Machine Learning Capstone Project

Human Activity Recognition (HAR) is a research that aims to develop systems to realize automatic recognition of physical activities to extract information about the user-behavior so that these systems can proactively improve user’s experience and interaction with the computer. This is usually done by utilizing external sensors (e.g. environmental cameras), sensors on the user (e.g. wearables, body-worn sensors), or sensors embedded in the objects that we interact with (e.g. smartphones). HAR systems successfully took place in products like Nintendo Wii for entertainment and Nike+ running shoes for fitness.

One challenge with the activity recognition that doesn’t necessarily exist in object recognition or speech recognition is that HAR offers more degrees of freedom in terms of system design and implementation. Due to its temporal nature, it is not very clear what starts/ends and when. In other words, there is no common clear definition, grammar or structure of human activities that we can use to make a clear and generic problem statement (**Bulling et al., 2014**).

# Project Overview

The focus of this project is to recognize activities of daily living based on the motion related data acquired through a waist-mounted smartphone with embedded motion-sensitive sensors.

Dataset is consisted of the motion related data sampled from the activities of 30 people performing walking, walking-upstairs, walking-downstairs, sitting, standing, and laying and the activity labels given by the experimented based on the recorded video of the performed activities.

Source of the sampled data is the embedded sensor streaming data regarding 3-axial linear acceleration and 3-axial angular velocity at a constant rate of 50Hz.

# Problem Statement

In this project, the main goal is to classify the 561 dimensional feature space representing the daily life activities into the categories corresponding to the activities labeled as walking, walking-upstairs, walking-downstairs, sitting, standing, and laying. To achieve this goal, various machine learning methods are trained and their learning and prediction performances are compared, and SVM model has been chosen to carry out other data processing and tuning operations.

# Metrics

To measure the classification performance of different methods, I used k-fold cross-validation and extracted precision, recall, f-score metrics for every single fold. I considered the weighted average of the metrics based on the number of class labels. This way, obtained metrics are more robust against the imbalance between different class labels.

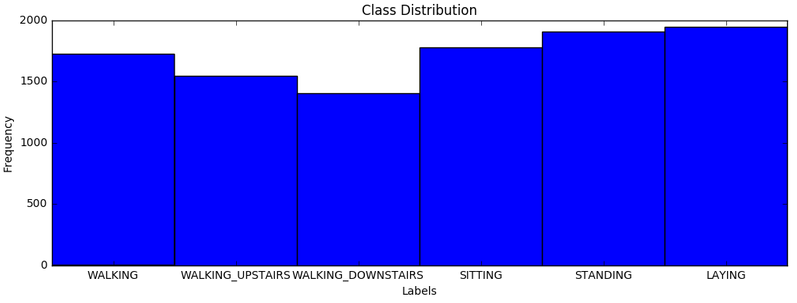
I took the mean of these metrics for every single folding to obtain final metrics regarding the performance of the classification methods.

# Exploratory Data Analysis

## Dataset

Dataset is consisted of 10299 manually labeled samples. Labels are walking, walking-upstairs, walking-downstairs, sitting, standing and laying, and their distribution is as follows:

* 1722 WALKING,
* 1544 WALKING\_UPSTAIRS,
* 1406 WALKING\_DOWNSTAIRS,
* 1777 SITTING,
* 1906 STANDING,
* 1944 LAYING



Sampled data were already pre-processed by applying noise filters and re-sampled with fixed-width sliding windows of length 128 (2.56 sec/window) and 50% overlap. In addition, linear acceleration data was filtered from gravitational acceleration as it is almost constant and carries no information regarding the activities. A Butterworth low-pass filter was applied to remove the gravitational acceleration in frequency domain which was assumed to correspond to a frequency of less than 0.3Hz.

Filtered and re-sampled data was then converted into a feature vector of size 561. Features are subtracted from the properties in temporal and frequency domain. These include first order statistical properties like minimum, maximum, mean, standard deviation, and other variability measures like mean absolute deviation, interquartile range, and some other statistical properties like auto-regression coefficients, and correlation.

## Features

Visualization of a space represented by 561 features is quite difficult for bare eyes. I utilized KBest feature selection method to reduce the dimension of the feature space. To decide on the best “K”, I decided to train dataset and pick the best and most reasonable K value to have closer look at how the features are distributed. In order to decide on the training method, I applied various supervised machine learning methods that are available in scikit learn. These are Stochastic Gradient Descent Classifier, Ada-boost Classifier, Decision Tree Classifier, K-Neighbors Classifier, Gaussian Naïve Bayes and Support Vector Machine. Here is the classification performance of these methods:

GNB precision: 0.80 recall: 0.73 fscore: 0.72 t\_train: 0.2316sec t\_pred: 0.0533sec

**SVM precision: 0.94 recall: 0.94 fscore: 0.93 t\_train: 13.0880sec t\_pred: 3.1674sec**

**SGD precision: 0.95 recall: 0.94 fscore: 0.94 t\_train: 0.4460sec t\_pred: 0.0050sec**

Ada precision: 0.37 recall: 0.54 fscore: 0.41 t\_train: 41.1263sec t\_pred: 0.0405sec

DTR precision: 0.88 recall: 0.87 fscore: 0.87 t\_train: 7.0560sec t\_pred: 0.0027sec

KNC precision: 0.91 recall: 0.91 fscore: 0.91 t\_train: 0.9203sec t\_pred: 11.7007sec

