Data Science Capstone Project: Human Activity Recognition

Presented by Sofy Weisenberg

June 2020

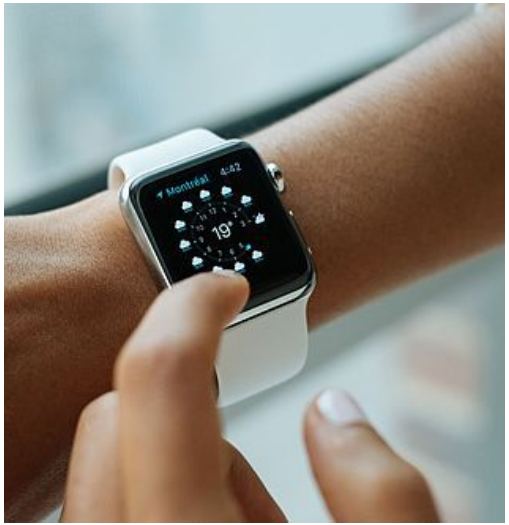
Presentation Outline

- Introduction: Motivations and Problem Description
- Dataset and Feature Engineering
- Exploratory Data Analysis / Statistical Inference
- Implementing and Comparing Machine Learning Models
- Conclusions

Introduction: Motivations and Problem Description

Background/Motivation

- Human activity recognition (HAR) has become an in-demand capability for many growing applications over recent years, including:
  - monitoring of activities of patients suffering from chronic diseases (such as diabetes or COPD) to continuously manage their health/treatments
  - monitoring sleep and exercise patterns
- The reduced size of accelerometers and gyroscope components have made them standard in most smartphone and wearable computer devices.
- However, accurate, real-time activity classification is an ongoing area of research.



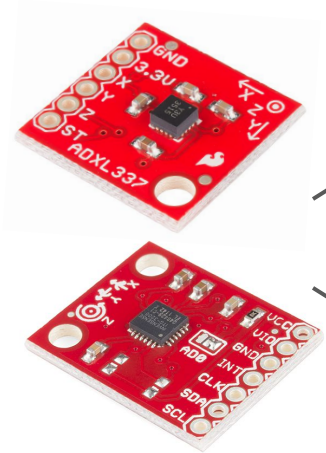
Problem Description

Given sensor data gathered from several subjects performing 6 activity types, train a model that is able to classify new (unseen) sensor data into one of the observed activity types.

Proposed Solution

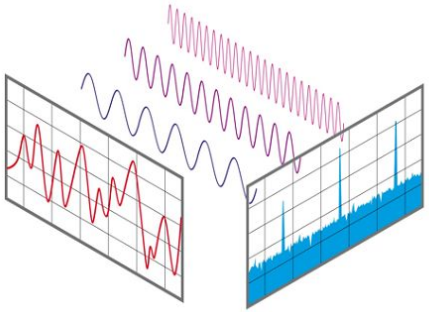
Per literature on the subject of human activity recognition, train a traditional machine learning algorithm (SVM) and various neural networks (variations on RNN) to evaluate and compare model classification performance when supplied with new (unseen) sensor data.

Traditional ML vs. Neural Network

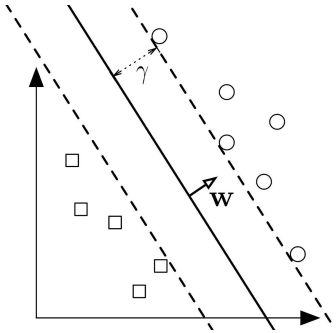


raw sensor data

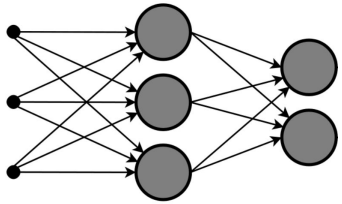
signal processing/
feature engineering



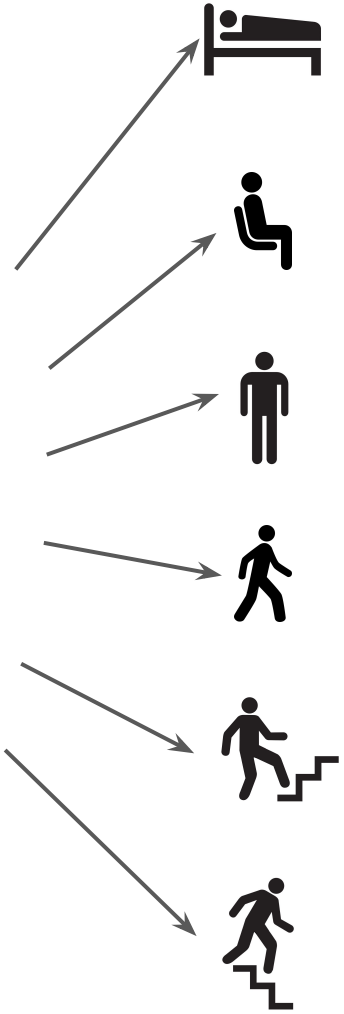
no feature
engineering
required



train SVM & predict



train NN & predict



Dataset and Feature Engineering

Dataset

- Data was gathered for 30 subjects, aged 19-48 years old. Each subject performed 6 activities while wearing a smartphone (Samsung Galaxy S2) on their waist: laying, sitting, standing, walking, walking upstairs, and walking downstairs.
- The smartphone contains an accelerometer and a gyroscope for measuring 3-axial linear acceleration and angular velocity respectively at a constant rate of 50 Hz, which is sufficient for capturing human body motion.
- The data was manually labeled using corresponding video recordings of the experiments. See example here: https://www.youtube.com/watch?v=XOEN9W05_4A&feature=emb_title

Raw Data Preparation

- The sensor signals (accelerometer and gyroscope) were pre-processed by applying noise filters and then sampled in fixed-width sliding windows of 2.56 sec and 50% window overlap (at 50 Hz → 128 readings/window). This “windowed” raw data was used to train the neural network models.
- The accelerometer signal, which has gravitational and body motion components, was separated using a Butterworth low-pass filter into body acceleration and gravity.
- From each window, a vector of features was obtained by calculating variables from the time and frequency domain (feature engineering). This feature vector was used to train the traditional ML model (SVM) as well as for statistical analysis and data visualization.

Feature Engineering

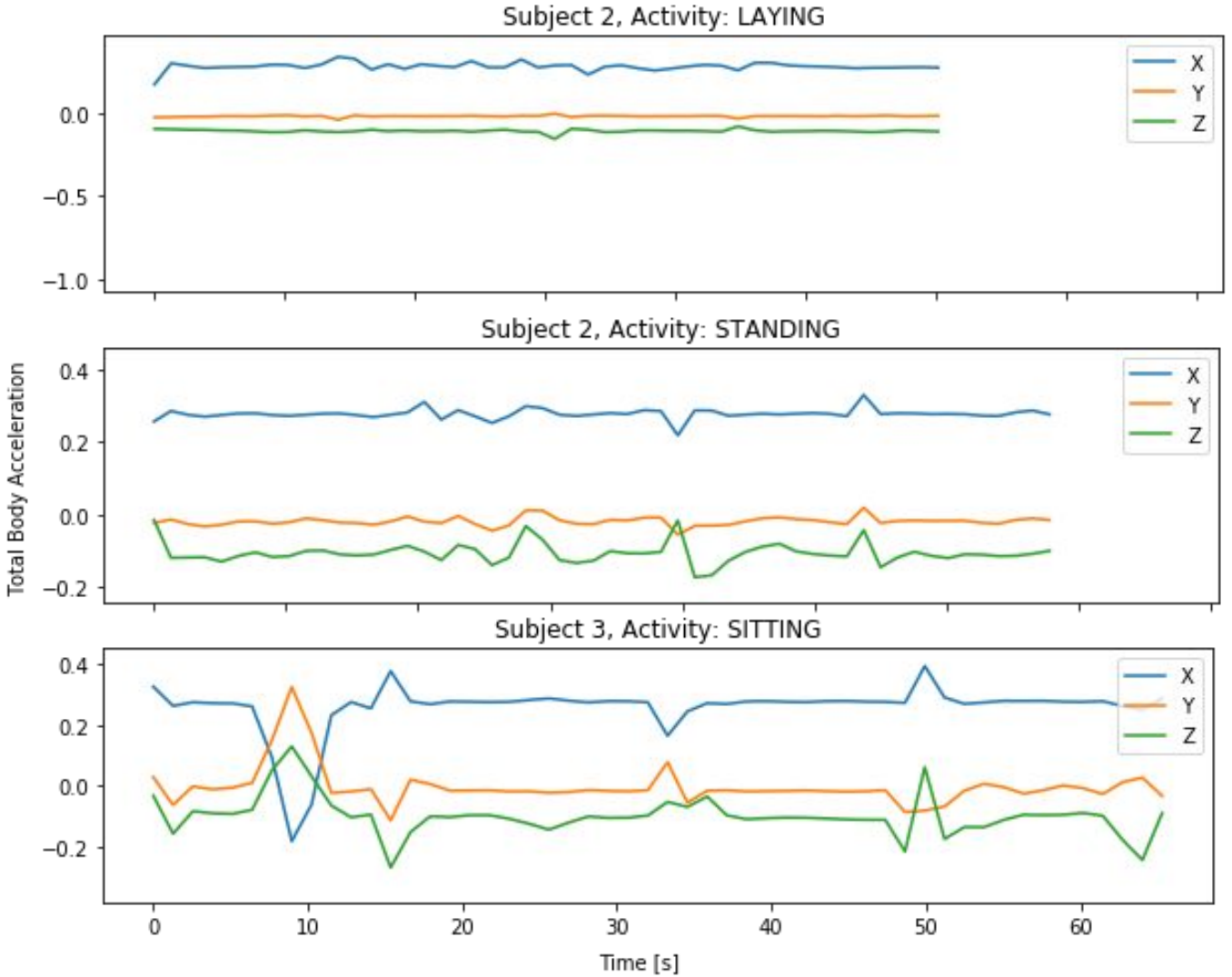
- Transforming the raw signals resulted in the following processed signals:
  (t = total, f = Fast Fourier Transform)

