Columbia University
Alexandre RAME

Advanced Machine Learning Project

STACKED GRADIENT BOOSTING TREES

Abstract

We aimed at classifying whether a user is standing still, walking, running or biking by analyzing accelerometer data. We used Stochastic Gradient Boosting Trees to perform this classification. We then stacked different models to improve prediction accuracy. In this report we first explain how we computed relevant features from the raw data of the accelerometer obtained from a Smartphone GPS. We describe Gradient Boosting Trees and how we applied it to our classification problem. We explain how we can train new trees step by step using gradient-descent until the model is satisfactory. Then we show how we used Stochastic Gradient Boosting Trees to improve both the accuracy and speed of our model. Finally we go through several stacking methods and compare their results. We performed stacking by averaging the predictions of the classifiers but also using methods such as logistic regression and neural networks.

Contents

1 Data and Problems

1.1 Raw data from accelerometer

Our data-set consists of 6424 data-points corresponding to labels $L_i, i \in [1,4]$ describing whether the person is in one of the four categories {Still, Walking, Running, Biking}. Each data-point consists of acceleration measures in a three-axis-basis from the accelerometer: $(X_t, Y_t, Z_t), t \in [0,140[$. The curves below show the acceleration (X_t, Y_t, Z_t) (respectively in blue, green, and red) for one data-point corresponding to a person running.

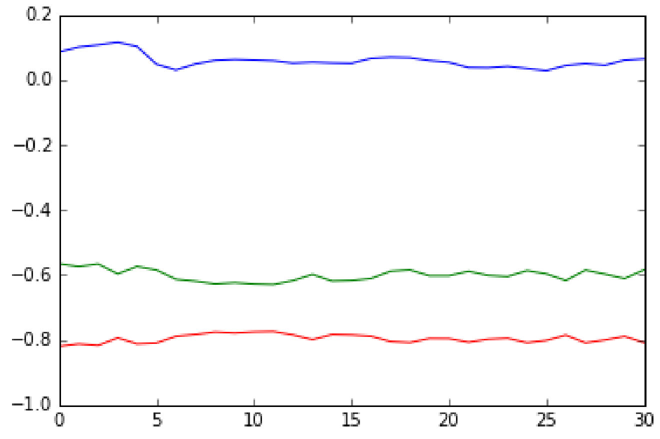


Figure 1: Raw acceleration on X, Y, Z over time

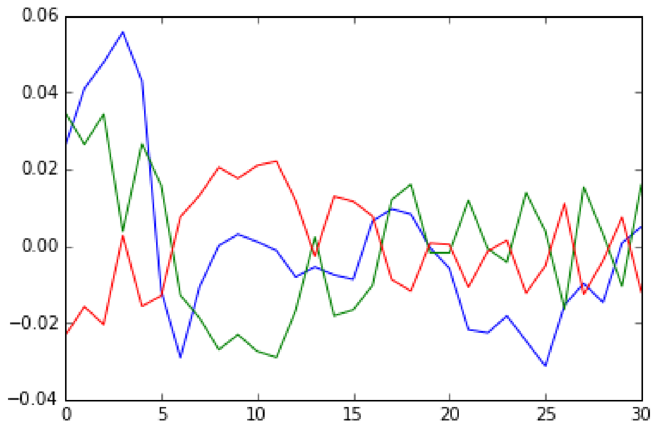


Figure 2: Centered acceleration on X, Y, Z over time

Because the acceleration measures depend on the orientation of the accelerometer, we had to come-up with new features that would be more relevant to perform the classification. We mostly used Short-time-Fourier-transform (STFT) to create new, more relevant features.

1.2 Short-time-Fourier-transform

The STFT of a signal is the Fourier-transform of the signal coupled with a moving-window h_τ for several τ. We chose a step-function as the moving window:

$$h : u \mapsto \begin{cases} 1 & \text{if u<w} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$h_\tau : u \mapsto h(u - \tau) \tag{2}$$

The STFT transform of a signal f is then:

$$STFT(\tau, \omega) = \int_t f(t) \cdot h(t-\tau) \cdot e^{-i \cdot \omega \cdot t} dt \tag{3}$$

For each acceleration components $\{ (X_t, Y_t, Z_t), t \in [0,140[\}$ we computed and used as features $\{ STFT_{X,Y,Z}(\tau, \omega), \tau, \omega \in [0,140[\}$.