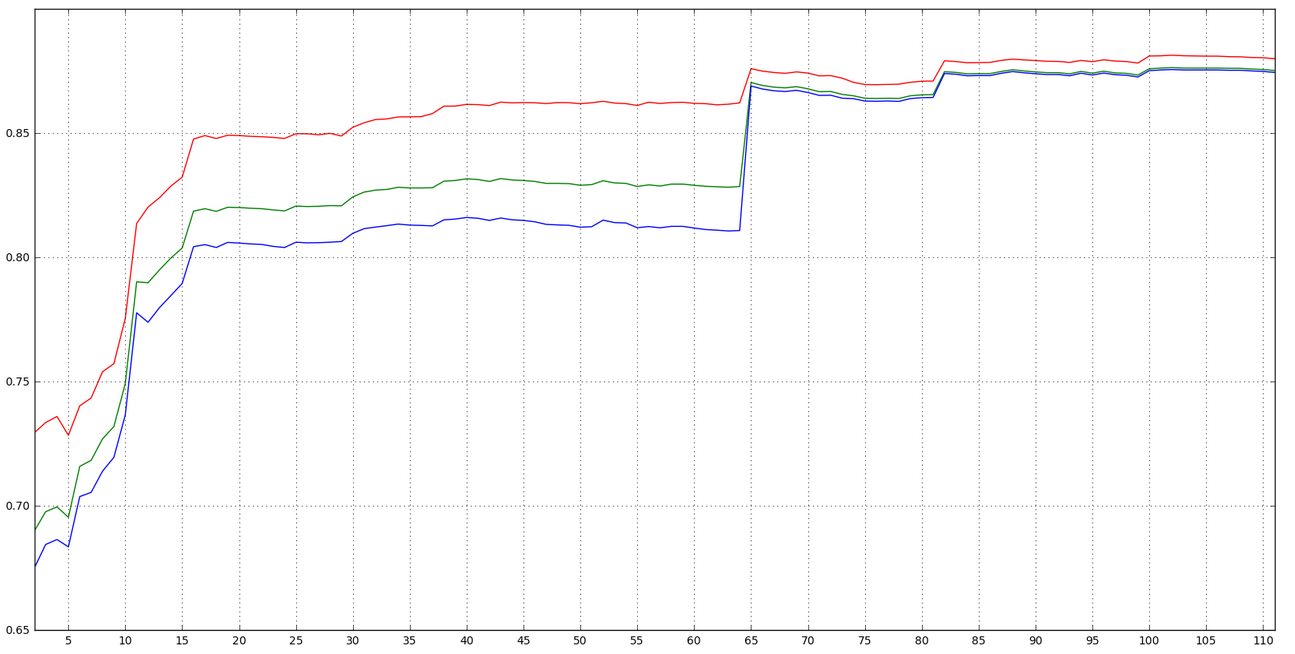
SVM and SGD have the highest precision, recall and f1-score. SGD is the quickest in prediction and second quickest in training. I will be using SVM as the main method while discovering different feature processing methods with the default parameters that comes with the SVC model in *scikit learn*:

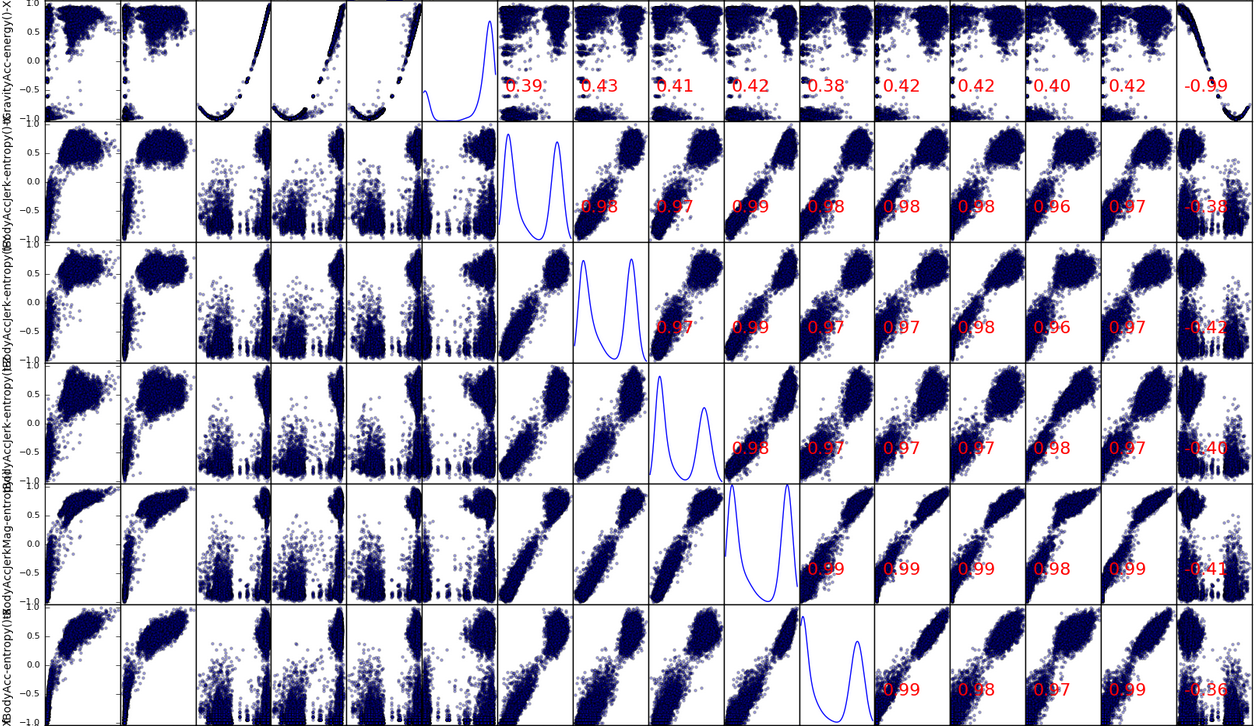
SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0, decision\_function\_shape=None, degree=3, gamma='auto', kernel='rbf', max\_iter=-1, probability=False, random\_state=None, shrinking=True, tol=0.001, verbose=False)