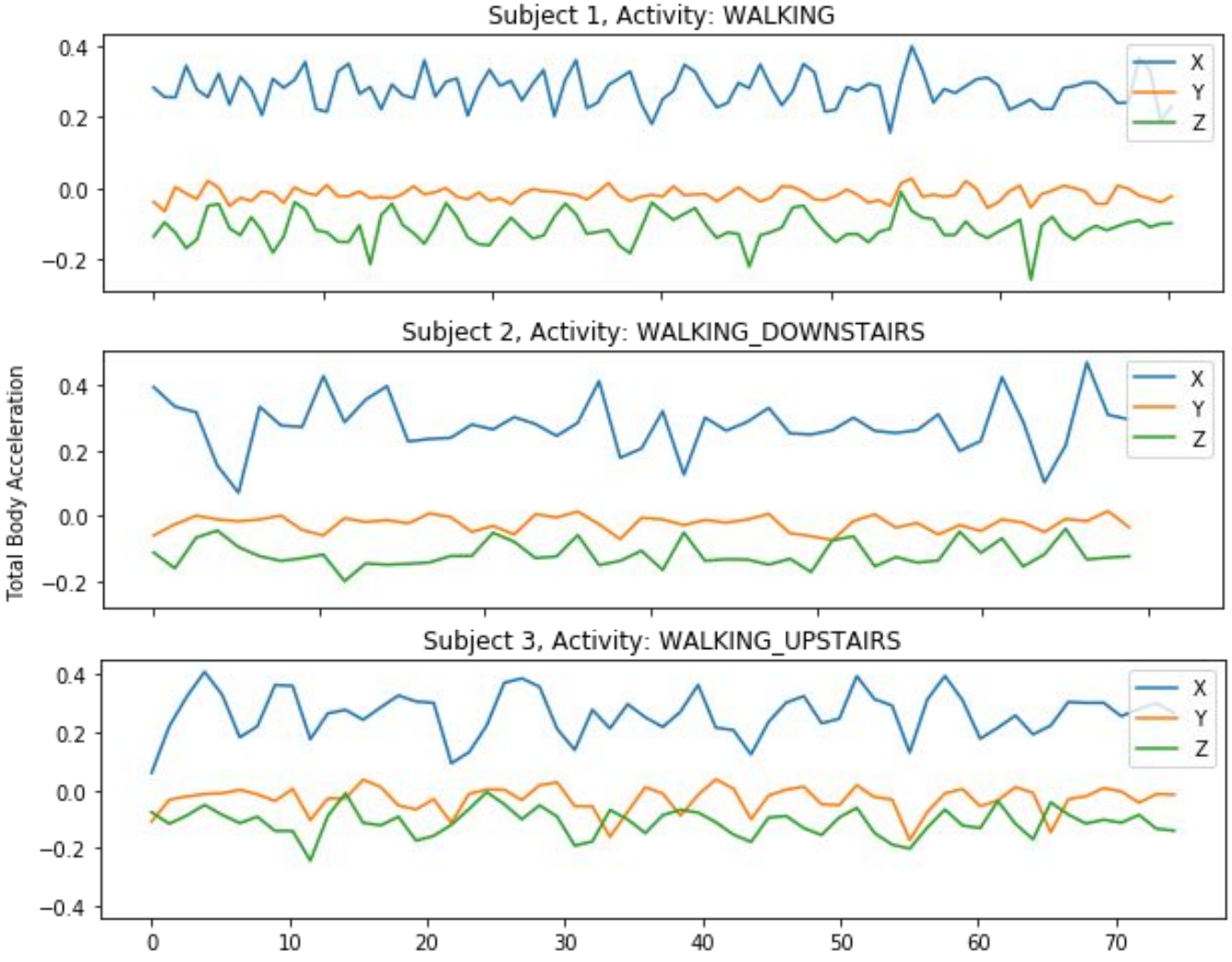
Acc-XYZ
Gyro-XYZ

- tBodyAcc-XYZ
- tGravityAcc-XYZ
- tBodyAccJerk-XYZ
- tBodyGyro-XYZ
- tBodyGyroJerk-XYZ
- tBodyAccMag
- tGravityAccMag
- tBodyAccJerkMag
- tBodyGyroMag
- tBodyGyroJerkMag

- fBodyAcc-XYZ
- fBodyAccJerk-XYZ
- fBodyGyro-XYZ
- fBodyAccMag
- fBodyAccJerkMag
- fBodyGyroMag
- fBodyGyroJerkMag

Example Processed Signal Data (Total Body Acceleration)


Subject 2, Activity: LAYING
Subject 2, Activity: STANDING
Subject 3, Activity: SITTING
X
Y
Z
Total Body Acceleration
Time [s]
0.0
−0.5
−1.0
0.4
0.2
0.0
−0.2
0
10
20
30
40
50
60


Subject 1, Activity: WALKING
Subject 2, Activity: WALKING_DOWNSTAIRS
Subject 3, Activity: WALKING_UPSTAIRS
Total Body Acceleration
X
Y
Z
0.4
0.2
0.0
−0.2
−0.4
0
10
20
30
40
50
60
70

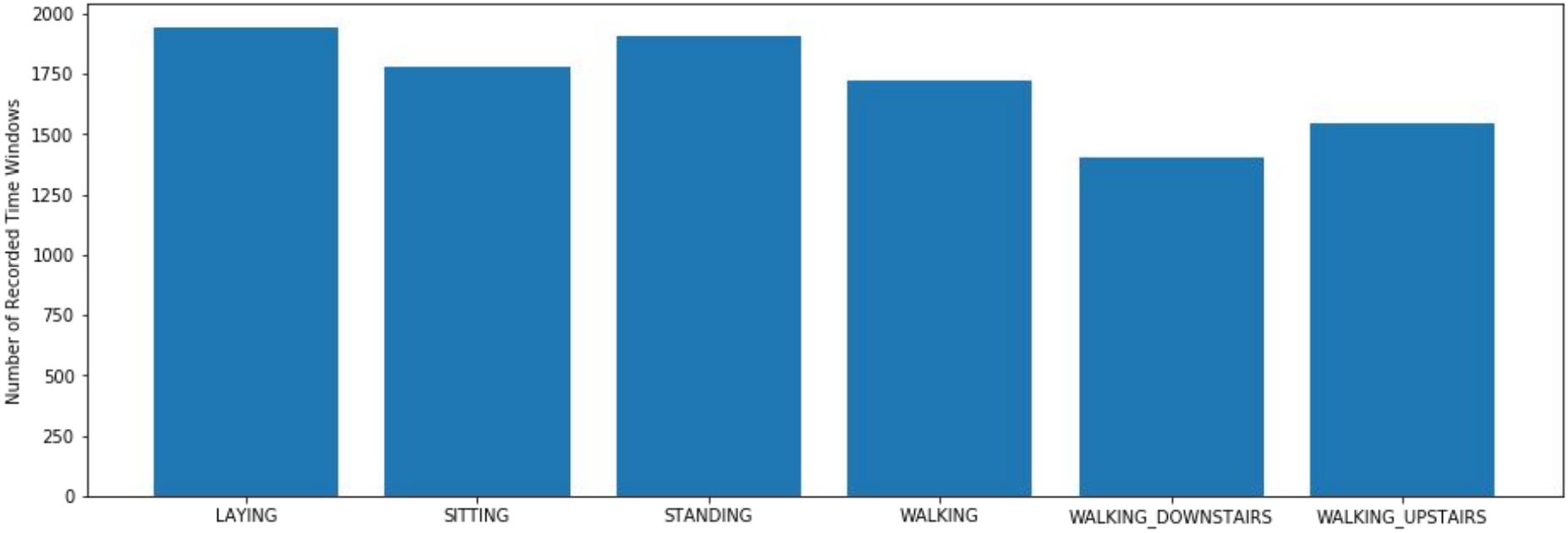
Feature Engineering (cont'd)

Each of the 33 processed signals was reduced to a several feature values (per window) using one or more of the following functions. This resulted in a feature vector of length 561 for each time window.

- mean(): Me.an value
- std(): Standard deviation
- mad(): Median absolute deviation
- max(): Largest value in array
- min(): Smallest value in array
- sma(): Signal magnitude area
- energy(): Energy measure. Sum of the squares divided by the number of values.
- iqr(): Interquartile range
- entropy(): Signal entropy
- arCoeff(): Autorregresion coefficients with Burg order equal to 4
- correlation(): correlation coefficient between two signals
- maxInds(): index of the frequency component with largest magnitude
- meanFreq(): Weighted average of the frequency components to obtain a mean frequency
- skewness(): skewness of the frequency domain signal
- kurtosis(): kurtosis of the frequency domain signal
- bandsEnergy(): Energy of a frequency interval within the 64 bins of the FFT of each window.
- angle(): Angle between two vectors.

“Windowed” Activity Classes

- Each activity has approximately between 36-72 windows (~ 1-2 minutes of recorded activity data) total, each with a corresponding “class label”
- The distribution of activity classes across the dataset is approximately even.