1.3 New features

We computed new features independently for each data-point of our set. First, in order to get rid of the accelerometer-orientation-dependency, we performed PCA across the acceleration-data for each data-point. The idea is to find the main component for the oscillations, independently of the position and orientation of the accelerometer. This provides us with the main direction for acceleration-variations. We then projected the acceleration on this direction. We performed a STFT of the acceleration over this direction and used it for new features. The new features that we took into account for each data-point are the following:

- $STFT(\tau, \omega), \tau, \omega \in [0,140[$ on X, Y, Z

- $\bar{X}$, $\bar{Y}$, $\bar{Z}$ (average values of X_t, Y_t, Z_t) and $Var(X)$, $Var(Y)$, $Var(Z)$

- PCA eigenvectors $V_i, i = 1..3$

- $STFT(\tau, \omega), \tau, \omega \in [0, 140[$ along V_1

1.4 Feature selection

So far our features depend on one parameter which is the width of the window. We tried to find the best value for this parameter doing cross-validation using a Random Forest Classifier. We picked-up the value corresponding to the best classification rate on the test-set. Below is the curve of classification-rates for different values of window-width.

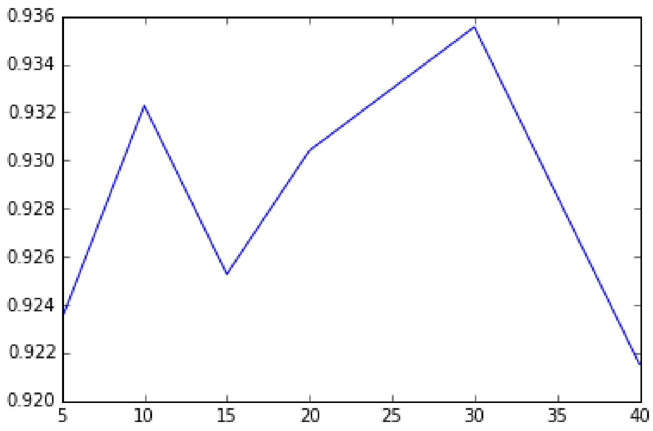


Figure 3: Classification rate over window-width

For all that follow, we used a widow-width of 30.

With this new features we can reach a score of 0.936 with a random forest classifier instead of 0.883 with the raw features. The error is down from 11.7% to 6.4% which is a 45% improvement.

1.5 Model and Objective function

We introduce score functions F_1, F_2, F_3, F_4 that are related respectively to class 1,2,3 and 4. Basically, given i, if $F_1(x_i) > F_2(x_i)$ it would mean that Y_i is more likely to be equal to 0 ("STILL") than 1 ("WALKING"). Then we try to predict the labels Y_i with $\hat{Y_i}$ defined by

$$\hat{Y_i} = [\frac{e^{F_1(x_i)}}{\sum_{c=1}^{4} e^{F_c(x_i)}}, \frac{e^{F_2(x_i)}}{\sum_{c=1}^{4} e^{F_c(x_i)}}, \frac{e^{F_3(x_i)}}{\sum_{c=1}^{4} e^{F_c(x_i)}}, \frac{e^{F_4(x_i)}}{\sum_{c=1}^{4} e^{F_c(x_i)}}]$$

Given x_i, the predicted class is the argument of $\hat{Y_i}$ with the highest value. We would like this argument to be equal to Y_i.

We are trying to minimize this objective function:

$$Obj = \sum_{i=1}^{N} J(Y_i, \hat{Y_i}) + \sum_{c=1}^{4} \Omega(F_c)$$

The regularization term Ω is function of the lengths of the trees used in the estimation of F. J is the log loss defined by $J(Y_i, \hat{Y_i}) = -\sum_{i=1}^{N} log(\hat{Y_i}[Y_i])$.

Our goal is to minimize the objective function by adjusting our score functions F_1, F_2, F_3, F_4. So basically we have transformed a classification problem into 4 regression problems. In order to fit F_1, F_2, F_3, F_4, we will use Stochastic Gradient Boosting Trees.

2 Gradient Boosting

2.1 Boosting

Boosting is a general method for improving the accuracy of any given learning algorithm: it calls this algorithm repeatedly in a series. Boosting can be used to carry out variable selection during the fitting process and addresses multicollinearity problems (by shrinking effect estimates towards zero). Boosting optimizes prediction accuracy. In Adaboost, the main ideas of the algorithm is to maintain a distribution or set of weights over the training set. Shortcoming are identified by high weight data points as it is described in paper [1]. In Gradient Boosting, shortcomings are identified by gradients. Both high-weight data points and gradients tell us how to improve our model.