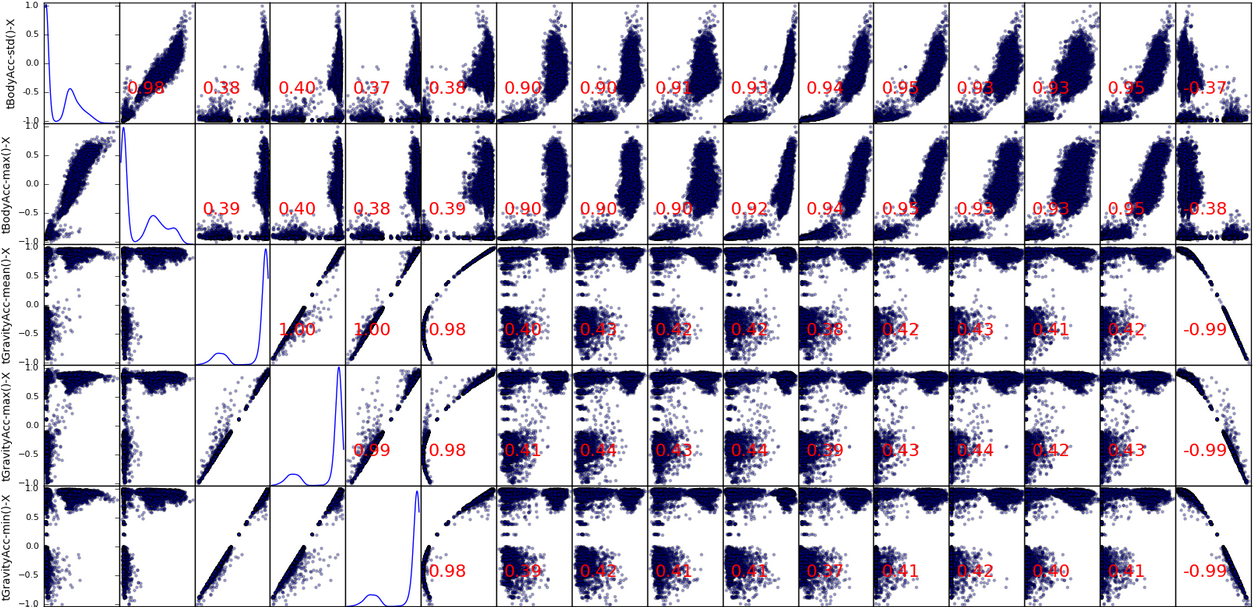
Although SVM's cross-validated classification performance is already very high (p=0.94, r=0.94, f=0.93), further investigation might still yield in even a higher classification performance. For instance, removing the outliers is one way to improve the model. Moreover, some of the features might be redundant. Redundant features can easily be distinguished by investigating the correlation between them and the other features. If there is high correlation with other features which means there is no reason to carry this feature as the information represented by this feature is already conveyed through other features.

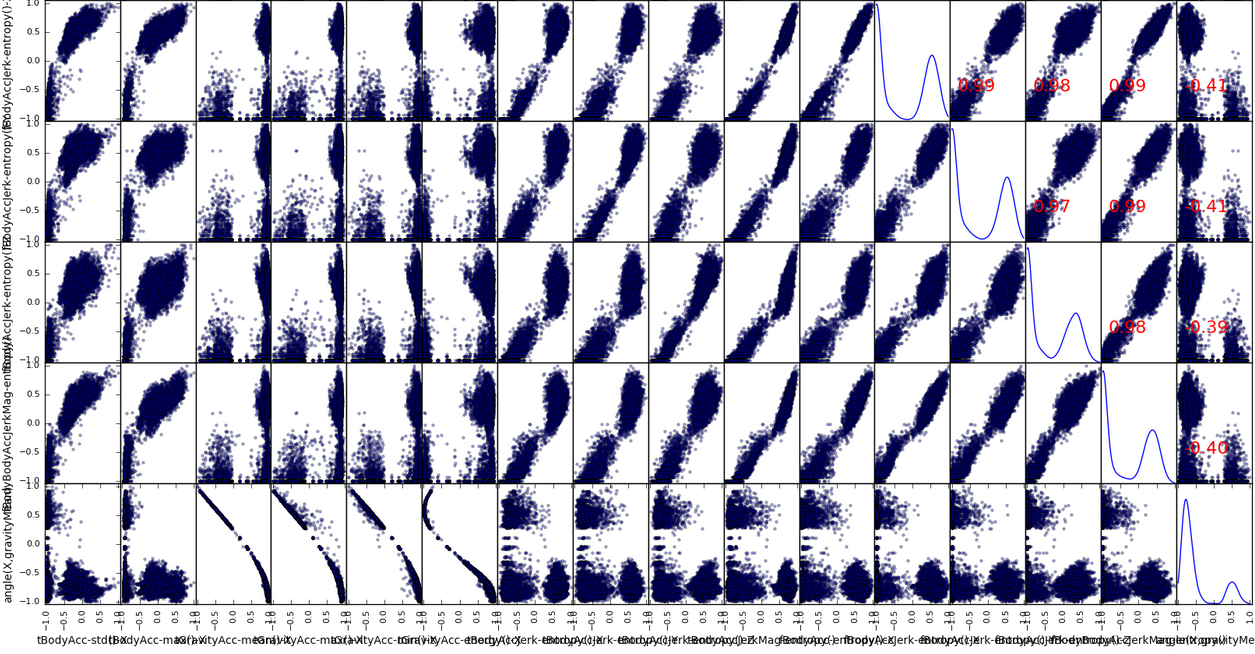
Correlation matrix will allow us to see the distribution of the feature values individually and correlation between them. SelectKBest method helped me to choose a subset of features for this investigation. In order to decide on the number K, I trained the SVM model with K’s starting from K=2 to 20% of the feature vector size which is 112.