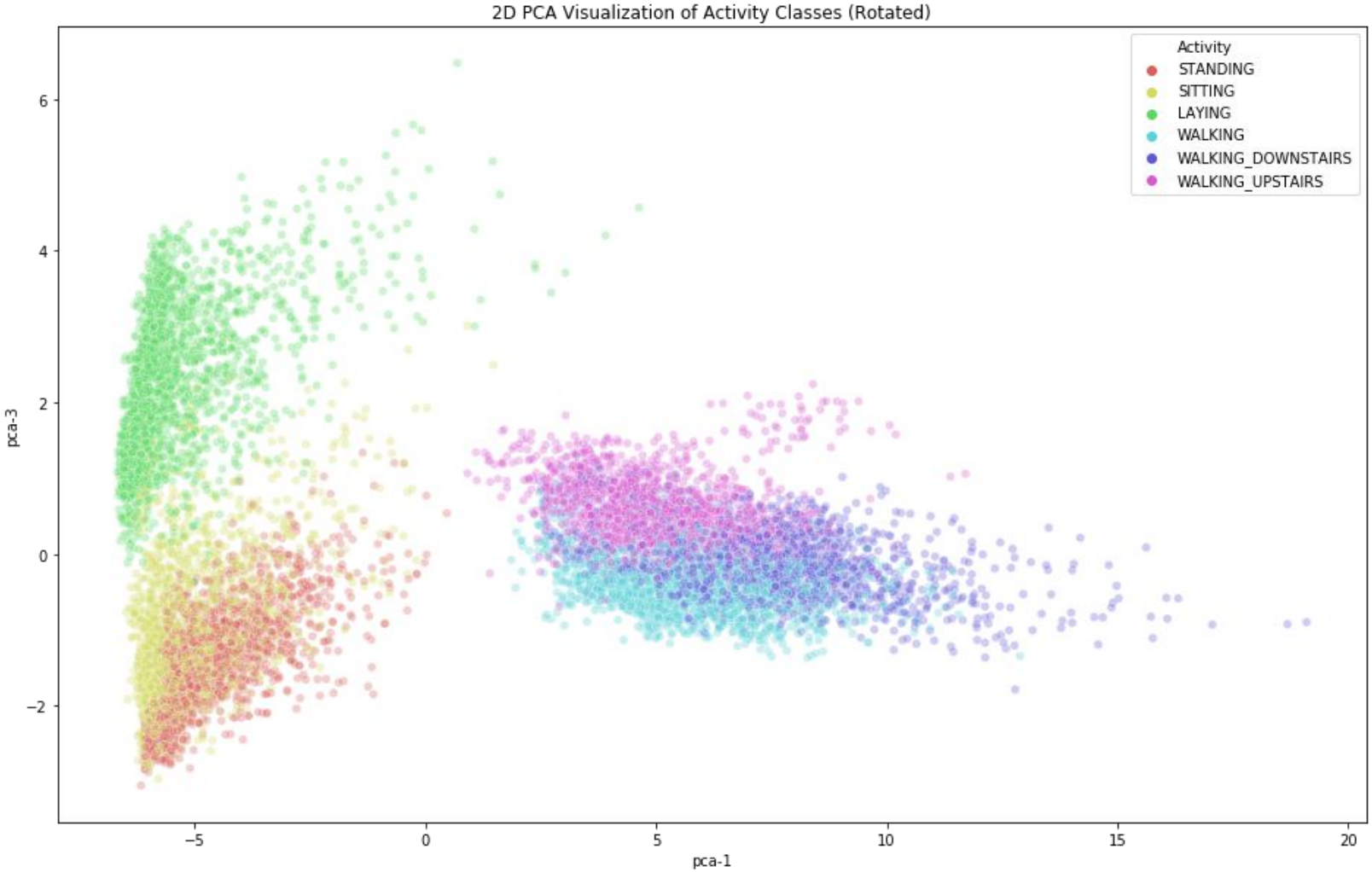


Exploratory Data Analysis / Statistical Inference

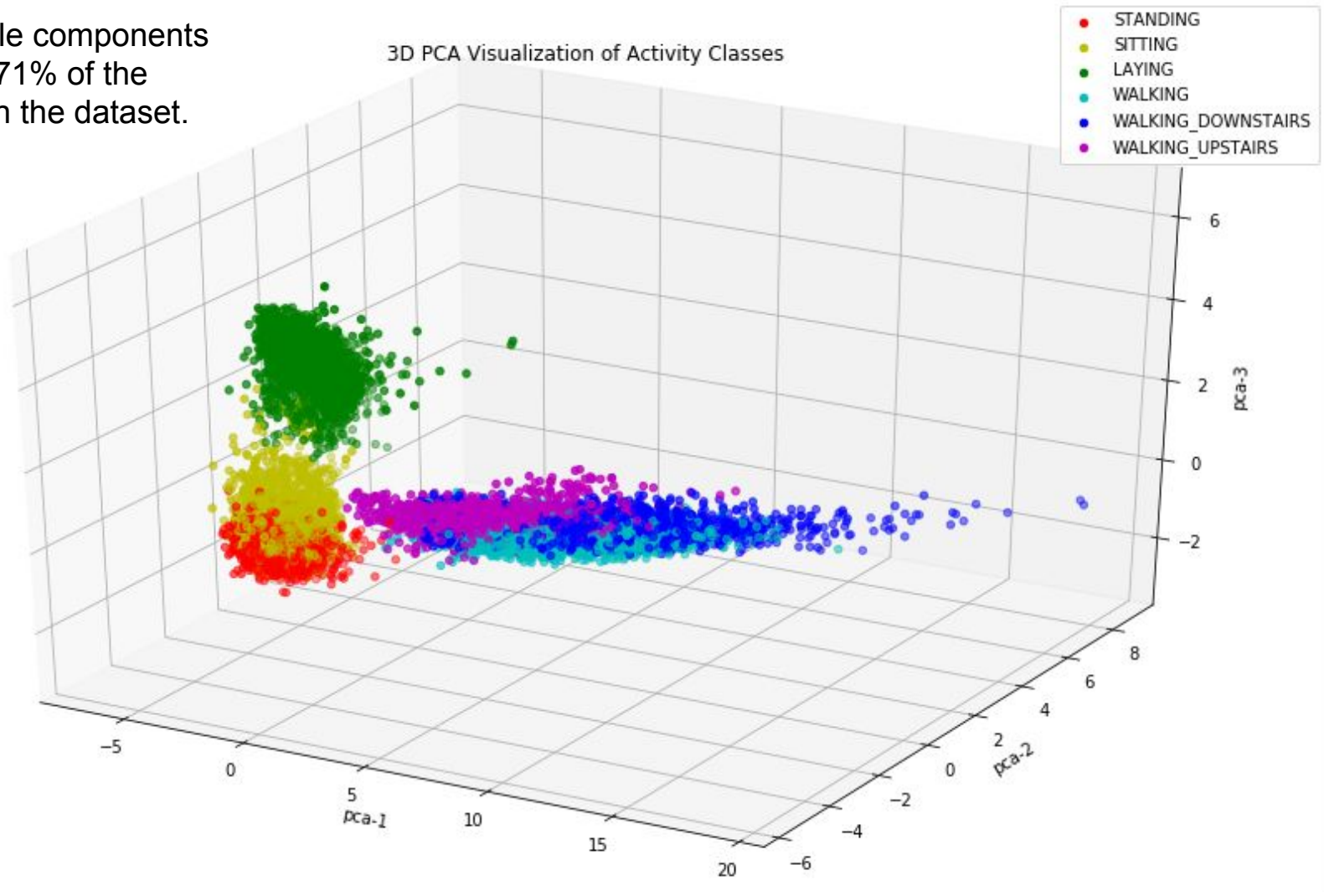
Visualizing Class Separability

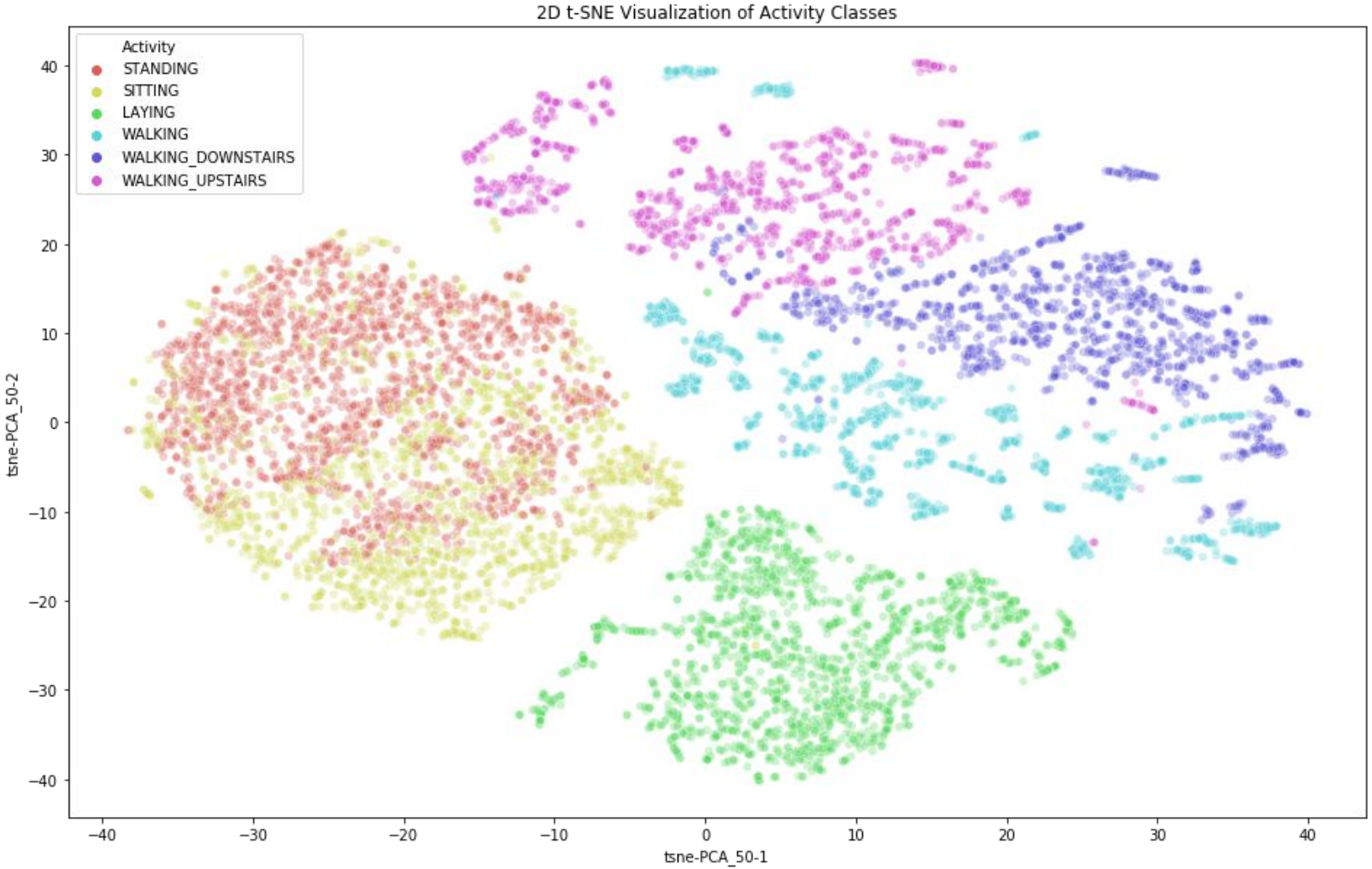
Two dimensionality reduction methods will be implemented to visualize class separability using the 561-feature vector data:

- Principal component analysis (PCA) uses the eigenvectors of the covariance matrix to reduce the dimensionality of the data. The eigenvectors point in the directions of maximum variation in a dataset, preserving as much of the variability in the data in as few dimensions as possible.
- t-Distributed Stochastic Neighbor Embedding (t-SNE) is a probabilistic technique for dimensionality reduction, which minimizes the divergence between two distributions: a distribution that measures pairwise similarities of the input objects and a distribution that measures pairwise similarities of the corresponding low-dimensional points in the embedding.


2D PCA Visualization of Activity Classes (Rotated)
Activity
STANDING
SITTING
LAYING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
pca-3
pca-1
6
4
2
0
−2
−5
0
5
10
15
20

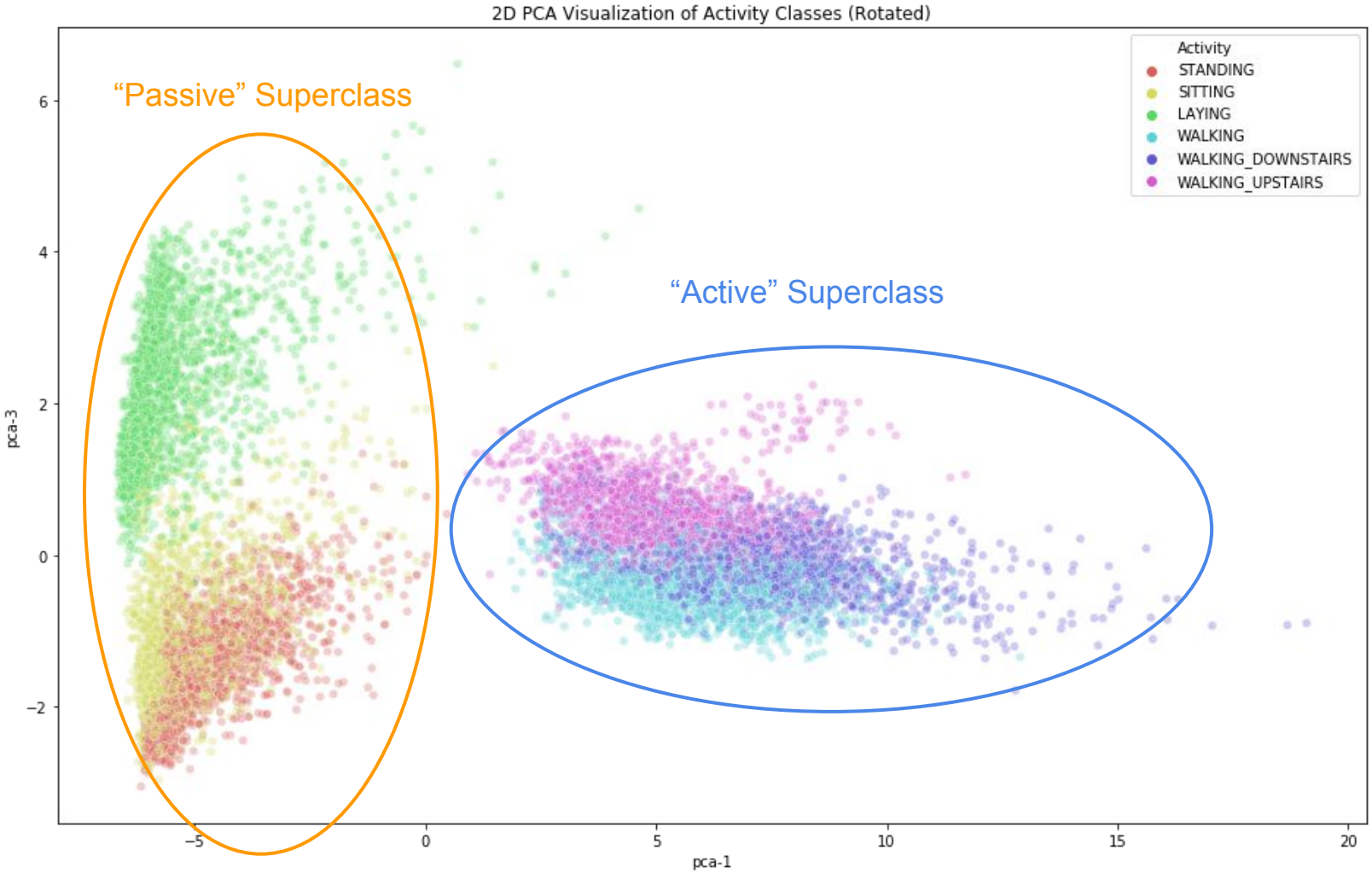
The first 3 principle components represent nearly 71% of the overall variation in the dataset.




2D t-SNE Visualization of Activity Classes
Activity
STANDING
SITTING
LAYING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
tsne-PCA_50-2
tsne-PCA_50-1
40
30
20
10
0
−10
−20
−30
−40
−40
−30
−20
−10
0
10
20
30
40

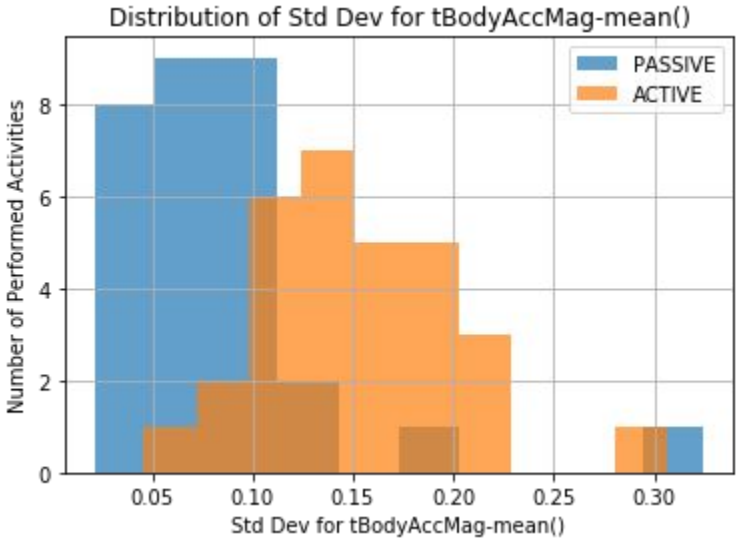
Class Separability - Statistical Inference

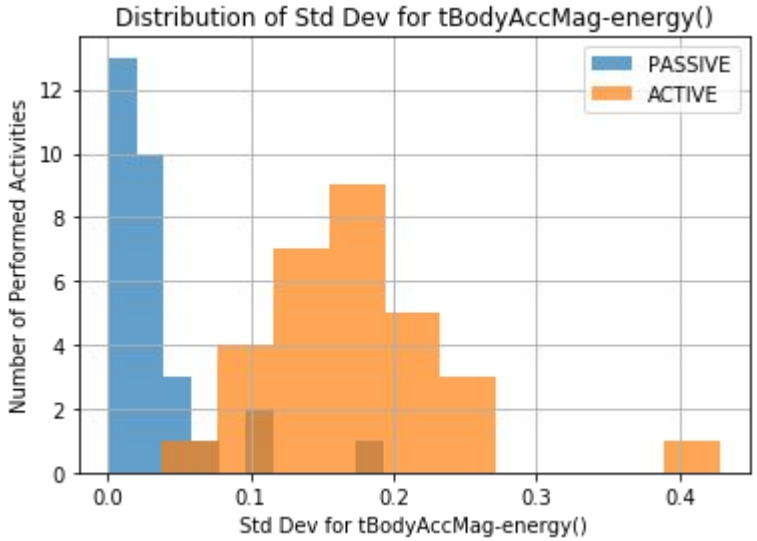
- With PCA offering better interpretability than t-SNE, as well as being faster and deterministic, it was the better option for the visual analysis.
- The classes do appear to be quite distinct, with some overlap. This is promising, but it would be helpful to test this separability with statistical inference.
- It can be observed that the “active” and “passive” classes are more distinguishable as super-clusters containing the individual activity class clusters. For the statistical analysis, each observation has a “superclass” label.
- The feature vector data is also reduced by one more dimension (time). Each activity for a given subject is summarized (time, mean, median, std, max, min) across the full activity time range for all features. These summary attributes are analyzed.