2.2 Gradient Boosting Tree

Gradient Boosting Trees use regression trees as base-learners with the negative gradient as the dependent variable. These gradients $-g_c(x_i)$ are approximations of the residuals. These are the parts that the existing model cannot do well.

$$-g_c(x_i) = -\frac{\delta Obj}{\delta F_c(x_i)}$$

In each stage, for $c \in [1, 4]$, a regression tree h_c partitions the input space into nodes and predicts a separate constant value in each one and try to approximate these gradients. This tree aims at compensate the shortcomings of the current model. We then add $\mu * h_c$ to the current score function F_c: $\mu \in [0, 1]$ is the shrinkage coefficient that controls the learning rate of the algorithm. Shrinkage leads to a downward bias (in absolute value) but to a smaller variance of the effect estimates. The new model is still not satisfactory, so we can add other regression trees until convergence.The stopping iteration is chosen such that it maximizes prediction accuracy. The algorithm is introduced in [2] and extensively in [3].

Algorithm 1 Gradient Boosting Trees

```
1: procedure
2:     $F_1, F_2, F_3, F_4$ random
3:     while the model is not satisfactory do
4:         for $c \in [1, 4]$ do
5:             for $i \in [1, N]$ do
6:                 compute negative gradients $-g_c(x_i) = -\frac{\delta Obj}{\delta F_c(x_i)}$
7:             end for
8:             fit a regression tree $h_c$ with data $(\{x_i, -g_c(x_i)\}_1^N)$
9:             $F_c := F_c + \mu * h_c$
10:        end for
11:    end while
12:    return $F_1, F_2, F_3, F_4$
13: end procedure
```

2.3 Stochastic Gradient Boosting Trees

We select a random data subsample of size $\tilde{N}$ in place of the full sample of size N to fit the base learners and compute the model update at each iteration. Similarly to random forests, we will also select only a subsample of size $\tilde{M}$ of features in order to increase variance among the tree classifiers. It will increase the execution speed of gradient boosting. Moreover, it is shown in [4] that the prediction accuracy can be improved by preventing overfitting.

Algorithm 2 Stochastic Gradient Boosting Trees

```
1: procedure
2:     $F_1, F_2, F_3, F_4$ random
3:     while the model is not satisfactory do
4:         $\{\pi(i)\}_1^N = RandomPermutation(\{i\}_1^N)$
5:         $\{\sigma(i)\}_1^M = RandomPermutation(\{i\}_1^M)$
6:         for $c \in [1, 4]$ do
7:             for $i \in [1, \tilde{N}]$ do
8:                 compute negative gradients $-g_c(x_{\pi(i)}) = -\frac{\delta Obj}{\delta F_c(x_{\pi(i)})}$
9:             end for
10:            fit a regression tree $h_c$ with data $(\{x_{\pi(i)}[\{\sigma(i)\}_1^{\tilde{M}}], -g_c(x_{\pi(i)})\}_1^{\tilde{N}})$
11:            $F_c := F_c + \mu * h_c$
12:        end for
13:    end while
14:    return $F_1, F_2, F_3, F_4$
15: end procedure
```

2.4 Selecting Stochastic Gradient Boosting Parameters

We use the XGboost package that implements Stochastic Gradient Boosting Trees on python, [5]. We have to find the best possible values for our 4 parameters

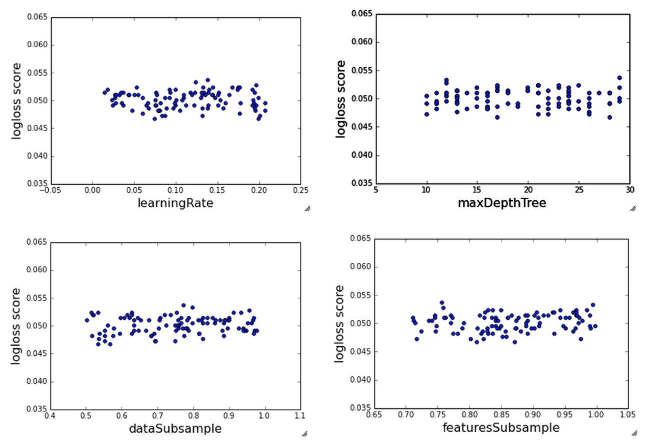


Figure 4: Logloss score as function of the parameters in our experiments

- maxDepthTree: the maximum depth of the regression trees fit at each stage.
- μ: the learning rate of the algorithm
- dataSubsample=$\frac{\tilde{N}}{N}$: the proportion of data used at each iteration to fit the regression trees
- featuresSubsample=$\frac{\tilde{M}}{M}$: the proportion of features used at each iteration to fit the regression trees

