CMPUT 466/566 Mini - Project

Name: Vaibhav Chugh
Student ID: 1575034

Introduction

In this project, I'm implementing 4 multi-class classification models on the Human Activity Recognition Using Smartphones (HAR) data set with a trivial majority-guess model as a baseline. The data set is obtained from Kaggle *(1)* and its information is obtained from the UCI repository *(2)*.

The data set is built from the recordings of 30 subjects within an age bracket of 19-48 years performing daily-living activities while carrying a waist-mounted smartphone (Samsung Galaxy S II) with embedded inertial sensors. Each person performed six activities - WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING and LAYING. Using the smartphone's embedded accelerometer and gyroscope, 3-axial linear acceleration and 3-axial angular velocity were captured at a constant rate of 50Hz. The experiments have been video-recorded to label the data manually *(3)*.

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% overlap (128 readings/window). The sensor acceleration signal, which has gravitational and body motion components, was separated using a Butterworth low-pass filter into body acceleration and gravity. The gravitational force is assumed to have only low frequency components, therefore a filter with 0.3 Hz cutoff frequency was used. From each window, a vector of features was obtained by calculating variables from the time and frequency domain.

The objective is to train different models to classify an input reading into one of the six activities performed and choose the model with the best performance metric.

Data Pre-Processing

For each record we have the following features:

- Triaxial acceleration from the accelerometer (total acceleration) and the estimated body acceleration.
- Triaxial Angular velocity from the gyroscope.
- A 561-feature vector with time and frequency domain variables corresponding to the readings above.
- An identifier of the subject who carried out the experiment
- Activity Name

The main preprocessing and feature extraction is already done in the given data set. Features are already normalized and bounded within [-1,1]. Each feature vector is a row on the csv files given as the data set. The units used for the accelerations (total and body) are 'g's (gravity of earth -> 9.80665 m/seg2) and the gyroscope units are rad/seg.

To perform the classification task, we need to add an additional feature which converts the activity names to their corresponding labels (1 through 6 for the six activities) which is our target variable. So for each row/input, we have a feature vector of length 564 which is our input to the model.

Total number of samples is 10,299. The obtained data set has already been randomly partitioned into two sets, where 70% of the volunteers were selected for generating the training data and 30% the test data.

Shape of training and test data:

Train.shape: (7352, 564)
Test.shape: (2947, 564)

Shape of input X features and y targets:

X_train and y_train: (7352, 561) and (7352,)
X_test and y_test: (2947, 561) and (2947,)

Note: 3 features were dropped in input data as they weren't playing a role in the classification task namely "subject", "Activity" and our target variable "ActivityLabel". Additionally, to clean the data, I checked for duplicate and null values but there weren't any.

Exploratory Data Analysis

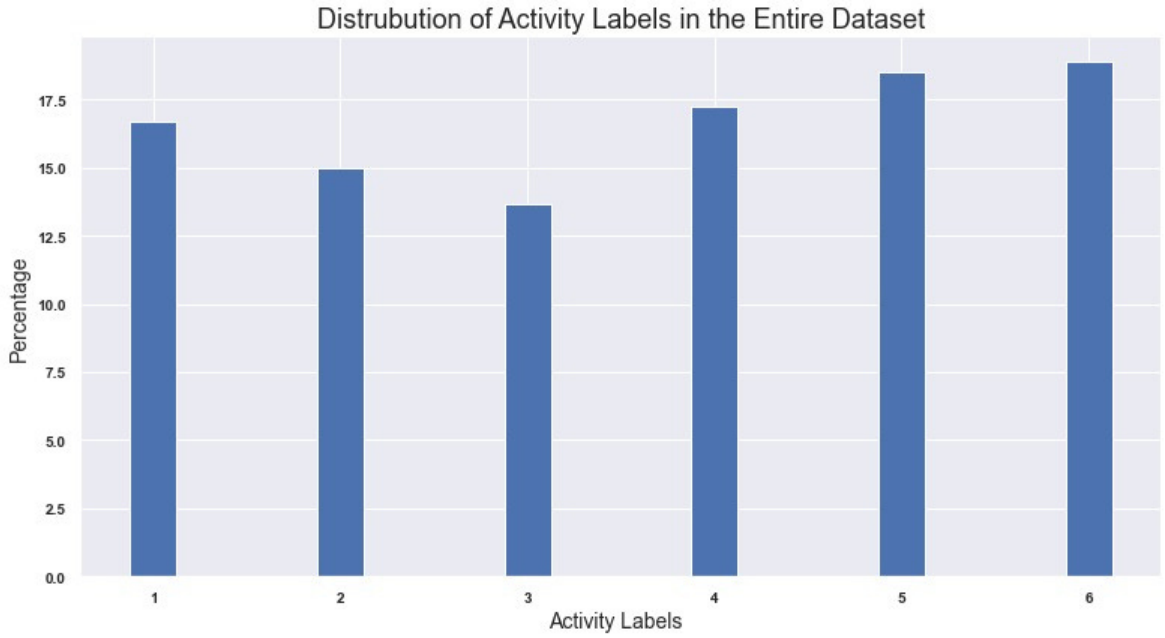
Activity names and their associated labels