2D PCA Visualization of Activity Classes (Rotated)
“Passive” Superclass
“Active” Superclass
Activity
STANDING
SITTING
LAYING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
pca-3
pca-1
6
4
2
0
−2
−5
0
5
10
15
20

Superclass Distributions

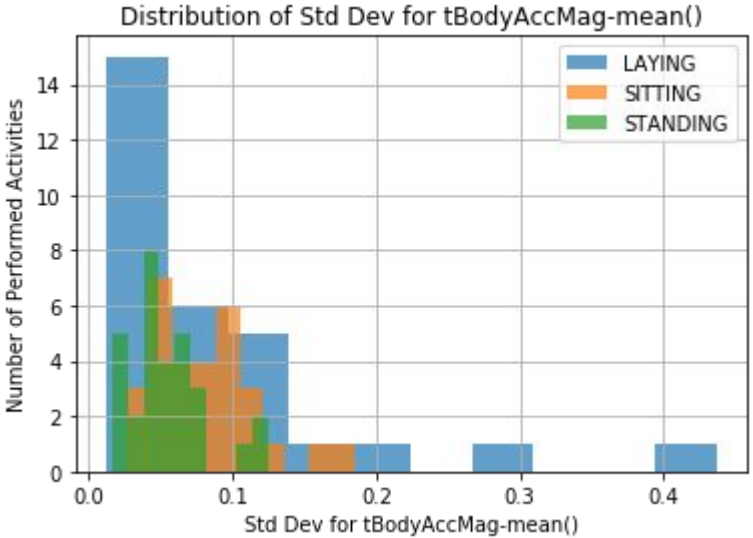
- tBodyAccMag-mean() is the total body acceleration magnitude, averaged over each 2.56 s time window
- tBodyAccMag-energy() the total body acceleration magnitude energy measure
- The distributions plotted below are of standard deviation of these signal features (during a full activity time range) across 30 subjects performing the same activity class.

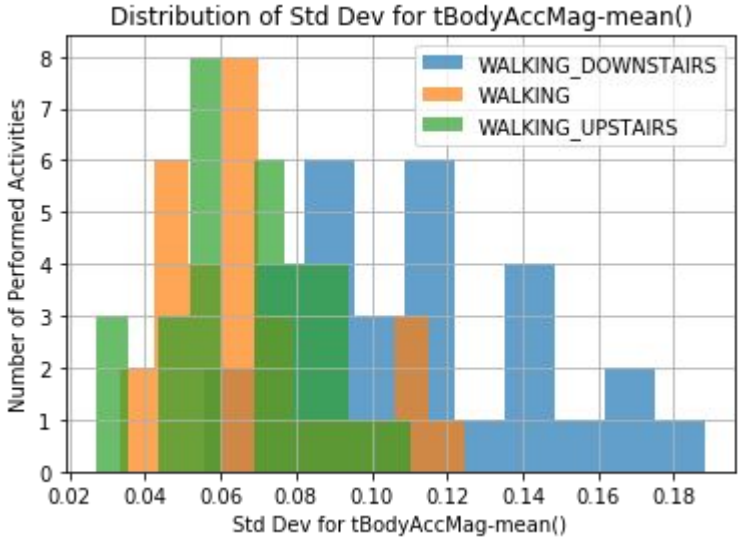




Class Distributions

The class distributions are less distinct than the superclass distributions, but still provide some insight into class separability.





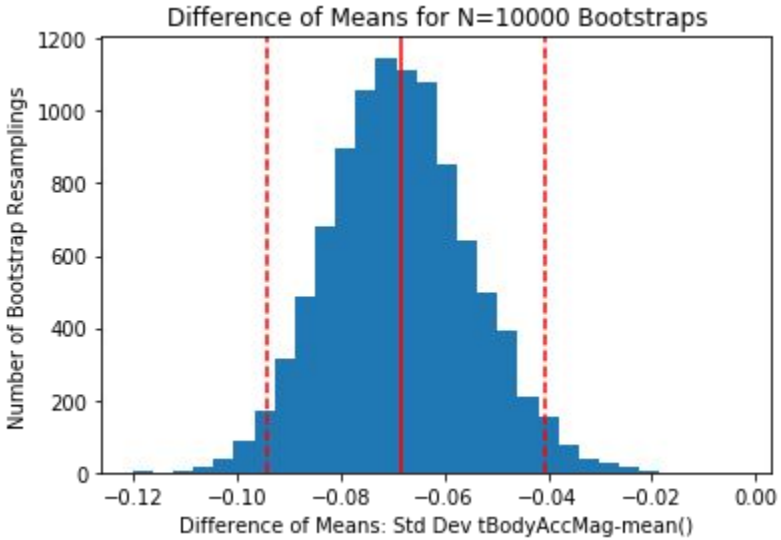
Bootstrap Hypothesis Testing

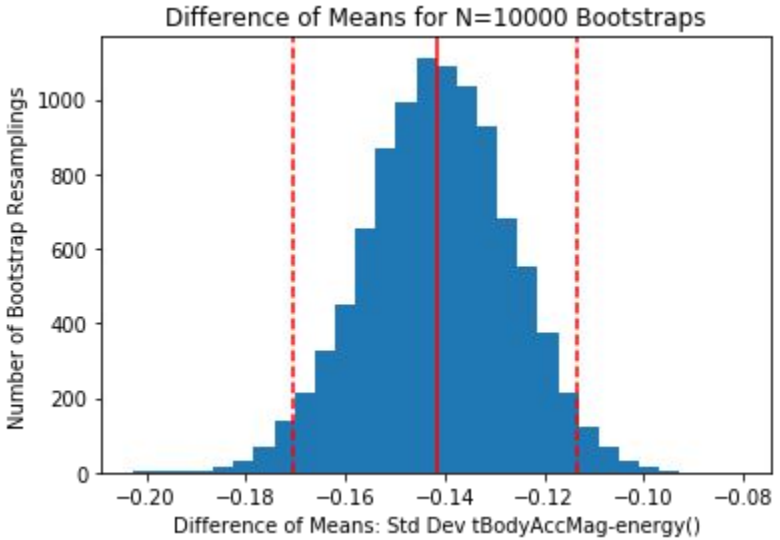
To test whether such distributions are statistically siginificantly different, they can be resampled using a bootstrap method and then tested with a “difference of means” hypothesis test. For example:

- H_0: The difference of means for the standard deviation distribution of tBodyAccMag-mean() between passive and active classes is equal to zero.
- H_A: The difference of means for the standard deviation distribution of tBodyAccMag-mean() between passive and active classes is not equal to zero.

If the 95% confidence interval for the difference of means does not contain 0, the null hypothesis H_0 may be rejected, indicating a statistically significant difference between the distributions.

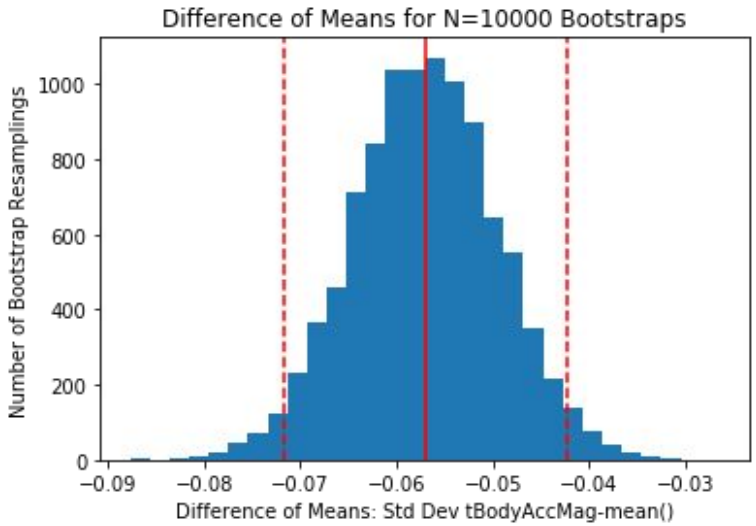
Bootstrap Hypothesis Testing - Superclass

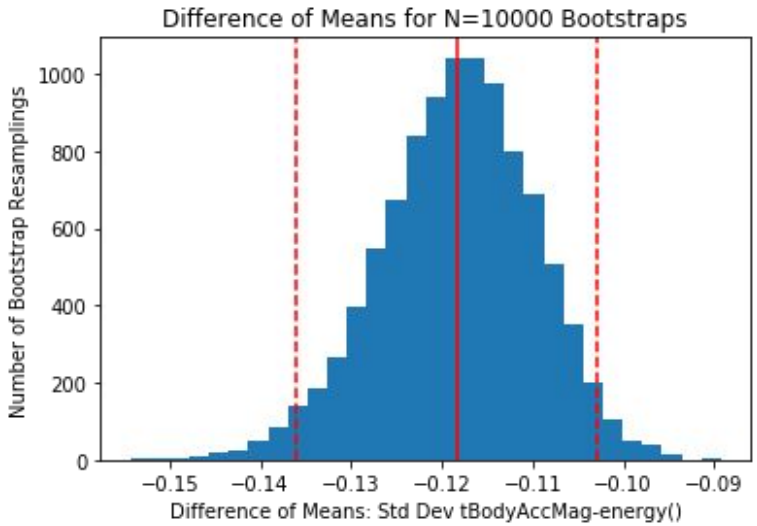




The null hypothesis H_0 may be rejected, indicating a statistically significant difference between the passive and active superclass distributions at 95% significance level.