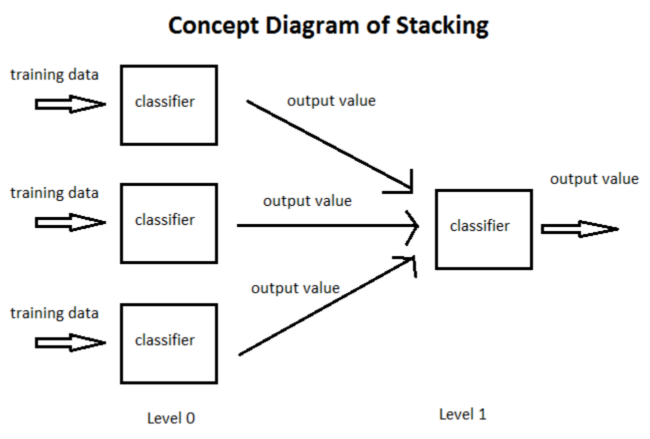
By running our algorithm several times with different parameters, we obtained different efficient models. To estimate log loss scores, we used 3 fold cross validation. We select the 10 best models and we try to have parameters as different as possible to increase variance of our models.

3 Stacking

3.1 Combining different models

By running our algorithm several times with different parameters, we obtained different efficient models. So given x_i, each model outputs $\hat{Y_i}$. We tried to blend these output probability arrays in order to make one even more powerful. The first idea is to use traditional ensemble learning. We use an automatic combining mechanism to combine the outputs of the different models. We have done it by averaging the results.

In stacking, the combining mechanism is also a classifier that needed to be trained. The output of the models computed previously will be used as training data for the second classifier to approximate the same target function.



3.2 Logistic Regression

Lasso Logistic regression is an efficient meta level classifier as it has low variance and does not overfit too much. We used the scikit-learn package of python to encode our Logistic Regression. We improved our final results with some features engineering. As every feature was a probability (in [0,1]) we choose $x := log(\frac{x}{1-x})$.

3.3 Neural Networks

We implemented a Neural Network classifier using the Theano package on python, [6]. Our layers were either Sigmoid function or Rectified Linear Uni function. Our neural network is trained according to the back propagation algorithm. The error propagate backward, from the output nodes (the labels) to the hidden nodes and the input nodes. It calculates the gradient of the error according to the different weights parameters, and update these parameters. More precisely, we used a batch stochastic gradient method: at each iteration, the algorithm only considers a small proportion of the training data. We obtained quick convergence to a local minima.

According to [7], adding a drop out probability, in other words randomly omitting some of the features on each training point, prevent complex co-adaptations on the training data and so overfitting. So we added a drop out probability equal to 0.5. We also prevent the parameters from being to huge by regularizing with a L_1 and a L_2 norm in the objective function of the neural network.

We first try to predict the different classes with our original features. We choose 5 layers, 200 nodes by hidden layers. Inspired by the method used to win the Netflix challenge as it is explained in [8], we used our neural network as a stacking classifieer. The features of this neural network were the outputs of the first classifiers and the original features x_i as neural nets handle large amount of features. This time we only take 3 layers. We increased our prediction accuracy compared to the mere neural network.

4 Results

Below are the comparative results. We can take the best Tree, without any staking as a baseline and then compare the different sacking methods: average, prediction, with a logistic regression, with additional engineered features and with neural networks.

Without Stacking	**Prediction Accuracy**
Sto. Gradient Boosting Trees	0.94
Neural Networks	0.88
Random Forest	0.936

Stacking classifier	**Prediction Accuracy**
Average prediction	0.945
Logistic regression	0.95
Logistic regression with features engineering	0.958
Neural Networks	0.93

We can see that the use of stacking with logistic regression allows to reduce the error from 0.06 to 0.042 which is a $\approx 30\%$ improvement. We notice that neural networks do not perform very well. We could improve our results by trying different number of layers, different number of nodes or new values for our parameters.

It is interesting to notice that some models perform better on some specific labels. By combining all this characteristics into one single classifier, stacking allows to outperform all the other classifiers taken separately. For example, if a certain model performs well to detect the label "STILL", then its weights will be increased to predict this label, but non necessarily to predict the other labels. As the different models make different kind of errors, combining them together enable to increase the prediction accuracy. It is a kind of model selection.

References

[1] Robert E Schapire. The boosting approach to machine learning: An overview. In *Nonlinear estimation and classification*, pages 149–171. Springer, 2003.

[2] Tianqi Chen. Introduction to boosted trees. 2014. http://homes.cs.washington.edu/ tqchen/pdf/BoostedTree.pdf.

[3] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[4] Jerome H Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

[5] Distributed (Deep) Machine Learning Common. Xgboost. https://github.com/dmlc/xgboost/blob/master/doc/python.md.

[6] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[7] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[8] Andreas Töscher and Michael Jahrer. The bigchaos solution to the netflix prize 2008. *Netflix Prize, Report*, 2008.