**Precision (Red), Recall (Green), F1-score (blue) vs number of KBest features**



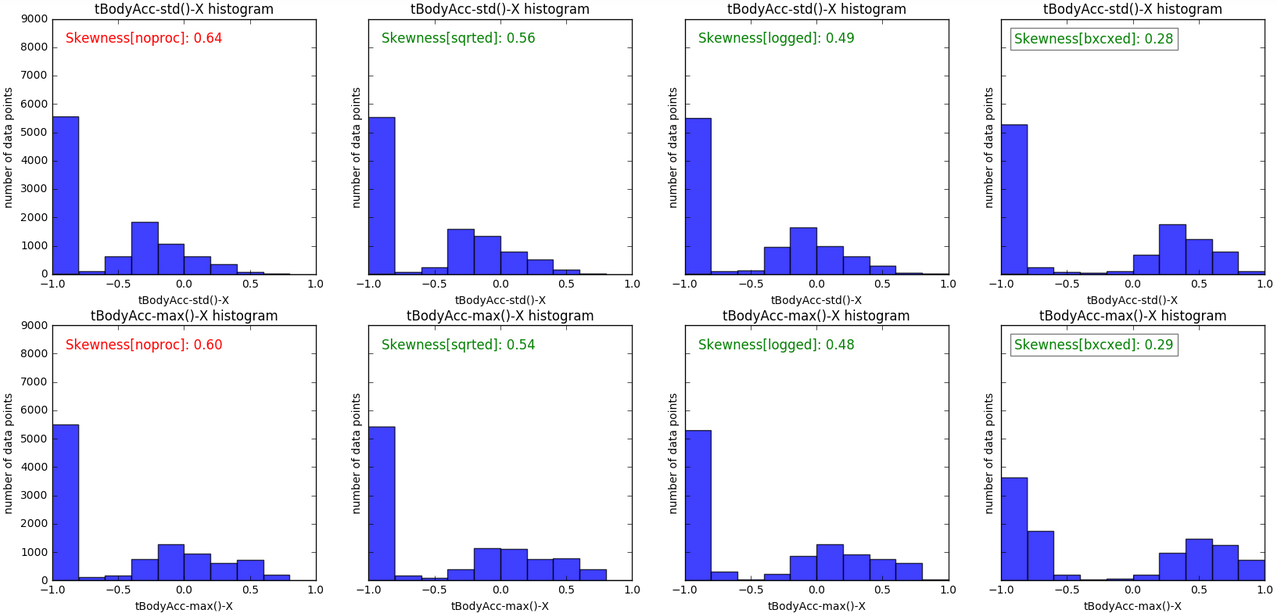
SVM’s classification performance has been measured with cross-validation with 4 folds in terms of precision, recall and f1-score metrics. This graph is drawn based on the mean of the 4 values for each of the metrics. According to the graph, we see that SVM reaches the first peak performance (precision: 0.85, recall: 0.82, f-score: 0.80) when “K=16”. To further investigate how these features are distributed and correlated, I plotted the correlation matrix.



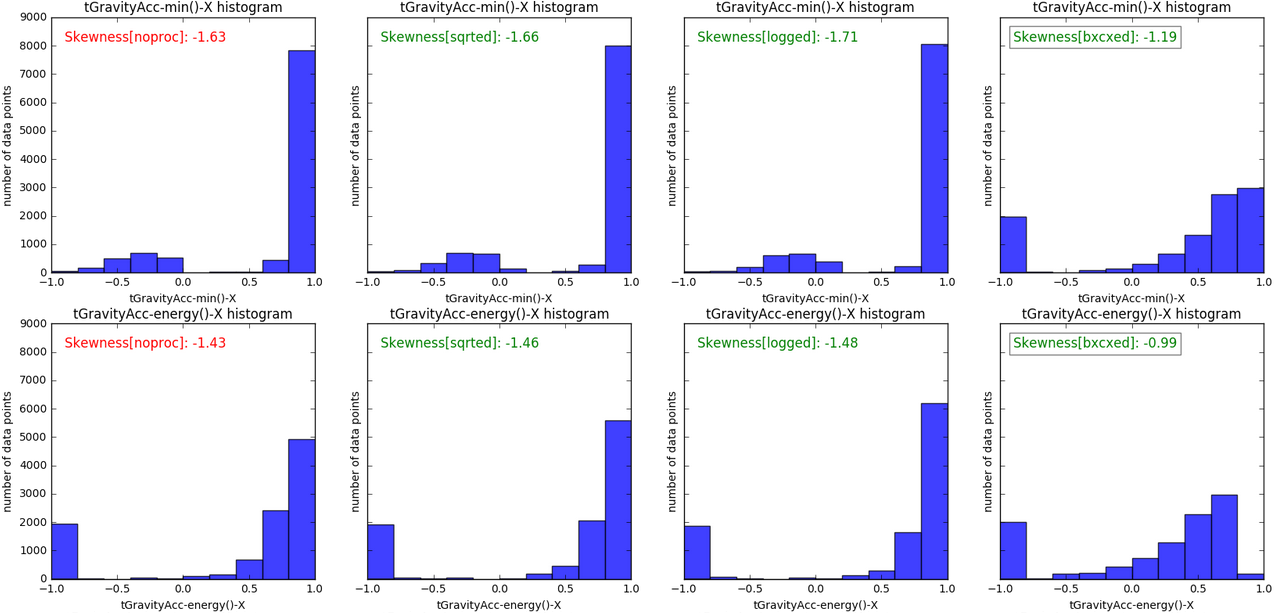
Having normally distributed features is the fundamental assumption in many predictive models. Normal distribution is un-skewed. It means the probability of falling in the right or left side of the mean is equally likely. Skewness greater than zero shows a positively skewed distribution, while lower than zero shows a negatively skewed distribution. As we can see from the correlation matrix above, these features are quite skewed, even mostly bimodal. Transforming these features with the log, square root, or inverse may help to remove the skew. However, feature values of the current dataset with selected features change between -1 and 1. Therefore, sqrt and log is not applicable. If we apply any of those transformations, most of the feature values will turn into NaN and dataset will be useless.