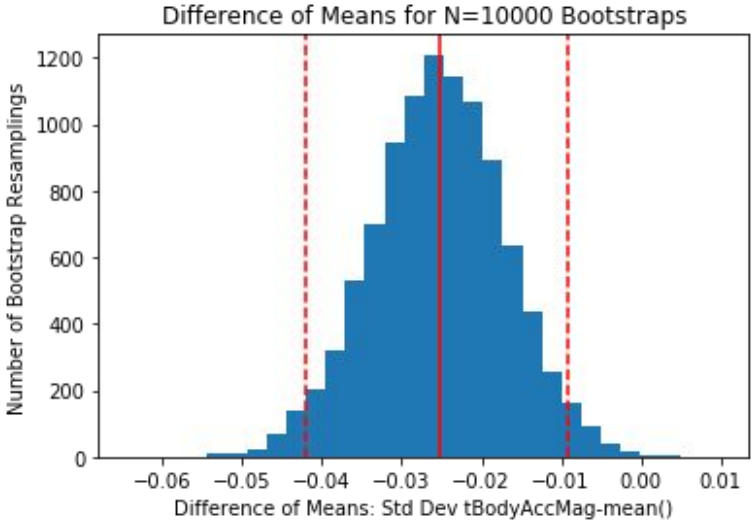
Bootstrap Hypothesis Testing - Dissimilar Classes

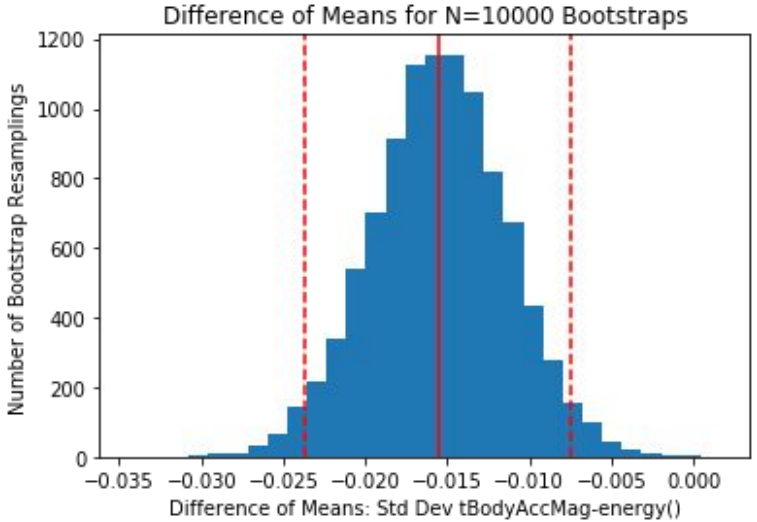




The null hypothesis H_0 may be rejected, indicating a statistically significant difference between the STANDING and WALKING_DOWNSTAIRS class distributions at 95% significance level.

Bootstrap Hypothesis Testing - Similar Classes





The null hypothesis H_0 may be rejected, indicating a statistically significant difference between the STANDING and SITTING class distributions at 95% significance level. This is surprising as these classes overlapped quite a bit in the PCA visualization.

Statistical Conclusions

- These bootstrap hypothesis tests confirm the high class separability that was seen in the PCA and t-SNE visualizations, even between “similar” classes such as STANDING and SITTING.
- This makes the dataset a strong candidate for success using machine learning classifiers such as SVM which will be implemented next.

Implementing and Comparing Machine Learning Models

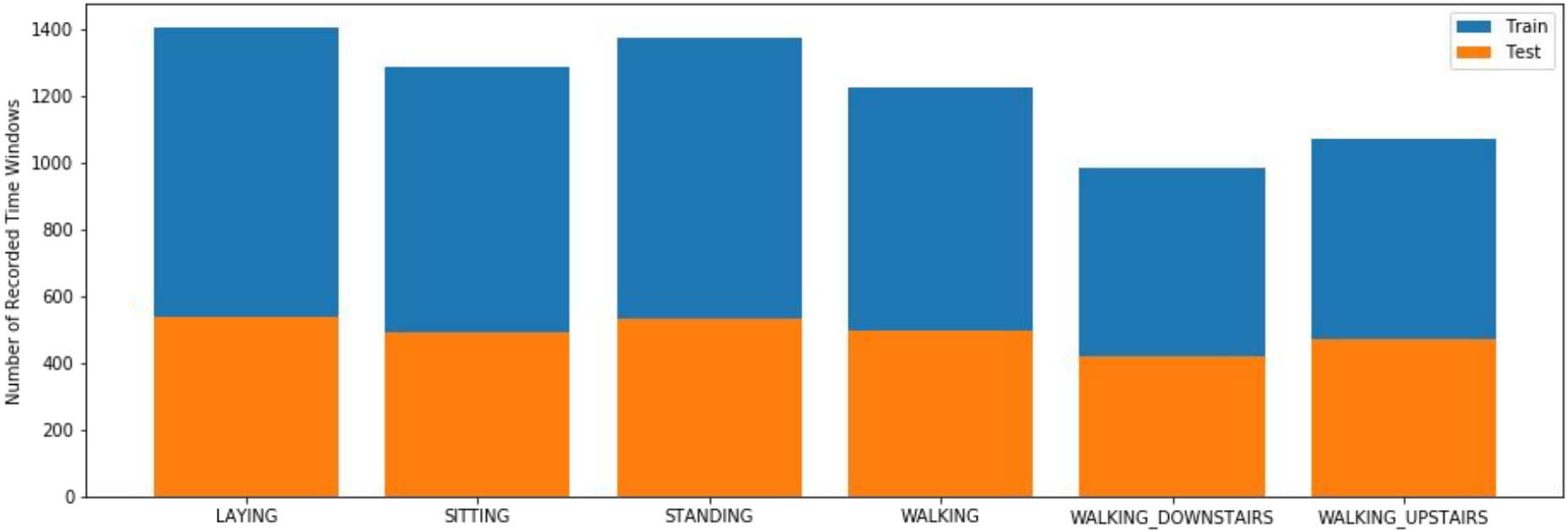
Machine Learning Overview

- This is a supervised (labeled training data) classification problem.
- Both traditional machine learning (SVM) and neural network approaches are be implemented and compared.
  - The SVM is trained on the feature engineered data.
  - Neural networks are trained on the raw “windowed” signal data.
- For all preprocessing and model hyperparameter tuning steps, the Scikit-Learn Python library is utilized.
  - The SVM is also implemented using this library.
- For neural network implementation and evaluation, a Keras-wrapped TensorFlow backend is utilized.

Data Preprocessing: Train/Test Split

- First, the full data set is split into a training set (used to train the ML model) and a test set (used to evaluate the model's performance).
- In this case, a stratified split is important (i.e. making sure that the classes are represented in both the training and test sets relatively equally).
- Also, it must be taken into account that entire activities and subject sets should be kept together. Movement patterns of the same subject should be segregated so as not to bias the test set.
- Therefore of the 30 participating subjects' data, 30% (9 subjects) will be segregated into the test set and saved for the evaluation phase. These subjects have been selected randomly.

Data Preprocessing: Train/Test Split



Data Preprocessing - Feature Scaling

- To improve performance of the machine learning model(s) to be applied to this classification problem, first the feature data should be appropriately scaled.
- Scaling must be performed both on the training and test data sets in the same manner. Best practice is to use only the training set to identify the correct scaling, and then blindly apply the same transformation to the test set.
- In this case, the feature matrix will be standardized (i.e. the mean of each feature set to zero and the std dev set to 1) using the sklearn method StandardScaler().
    - Note MinMaxScaler() with ranges of [-1,1] or [0,1] were also tested, but these resulted in poorer model performance.

Data Preprocessing - Class Label Encoding

Since the class labels (Activity column) for this data are in a categorical string format, they must first be encoded to a numerical format useable for supervised machine learning. Neural networks require an additional one-hot encoding step.

SVM Classifier

Laying	→	0
Sitting	→	1
Standing	→	2
Walking	→	3
Walking Upstairs	→	4
Walking Downstairs	→	5

Neural Network Classifier

Laying	→	1 0 0 0 0 0
Sitting	→	0 1 0 0 0 0
Standing	→	0 0 1 0 0 0
Walking	→	0 0 0 1 0 0
Walking Upstairs	→	0 0 0 0 1 0
Walking Downstairs	→	0 0 0 0 0 1

Hyperparameter Tuning

- In order to optimize performance of the models, a grid search across several model hyperparameters is performed.
- The GridSearchCV() method is used to scan across hyperparameter combinations using a k-fold cross validation to evaluate a mean model performance for each combination.