WALKING	1
WALKING_UPSTAIRS	2
WALKING_DOWNSTAIRS	3
SITTING	4
STANDING	5
LAYING	6

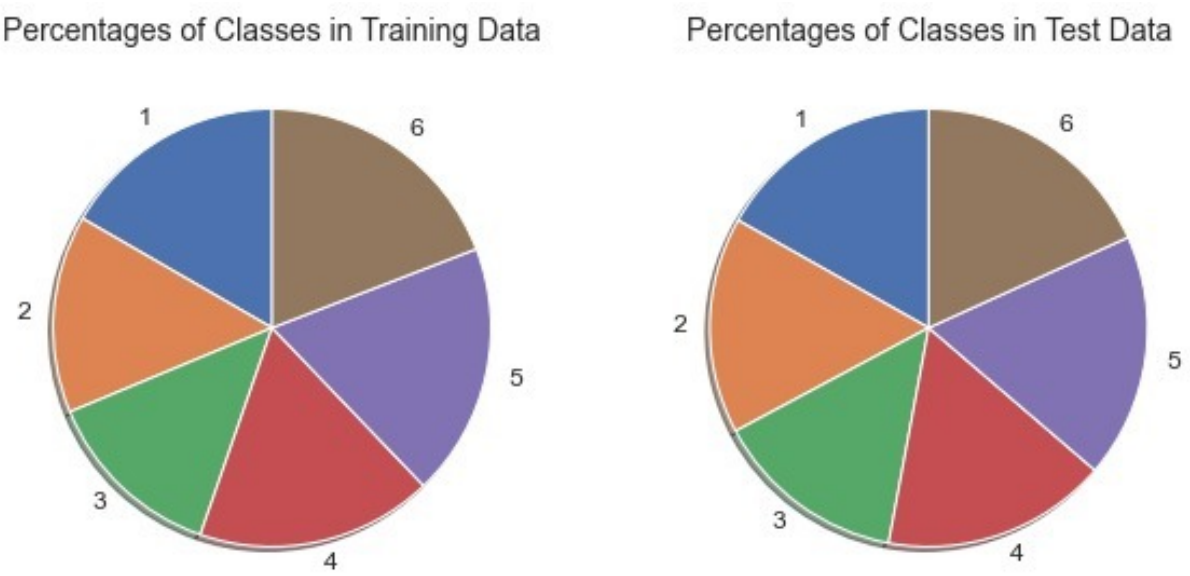
Data Balance

To check if the data is balanced, I plotted a bar plot for the entire data set and pie charts *(4)* for training and test data to analyze the distribution of samples for each activity.