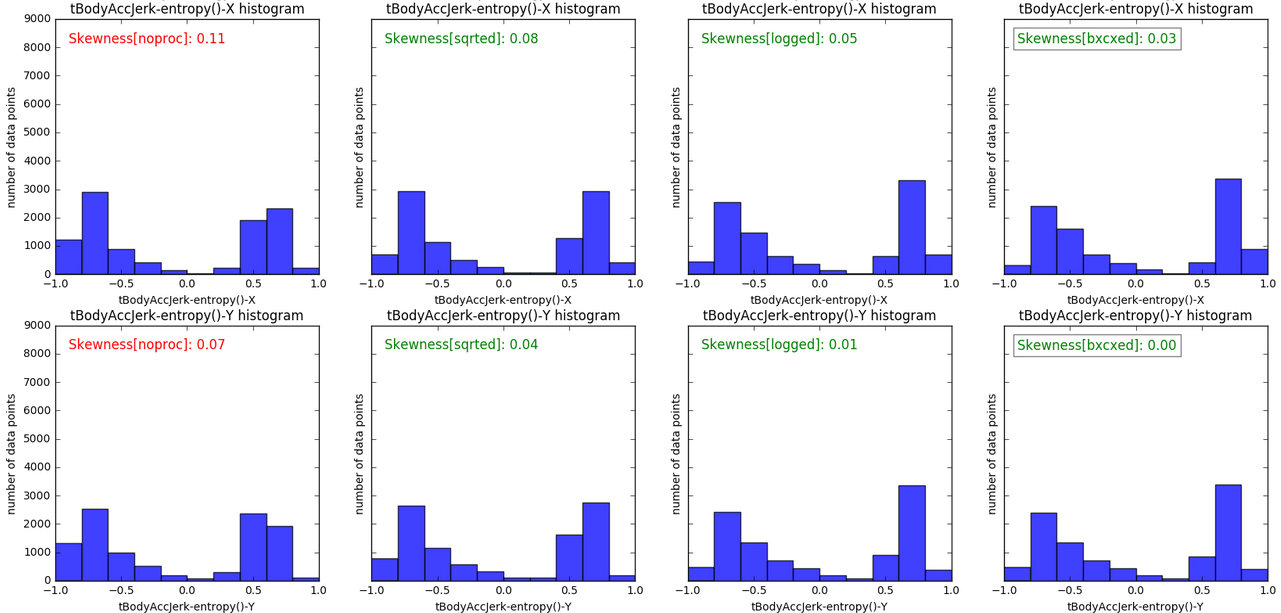
To avoid this we first shift data to a non-negative range, then apply the non-linear transformation, after that scale it back to -1 and 1 to be able to compare the change in the feature distribution with bare eyes. If all go right, we should be able to see less skewed feature distribution.

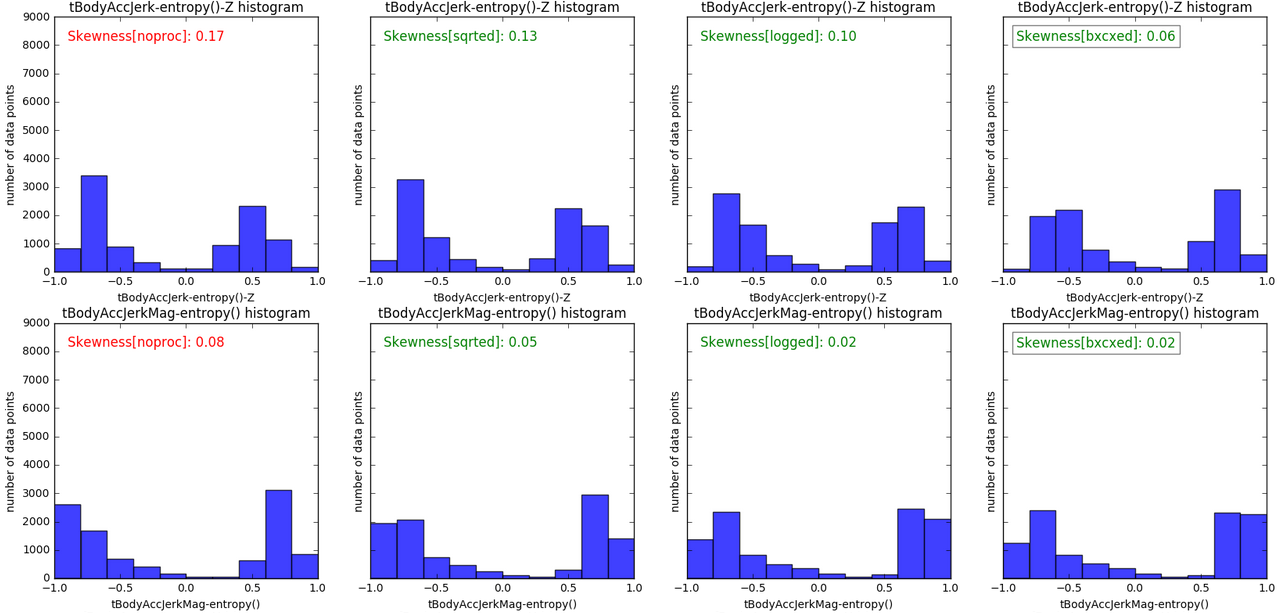
In addition to sqrt-ing and log-ing, I will also try boxcox-ing to reduce the skewness.