SVM Classifier

- Regularization parameter **C**: [1, 10, 1000, 10000]
- Kernel coefficient **gamma**: [1e-6, 1e-4, 1e-2]
- A linear kernel vs. (non-linear) RBF kernel

Neural Network Classifier

- **batch_size** = [32, 64, 128]
  Batch size controls the accuracy of the estimate of the error gradient
- **dropout** = [0.1, 0.3, 0.5]
  Dropout is an effective regularization method to reduce overfitting
- **neurons** = [50, 100, 200]
  The number of neurons in each hidden layer

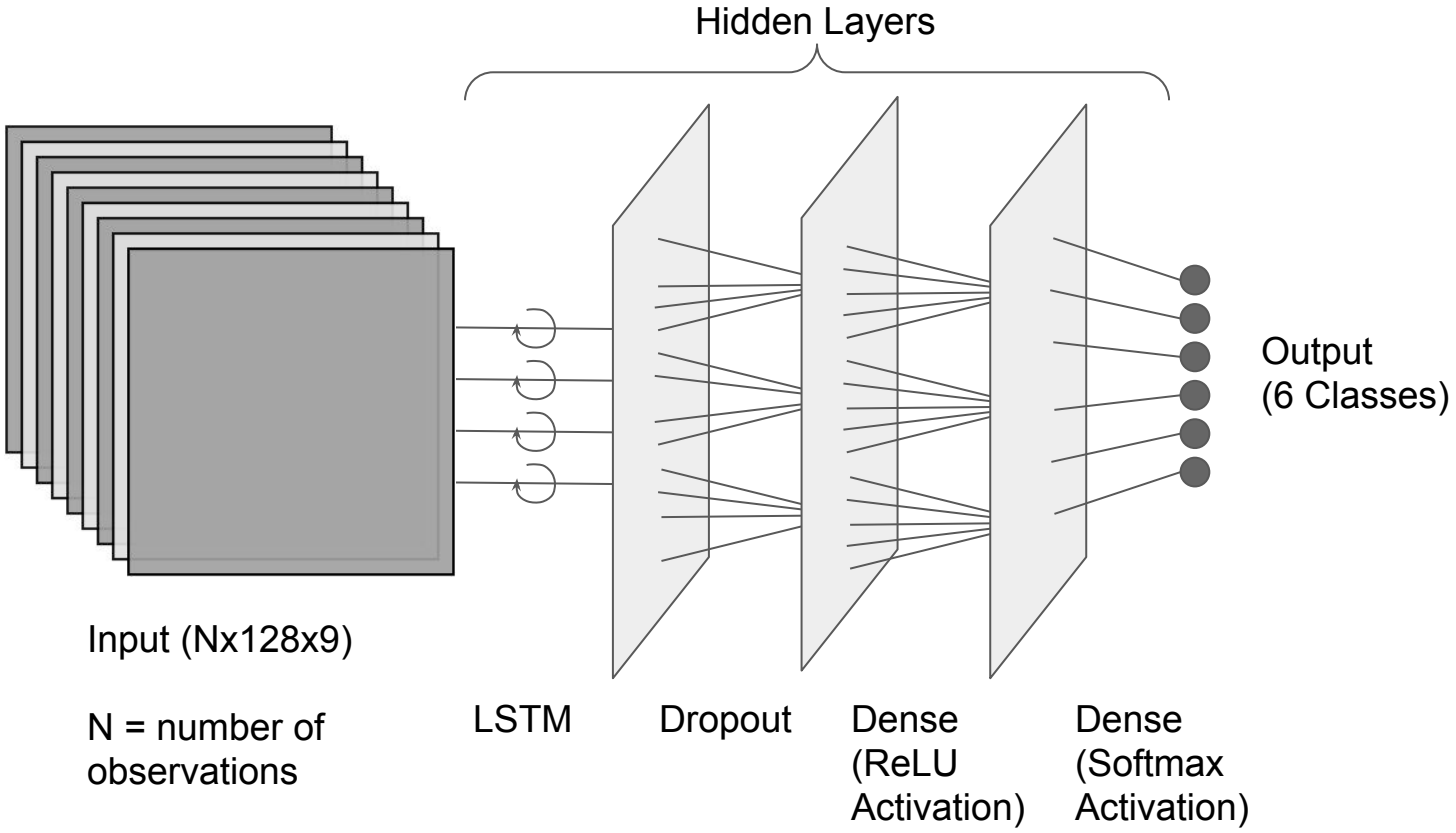
Hyperparameter Optimization

Model	Hyperparameters Selected	Total Trainable Parameters
SVM	C: 1000 gamma: 0.0001 kernel: rbf	561
LSTM	batch_size: 128 dropout: 0.1 neurons: 200	209,406
CNN-LSTM	batch_size: 64 dropout: 0.1 neurons: 200	933,150
ConvLSTM	batch_size: 128 dropout: 0.3 neurons: 50	152,676

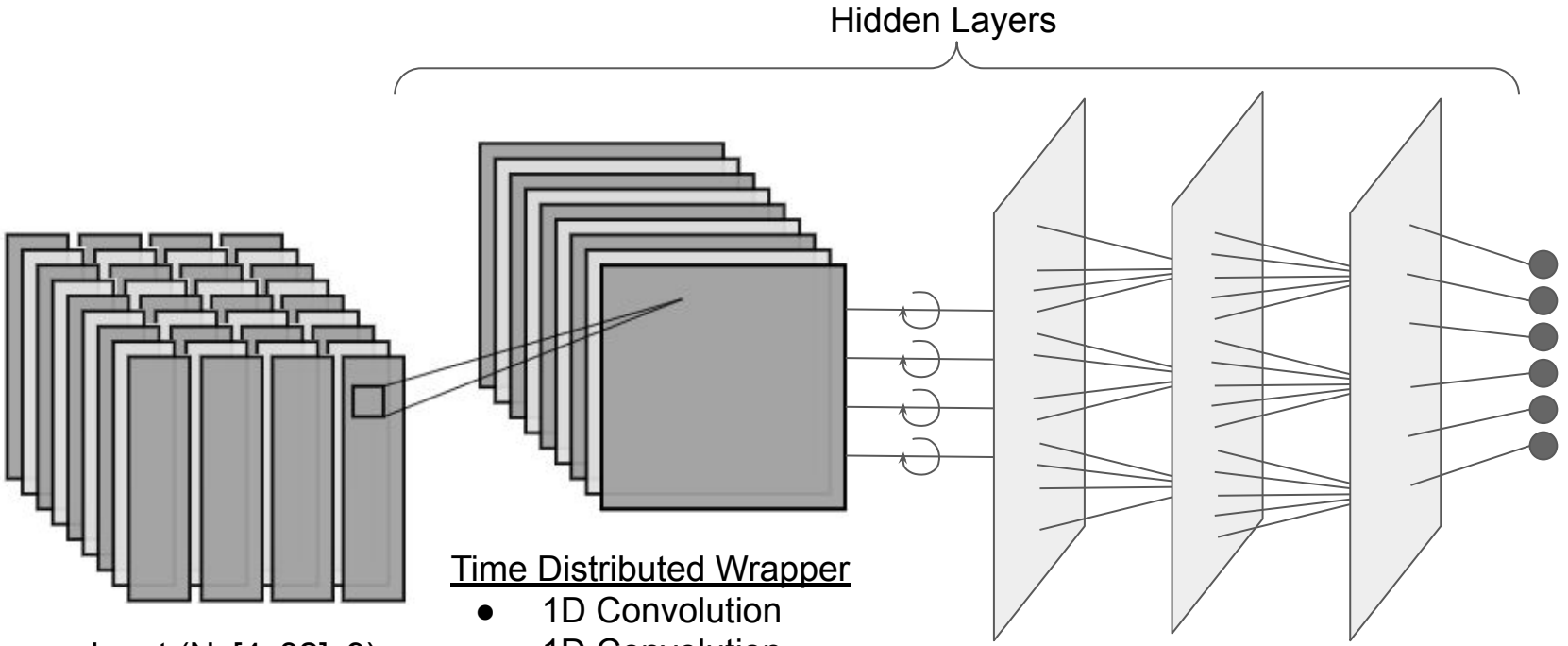
LSTM Recurrent Neural Network and Variations

- The LSTM (long short term memory) layer of a recurrent neural network (RNN), unlike feed forward neural network architectures, has feedback mechanisms that allow previous observations in the sequence to be "remembered" by the network. This works well for sequential signal data.
- Combining the LSTM with a convolutional neural network (CNN) architecture is often implemented in machine learning tasks with input that has both temporal and spatial characteristics, such as sequences of images (video), or sequences of words/sentences in text.
  - The CNN-LSTM architecture uses CNN layer(s) for feature extraction on the raw input data and uses subsequent LSTM layer(s) to support sequence prediction.
  - In ConvLSTM, the matrix multiplication calculation of the input with the LSTM cell is replaced by a convolution operation; the convolution is essentially embedded in the architecture.

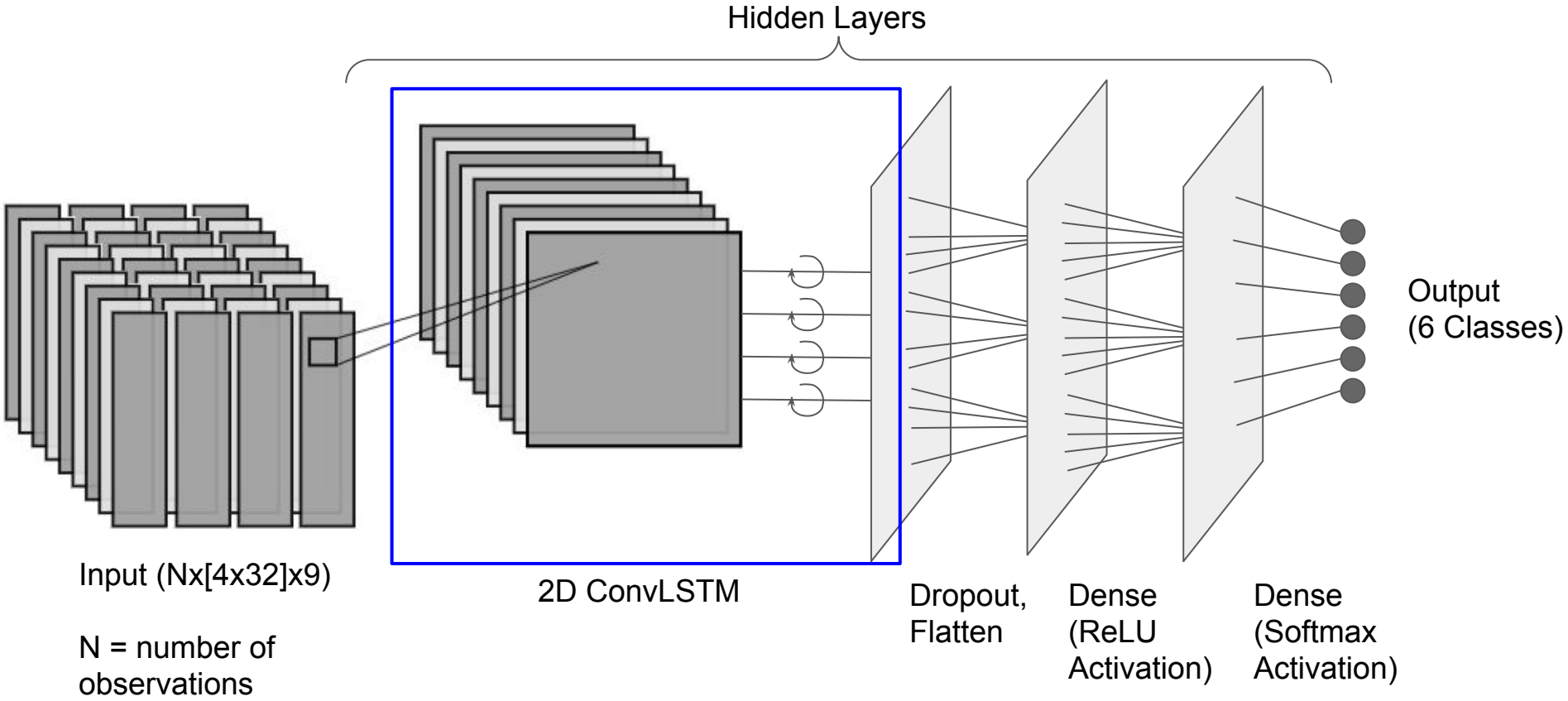
RNN - LSTM Baseline Network Architecture