Bar Plot



Pie Chart



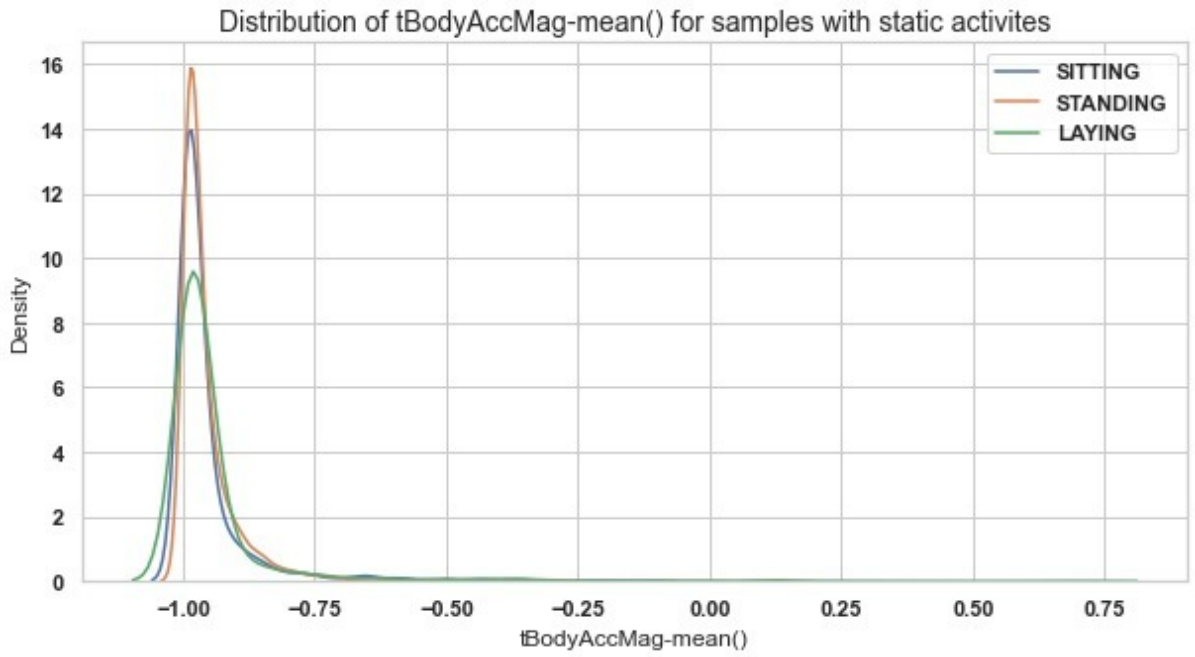
As we can see the data distribution is balanced

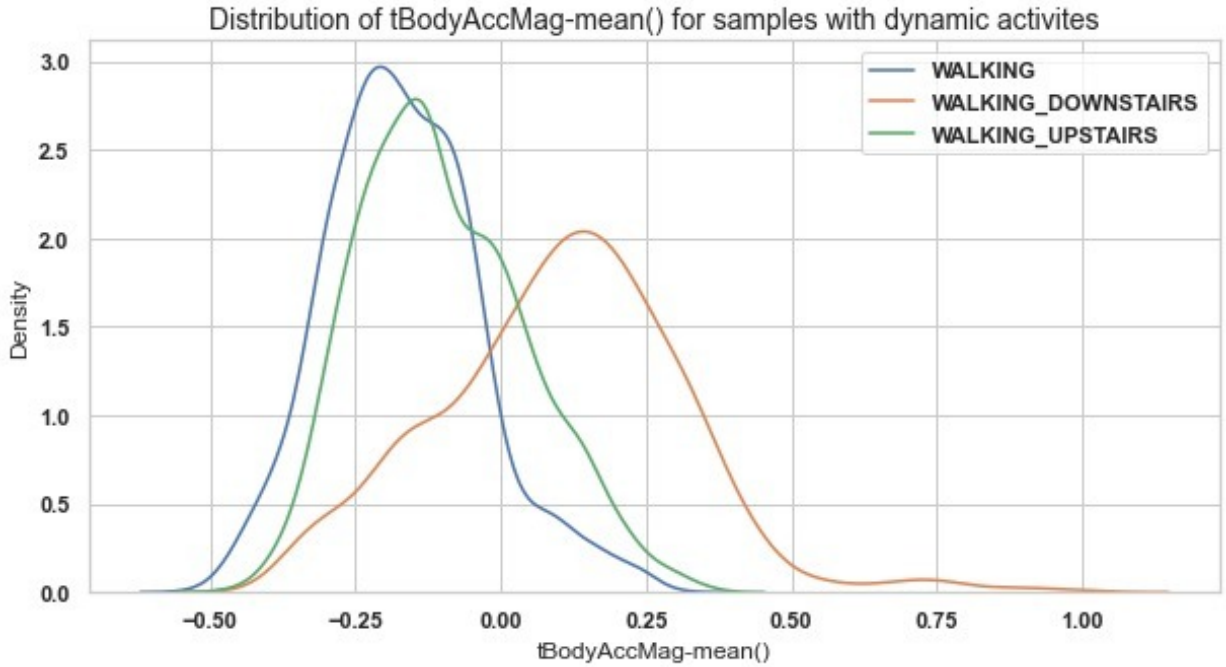
Static and Dynamic Activities Analysis:

From the 6 activity classes given to us, we can classify them into static and dynamic activities:

Static: SITTING, STANDING, LAYING
Dynamic: WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS

Exploring the distribution of static and dynamic samples by plotting average reading of the accelerometer from smartphones using density plots *(5)*:





As we can see the dynamic distributions are more varied and have a lower density as compared to the static distributions.

Models

According to our exploratory data analysis balance charts, although the data is balanced, we can see that the majority of the targets belong to activity label 6 i.e. "LAYING". So as a trivial baseline model, we will implement a model predicting the majority guess for all the test data i.e. predicting 6 for the test data.

For the machine learning algorithms I implemented the following algorithms:

1.Softmax Regression

Softmax regression *(6)* is a form of logistic regression that normalizes an input value into a vector of values that follows a probability distribution whose total sums up to 1. It is used for multi-class classification and it uses the softmax function and cross-entropy loss in its training objective.

To implement this algorithm, I used the Scikit-learn library, specifically the linear_model.LogisticRegression *(7)* class with multinomial as the class parameter to make sure the algorithm is using cross-entropy loss. I used different values for 'C' and 'penalty' as hyperparameters for regularization.

2.RBF Kernel SVM

SVM *(8)* works by mapping data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable. Different SVM algorithms use different types of kernel functions to take data as input and transform it to a different input space. RBF kernels are the most generalized form of kernelization and is one of the most widely used kernels due to its similarity to the Gaussian distribution *(9)*.

To implement this algorithm, I used the Scikit-learn library, specifically the SVC class with 'rbf' as the kernel parameter *(10)*. I used different values for 'C' and 'gamma' *(11,12)* as hyperparameters to ensure regularization and define influence of training examples respectively.

3.Linear SVC

The Linear Support Vector Classifier (SVC) *(13,14,15)* method applies a linear kernel function to perform classification and it performs well with a large number of samples. The objective of Linear SVC is to fit to the data we provide, returning a "best fit" hyperplane that divides, or categorizes, our data.

To implement this algorithm, I used the Scikit-learn library, specifically the LinearSVC class (16) with 0.00005 tolerance as the stopping criteria. I used different values for 'C' values
as hyperparameters to ensure regularization *(17)*.

4.Random Forest Classifier

Random forests *(18,19)* creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance.

To implement this algorithm, I used the Scikit-learn library, specifically the RandomForestClassifier() class *(20)*. For hyperparameters, I used 'n_estimators' and 'max_depth' which specify the number of trees and maximum depth of trees over a varied range of values *(21)*.