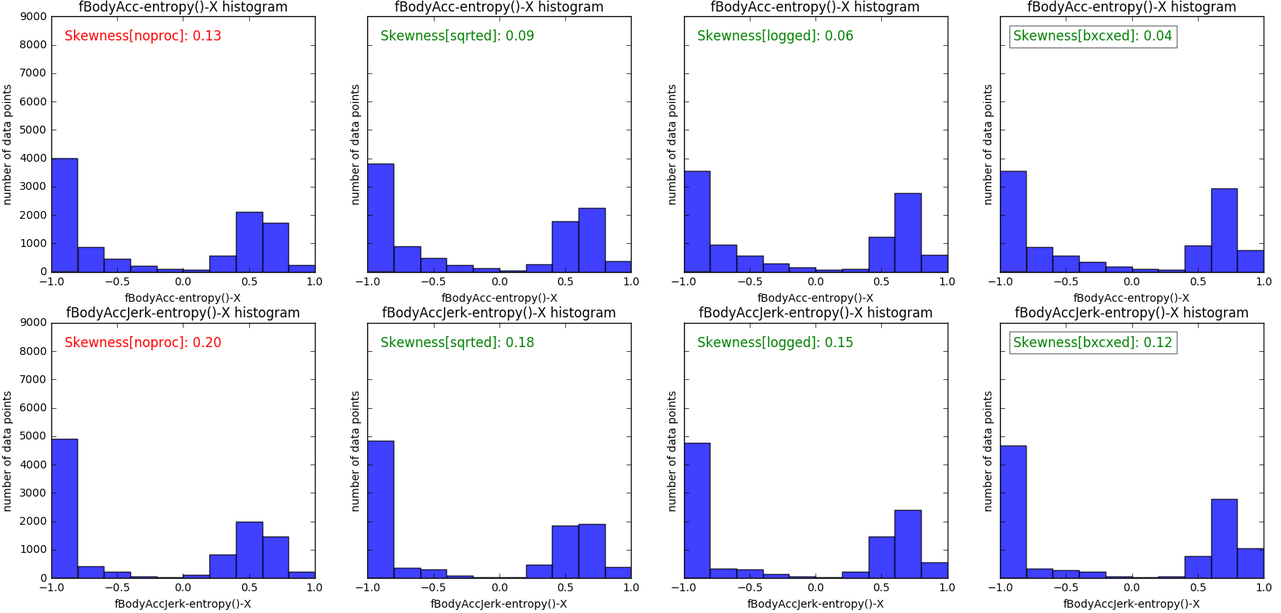


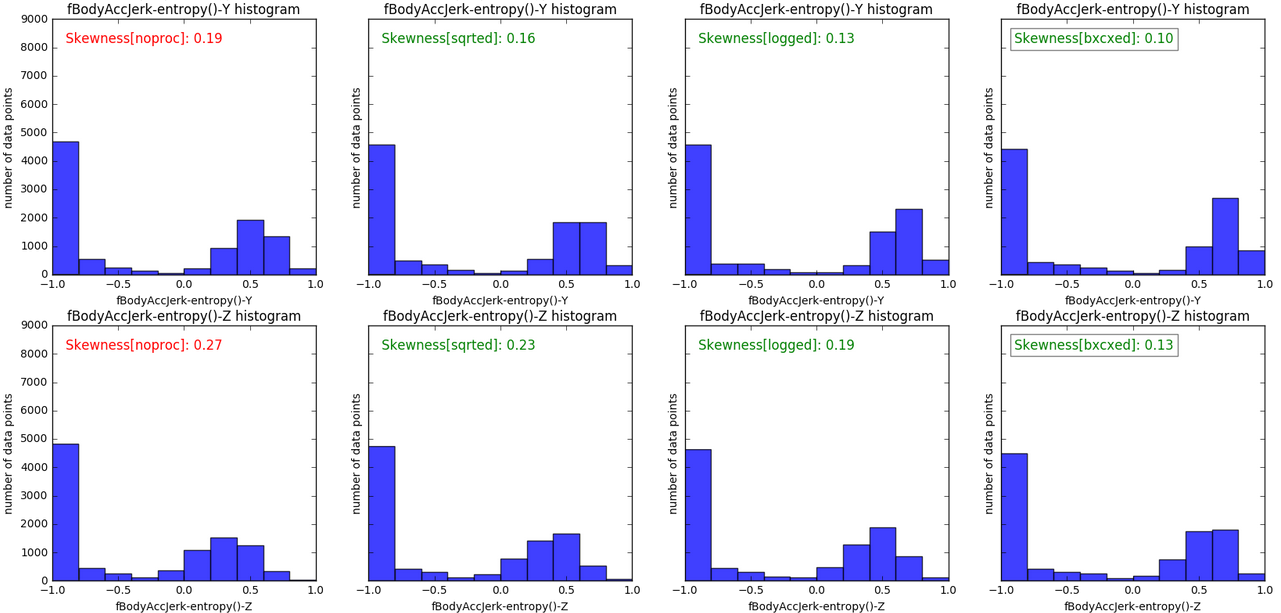


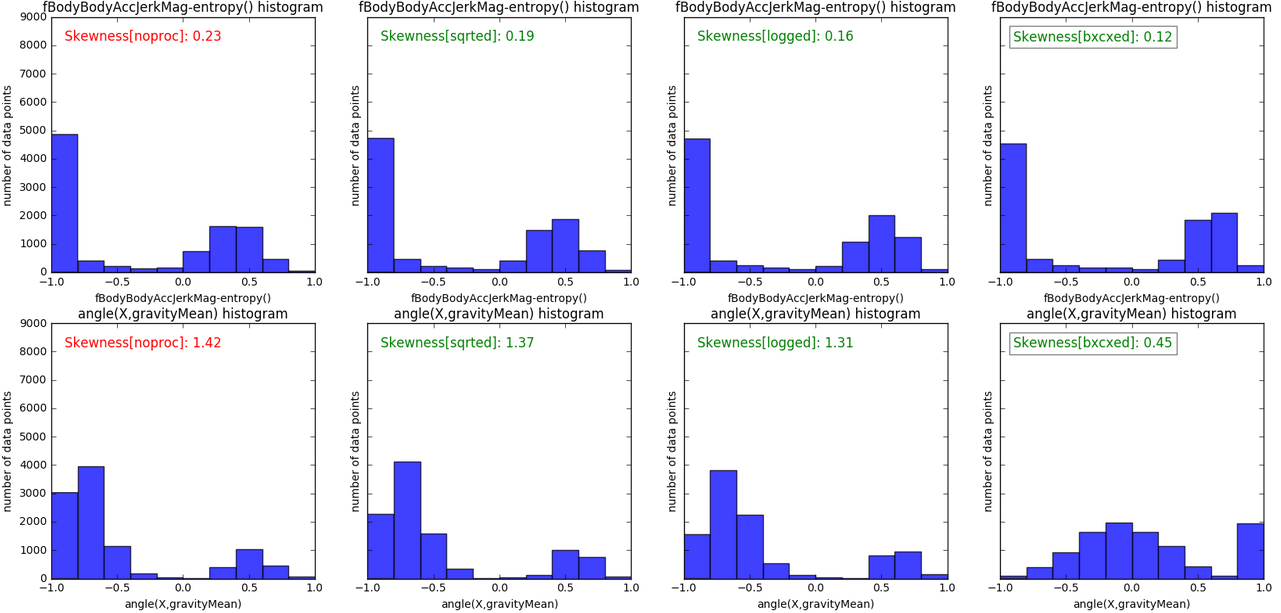












I was curious how these 16 features do compared to the other 545 features that were filtered by the SelectKBest method. I could at least check the skewness of all of the features after applying boxcox-ing to all of the 561 features, and sort them to see where these 16 features fall into in terms of the skewness. I expect to see that these 16 features among the top features that have the lowest skewness.

48\* 0.0741980322368 tBodyAccJerk-entropy()-Y

51\* 0.0827741497792 tBodyAccJerkMag-entropy()

65\* 0.10593815062 tBodyAccJerk-entropy()-X

75\* 0.125307722428 fBodyAcc-entropy()-X

92\* 0.165858145244 tBodyAccJerk-entropy()-Z

100\* 0.193019223009 fBodyAccJerk-entropy()-Y

105\* 0.200122462248 fBodyAccJerk-entropy()-X

117\* 0.225381717128 fBodyBodyAccJerkMag-entropy()

129\* 0.269810028483 fBodyAccJerk-entropy()-Z

181\* 0.601852530578 tBodyAcc-max()-X

194\* 0.63692184831 tBodyAcc-std()-X

347\* 1.42296186491 angle(X, gravityMean)

349\* 1.42884856959 tGravityAcc-energy()-X

365\* 1.62648611505 tGravityAcc-min()-X

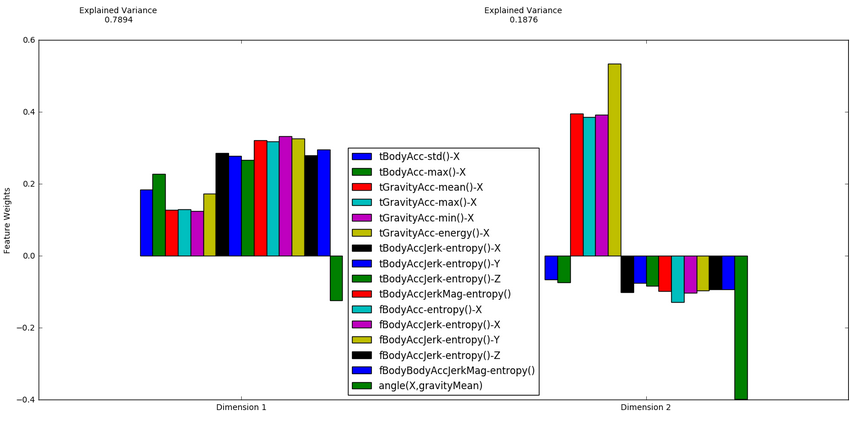
367\* 1.62924391195 tGravityAcc-mean()-X

369\* 1.64228675532 tGravityAcc-max()-X

Half of the selected features (8 of the 16 SelectBestK features) are in the top 20% of the 561 features sorted by their skewness from low to high.

## Feature Selection

To see how good these 16 KBest features are, I also want to have a look at their expressivity power. In other words, how good these 16 features are in terms of the information they carry. One way to do this is to see how correlated they are with the principal components that express this space.