CNN-LSTM Network Architecture

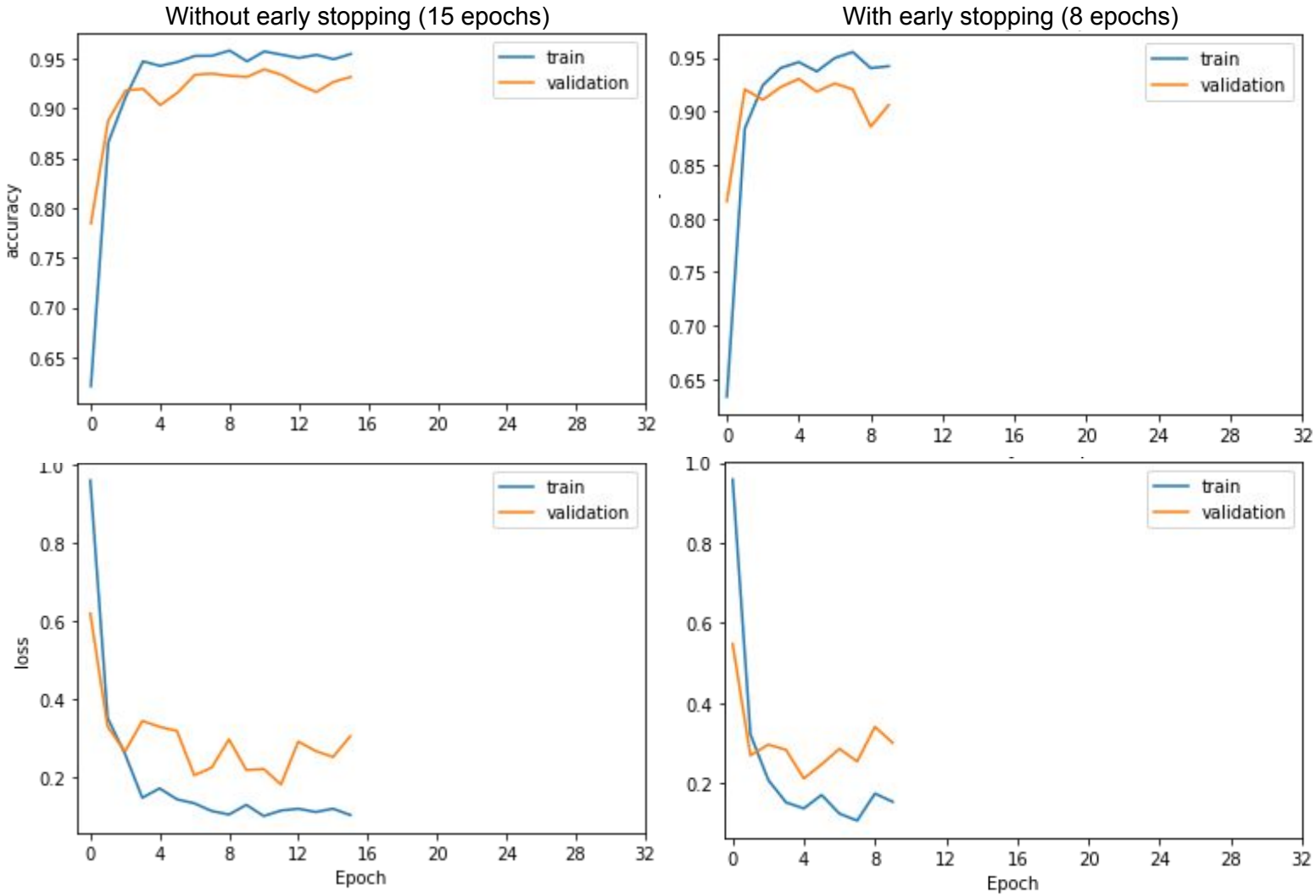


ConvLSTM Network Architecture



LSTM Neural Network

Example Training Histories



Comparing Machine Learning Model Performance

- The SVM (with feature engineered data) had the best performance across the board, both in terms of overall model accuracy and prediction time.
- The best NN model performance was the ConvLSTM model, though there was little variance across all 3 models.

Model	Prediction Time [s]
SVM	1.45
LSTM	2.48
CNN-LSTM	0.74
ConvLSTM	0.63

Model	Average Model Accuracy +/- Std. Dev.
SVM	96.17%
LSTM	91.39% (+/-1.06%)
CNN-LSTM	92.01% (+/-1.27%)
ConvLSTM	92.74% (+/-0.56%)

Classification performance:
Confusion Matrix

SVM Classifier

	LAY	SIT	STA	WALK	WALK_D	WALK_U
LAY	537	0	0	0	0	0
SIT	0	436	54	0	0	1
STA	0	15	517	0	0	0
WALK	0	0	0	493	3	0
WALK_D	0	0	0	5	398	17
WALK_U	0	0	0	16	2	453

ConvLSTM

	LAY	SIT	STA	WALK	WALK_D	WALK_U
LAY	472	6	18	0	0	0
SIT	8	446	17	0	0	0
STA	2	18	400	0	0	0
WALK	0	10	0	399	82	0
WALK_D	0	0	0	76	456	0
WALK_U	0	27	0	0	0	510

Comparing Machine Learning Models

	Benefits	Limitations
SVM	• lightweight, deterministic ML model • can be quickly trained and deployed • results in very highly accurate and repeatable classification.	• requires extensive feature engineering, which can be computationally expensive to achieve • code more cumbersome to update if changes on the signal processing side are required
NNs	• require no feature engineering • prediction times are low enough that they could feasibly be used for real-time classification	• large number of parameters result in longer training and prediction times • Unpredictable classification repeatability -- NNs are non-deterministic

Conclusions

- As personalized computing devices become smaller and faster, the race to implement the next generation of real-time activity recognition algorithms will continue.
- The classification problem investigated in this project consisted of classifying data into 6 human activity classes.
    - High class separability was observed with various visualization techniques as well as through statistical inference.
    - High class separability was also observed in the very good classification performance of the traditional machine learning model (SVM) and neural network models (RNN, CNN-LSTM, ConvLSTM) implemented.
- The high classification accuracy achieved can be improved even further to allow for a productionalized version of the model to be implemented.

Ideas for Future Improvement

- Additional subjects, beyond the 30 participants of this study, could likely serve to improve the generalizability across models. Similarly, longer activity periods or multiple activity periods would increase the number of training samples.
- Different window sizes could also be investigated to determine the impact of sequence lengths.
- Improvement for the SVM model could be found with additional model hyperparameter tuning and further feature engineering using additional transformations and statistical functions.
- Improvement for the NN models could be found by adding additional hidden layers and tuning additional hyperparameters, such as filter/pooling sizes for the convolutional layers.

Helpful Links and Resources

All of the code for the analysis and machine learning models in this presentation may be found on GitHub:

- https://github.com/s-weisenberg/Springboard/tree/master/Capstone_Project_2

Original datasets (publicly available):

- Kaggle: Human Activity Recognition with Smartphones
- UCI: Human Activity Recognition Using Smartphones Data Set

Machine learning publications on Human Activity Recognition

- A Public Domain Dataset for Human Activity Recognition Using Smartphones
- CNN Based Approach for Activity Recognition using a Wrist Worn Accelerometer
- Human Activity Recognition on Smartphones using a Multiclass Hardware-Friendly Support Vector Machine
- Divide and Conquer-Based 1D CNN Human Activity Recognition Using Test Data Sharpening
- Deep, Convolutional, and Recurrent Models for Human Activity Recognition using Wearables

Excellent online guides and tutorials for implementing RNNs on Human Activity Recognition data:

- Human Activity Recognition (HAR) Tutorial with Keras and Core ML
- Deep Learning Models for Human Activity Recognition
- How to Develop RNN Models for Human Activity Recognition Time Series Classification

Thank you!

Please feel free to email me with any questions or comments:
sofy.weisenberg@gmail.com