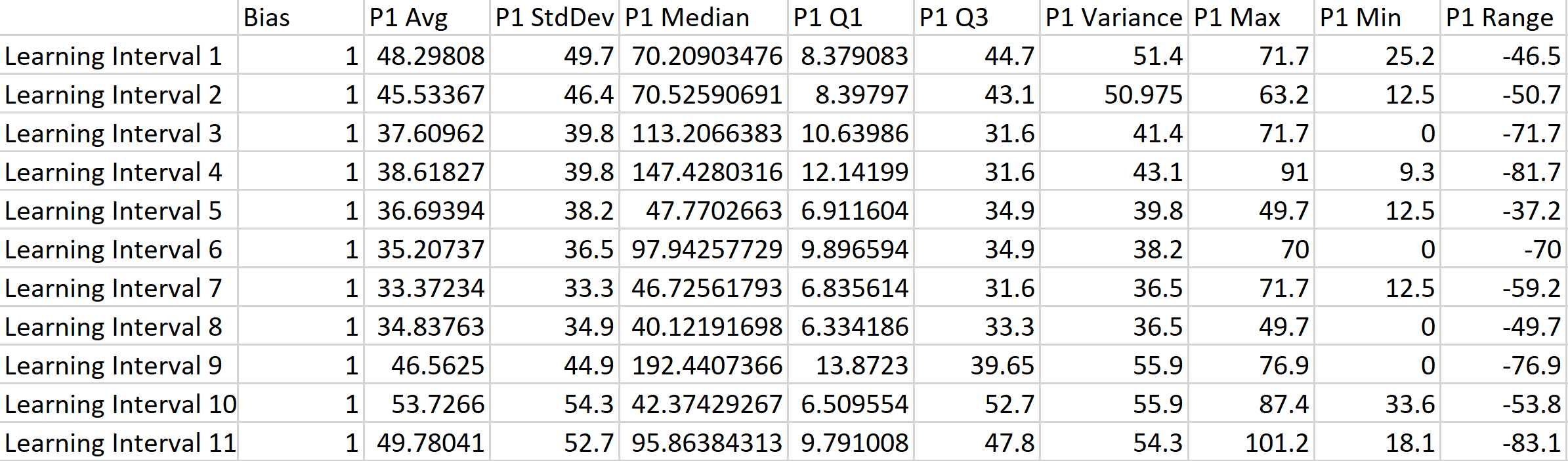
*Figure 2*: The average number of incorrect guesses for each class can be found e.g. on average, for cycling, there were 5.3 misclassifications of down stairs, 1.3 misclassifications of sitting, and the occasional prediction of up stairs and standing, as indicated by the highlighted values. The values are counts; therefore, this data can only be compared with testing sets of similar size.

As seen from Figure 2, in a group of trial runs, each with approximately four hundred test cases, a few classes commonly crossed over. This crossover resulted from similarities of these classes like, for instance, cycling and walking down stairs produced similar feature values. The answer to this dilemma was to create new features to better separate commonly confused classes. New features generated are listed below, but first the completely new sources of data should be explained.

By looking at documentation of the shoe used to gather this data, an observation was made: sensors were clustered in particular areas, notably the sole, heel, and ball of the foot. By grouping the sensors in each area and summing up the pressure data of each cluster, a total pressure applied to each area of the foot was obtained, e.g. sole pressure equals all the sensors in the sole added up. From this, new features were generated, as if the new data points were separate sensors. Something similar was also performed to the acceleration data, with the recorded acceleration of each dimension being combined into total acceleration via the Pythagorean theorem. From these new data points and the original sensors, the following features were calculated.

* Mean
* Median
* Standard deviation
* Variance
* First quartile
* Third quartile
* Maximum value
* Minimum value
* Range
* Interquartile range

The addition of the new features and datasets improved accuracy, but crossover between some classes still remained, showing that more feature engineering is needed. A current view of the generated features can be seen in Figure 3.

*Figure 3*: Above are the features generated from the first pressure sensor for the first eleven intervals. These rows would include the same features generated for sensors p2-p8; acceleration x, y & z; total acceleration; and heel, sole, and ball pressure data.