This figure shows the level of alignment each feature has with the principal components. Major principal component has a power of 0.79 to explain the variance of the 16 dimensional feature space represented by the KBest (K=16 ) features. It seems these features are not aligned with the major principal component nor the minor principal component. Therefore, principal components can improve the classification performance of the model trained by using the KBest features. After I add these two principal components to the features space represented by 16 KBest features, classification performance increased:

kbest: 16 **precision: 0.85 recall: 0.82 fscore: 0.80** t\_train: 0.652sec t\_pred: 0.425sec

(kbest 16)(pca\_n 2) **precision: 0.86 recall: 0.85 fscore: 0.85** t\_proc: 1.23 t\_train: 0.88 t\_test: 0.19

Based on this improvement, I decided to combine KBest features with the PCA’s components to maximize the classification performance. Those are some of the highest scores I obtained by combining KBest features and PCA’s components.

**(kbest 5)(pca\_n 50) precision: 0.94 recall: 0.93 fscore: 0.93 t\_proc: 1.58 t\_train: 1.16 t\_test: 0.25**

(kbest 5)(pca\_n100) precision: 0.95 recall: 0.94 fscore: 0.94 t\_proc: 1.50 t\_train: 1.81 t\_test: 0.42

(kbest 5)(pca\_n200) precision: 0.95 recall: 0.95 fscore: 0.95 t\_proc: 1.54 t\_train: 3.65 t\_test: 0.90

(kbest 10)(pca\_n 50) precision: 0.94 recall: 0.93 fscore: 0.93 t\_proc: 1.51 t\_train: 1.23 t\_test: 0.27

(kbest 10)(pca\_n100) precision: 0.95 recall: 0.94 fscore: 0.94 t\_proc: 1.52 t\_train: 1.93 t\_test: 0.45

**(kbest 10)(pca\_n200) precision: 0.95 recall: 0.95 fscore: 0.95 t\_proc: 1.58 t\_train: 3.72 t\_test: 0.92**

(kbest 15)(pca\_n100) precision: 0.95 recall: 0.94 fscore: 0.94 t\_proc: 1.56 t\_train: 1.97 t\_test: 0.47

(kbest 15)(pca\_n200) precision: 0.95 recall: 0.95 fscore: 0.95 t\_proc: 1.69 t\_train: 3.88 t\_test: 0.94

(kbest 20)(pca\_n100) precision: 0.95 recall: 0.94 fscore: 0.94 t\_proc: 1.75 t\_train: 2.03 t\_test: 0.48

(kbest 20)(pca\_n200) precision: 0.95 recall: 0.95 fscore: 0.94 t\_proc: 1.67 t\_train: 3.85 t\_test: 0.96

(kbest 50)(pca\_n100) precision: 0.95 recall: 0.94 fscore: 0.94 t\_proc: 2.14 t\_train: 2.50 t\_test: 0.61

(kbest 50)(pca\_n200) precision: 0.95 recall: 0.94 fscore: 0.94 t\_proc: 2.26 t\_train: 4.43 t\_test: 1.11

(kbest100)(pca\_n 40) precision: 0.94 recall: 0.93 fscore: 0.93 t\_proc: 3.14 t\_train: 2.42 t\_test: 0.60

(kbest100)(pca\_n 50) precision: 0.94 recall: 0.93 fscore: 0.93 t\_proc: 3.13 t\_train: 2.56 t\_test: 0.63

(kbest100)(pca\_n100) precision: 0.95 recall: 0.95 fscore: 0.94 t\_proc: 3.14 t\_train: 3.34 t\_test: 0.83

(kbest100)(pca\_n200) precision: 0.95 recall: 0.95 fscore: 0.95 t\_proc: 3.24 t\_train: 5.36 t\_test: 1.37

(kbest200)(pca\_n 30) precision: 0.94 recall: 0.93 fscore: 0.93 t\_proc: 5.47 t\_train: 4.01 t\_test: 1.03

(kbest200)(pca\_n 40) precision: 0.94 recall: 0.93 fscore: 0.93 t\_proc: 5.50 t\_train: 4.14 t\_test: 1.06

(kbest200)(pca\_n 50) precision: 0.94 recall: 0.94 fscore: 0.93 t\_proc: 5.52 t\_train: 4.24 t\_test: 1.08

(kbest200)(pca\_n100) precision: 0.95 recall: 0.94 fscore: 0.94 t\_proc: 5.54 t\_train: 5.04 t\_test: 1.29

(kbest200)(pca\_n200) precision: 0.95 recall: 0.95 fscore: 0.94 t\_proc: 5.56 t\_train: 7.13 t\_test: 1.85

(kbest561)(pca\_n 2) precision: 0.94 recall: 0.94 fscore: 0.94 t\_proc: 17.95 t\_train: 10.98 t\_test: 2.77

(kbest561)(pca\_n 5) precision: 0.94 recall: 0.94 fscore: 0.94 t\_proc: 17.98 t\_train: 10.31 t\_test: 2.64

(kbest561)(pca\_n 10) precision: 0.94 recall: 0.94 fscore: 0.94 t\_proc: 17.98 t\_train: 10.26 t\_test: 2.59

(kbest561)(pca\_n 15) precision: 0.94 recall: 0.93 fscore: 0.93 t\_proc: 17.97 t\_train: 10.15 t\_test: 2.56

(kbest561)(pca\_n 20) precision: 0.94 recall: 0.94 fscore: 0.93 t\_proc: 17.96 t\_train: 10.31 t\_test: 2.60

When we compare with the original results we obtained, feature filtering didn’t provide a considerable amount of classification performance.

Here is the SVM’s classification performance in the 561 dimensional feature space.

**(561 features) SVM precision: 0.94 recall: 0.94 fscore: 0.93 t\_train: 13.0880sec t\_pred: 3.1674sec**

Here are the SVM’s classification performance with various KBest and PCA combinations:

**(kbest 5)(pca\_n 50) precision: 0.94 recall: 0.93 fscore: 0.93 t\_proc: 1.58 t\_train: 1.16 t\_test: 0.25**

**(kbest 10)(pca\_n200) precision: 0.95 recall: 0.95 fscore: 0.95 t\_proc: 1.58 t\_train: 3.72 t\_test: 0.92**

With the smaller feature space (kbest = 5, pca\_n=50) we see 10+ times improvement in the training and testing time but no improvement in the classification accuracy. The smallest feature space giving the max classification accuracy is when kbest=10 and pca\_n=200, and that is still 4+ times quicker in training and testing than the default feature space.

## Outlier Detection

These results didn’t seem too impressive to me. I decided to check if there are potential outliers. I boxcox-transformed the 16 dimensional feature space, and ran a filter which marks the samples as outliers based on Tukey’s Method. There were only 2 features marking 23 samples as outliers, and three features marking 1953 samples as outliers. 1953 points seemed to be too much to remove (almost 20% of the dataset) given the fact that they are marked as outliers only in three feature directions. I removed them in anyways to see the effect in classification performance:

**subsetsize: 8346 precision: 0.88 recall: 0.87 fscore: 0.87**

This was better than the training performance of KBest (K=16):

**subsetsize: 10299 precision: 0.85 recall: 0.82 fscore: 0.81**

# Algorithms and Techniques

As the dataset already includes labels, I decided to apply supervised machine learning methods. To explore different techniques and compare their classification performances, I applied Stochastic Gradient Descent Classifier, Ada-boost Classifier, Decision Tree Classifier, K-Neighbors Classifier, Gaussian Naïve Bayes and Support Vector Machine. Among those, I chose SVM for further parameter tuning as it returned the second best default performance and it offers a set of parameters to be easily optimized.

By applying cross-validated grid search, best parameters found to be:

**The best parameters are {'kernel': 'rbf', 'C': 1000, 'gamma': 0.001} with a score of 0.93**

# References

Bulling, Andreas, Ulf Blanke, and Bernt Schiele. "A tutorial on human activity recognition using body-worn inertial sensors." *ACM Computing Surveys (CSUR)* 46.3 (2014): 33.