1. Elimination of Driving Class

Early models suffered from crossover of a few different classes. This crossover was especially problematic between the driving and sitting classes. The data was too similar for the model to consistently delineate a testing example. One possible explanation is the lack of data synchronization.

While the right foot is operating the pedals, it is generating pressure data that may be associated with driving. However, the left foot is merely resting, probably producing pressure readings more associated with sitting or standing, thus making it hard to determine the activity being performed. Synchronization could be utilized to solve for this, but at the cost of reducing the amount of training examples. Another solution was to combine the driving and sitting class into one, so that delineating between the two was no longer necessary. Combining the driving and sitting class increased accuracy by an average of 8% after ten test runs, as seen below in Figure 4.

*Figure 4:* The difference between accuracy of each category before and after combining the driving and sitting classes.

As seen from Figure 4, sitting accuracy jumped up significantly while other categories remained stagnant, showing that the combination of driving and sitting classes was beneficial.

Other classes could be combined to further increase accuracy. If the goal of the application utilizing the model is to track calories burnt, calorically similar activities could be combined. For instance, if research demonstrated that descending stairs and walking burned the same number of calories, combining the two classes would improve accuracy without sacrificing any functionality. Though if classes like driving are deemed essential to the application, implementing location services could help distinguish between driving and sitting with better accuracy.

1. Prediction Bias

One of the most important factors in predicting test cases came in the form of prediction bias. Prediction bias is defined as the model’s favorability to predict one class over another. This bias could be reliant on a number of factors, but all work the same way: The models predict a likelihood that the user is performing one of the seven activities based off of pressure and acceleration data. The most likely activity is then chosen. Bias increases the likelihood of predicting a particular activity based off of past behavior. Bias can be applied in several ways.

The type of bias implemented currently in predictions is the simplest form: if the user is performing some activity, *at*, for one two second interval of time *t*, then the next activity performed, *at+1*, is more likely to be the preceding activity, *at*. Simply stated, if the user is running for one second, he or she will probably be running the next second. This concept leads to the idea of continuation bias, adding weight to the previously predicted activity. This was applied to the prediction model and tested for by using a simulated set of continuous activities. The results show that continuation bias improved accuracy considerably. Different degrees of bias were tested for independently. The averages of these tests can be seen in the following chart, Figure 5, where a moderate continuation bias gives the best performance.

*Figure 5:* Demonstrating various degrees of continuation bias on accuracy. The blue bar indicates the best result, achieved with a moderate bias.

Another form of bias could track the user’s routines and apply bias where each activity is particularly common. For instance, if the user runs every day at 7 a.m., prediction bias could be applied to the running activity at this time every day. This sort of routine bias would be more difficult to implement, though, and could cause privacy concerns amongst users.

1. Regularization

In order to decrease the variance of the models, and decrease overfitting, regularization was implemented into the cost function. Regularization ensured that a model that worked extremely well for learning data may not be the model that produces the lowest cost; therefore, it ensured that unseen data was better represented by the model.

By scaling the lambda values up and down, varying degrees of regularization could be studied. The results showed that a low degree of regularization produced good learning dataset accuracy, but struggled classifying new cases. This demonstrates overfitting. On the other hand, high regularization produced poor values for both learning and testing datasets, or underfitted models. Finally, a lambda value between the two extremes generated models that predicted with strong accuracy for both learning and testing datasets. The data that allowed for these observations can be seen in Figure 7.

*Figure 7:*  This chart shows the effect of regularization on testing accuracy and learning accuracy. To the extreme left of the chart (lamda = 0), overfitting is observed. To the right (lamda > 0.5), underfitting is observed.

1. Conclusion

The Prediction Model Generator is beginning to fit the mold it was meant for: a flexible framework of modular machine learning tools. During the implementation of the aforementioned features and algorithms, refactoring the software to be modular and flexible was a constant goal. Though it is still only capable of generating models for activity prediction, with time spent on genericizing the existing functionality, eventually the framework will be capable of predicting a wide variety of classes and working with the user to provide him or her control over feature generation.

Future goals for the framework include improving model accuracy. A few routes to achieve this involve better algorithms. Support vector machines and neural networks are two potential candidates that may show improvement over logistic regression. Phase III will see the implementation of one of these algorithms, the implementation of more features, and the functionality to generate prediction models for a wide variety of data.