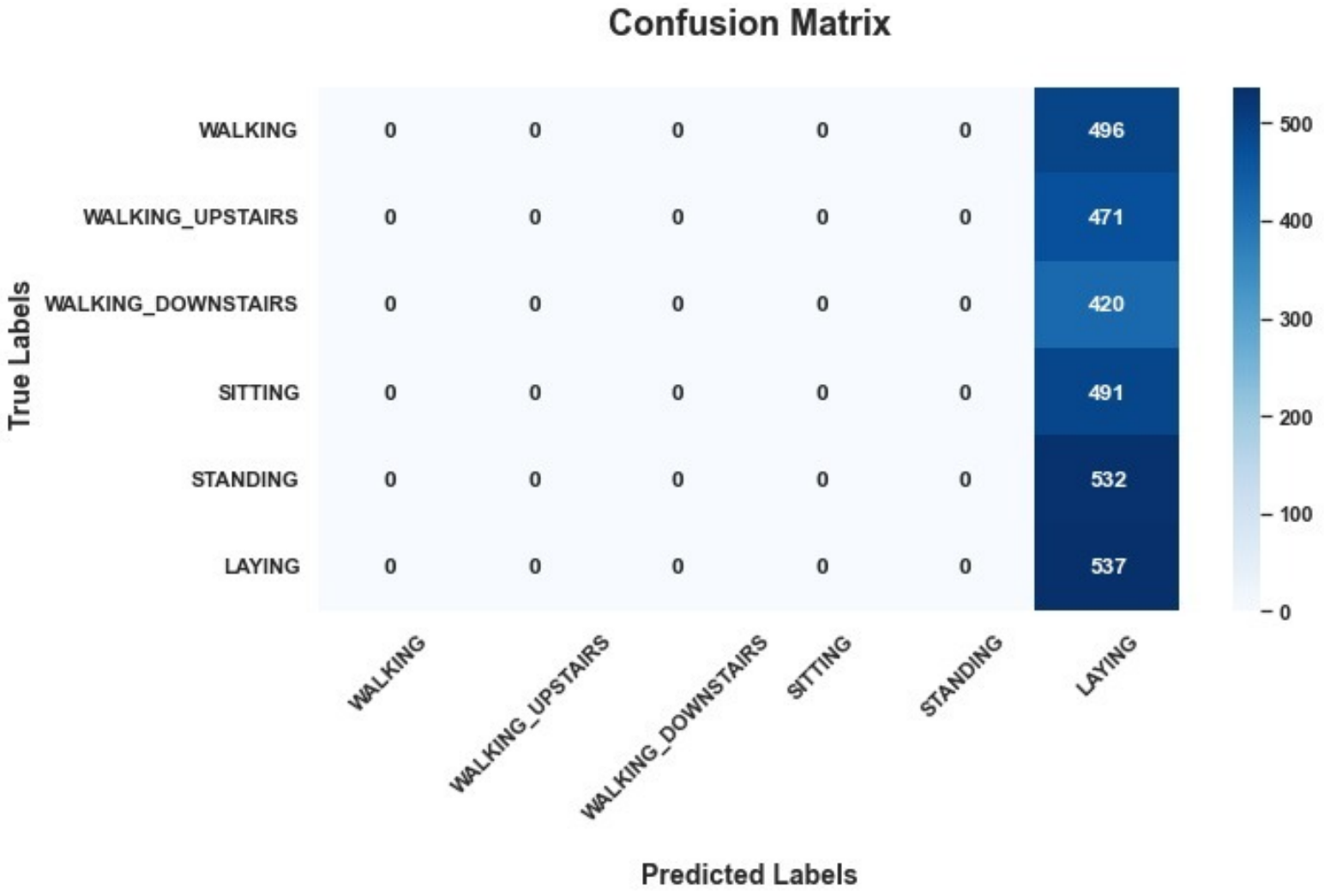
Implementation

The models are trained using the GridSearch *(22,23,24)* train-validation approach for hyperparameter tuning for an exhaustive search over specified parameters values to get the best estimator. For the evaluation metric, since the data is balanced, we can comfortably use accuracy to evaluate the performance of our models.

In the next few pages, you will find an implementation summary and the results for each model that includes hyperparameter values, running time, grid search report, confusion matrix *(25)* and accuracy.

1. Majority-Guess [Baseline]

Predicting 6 for every test case:



Accuracy: 18.22%

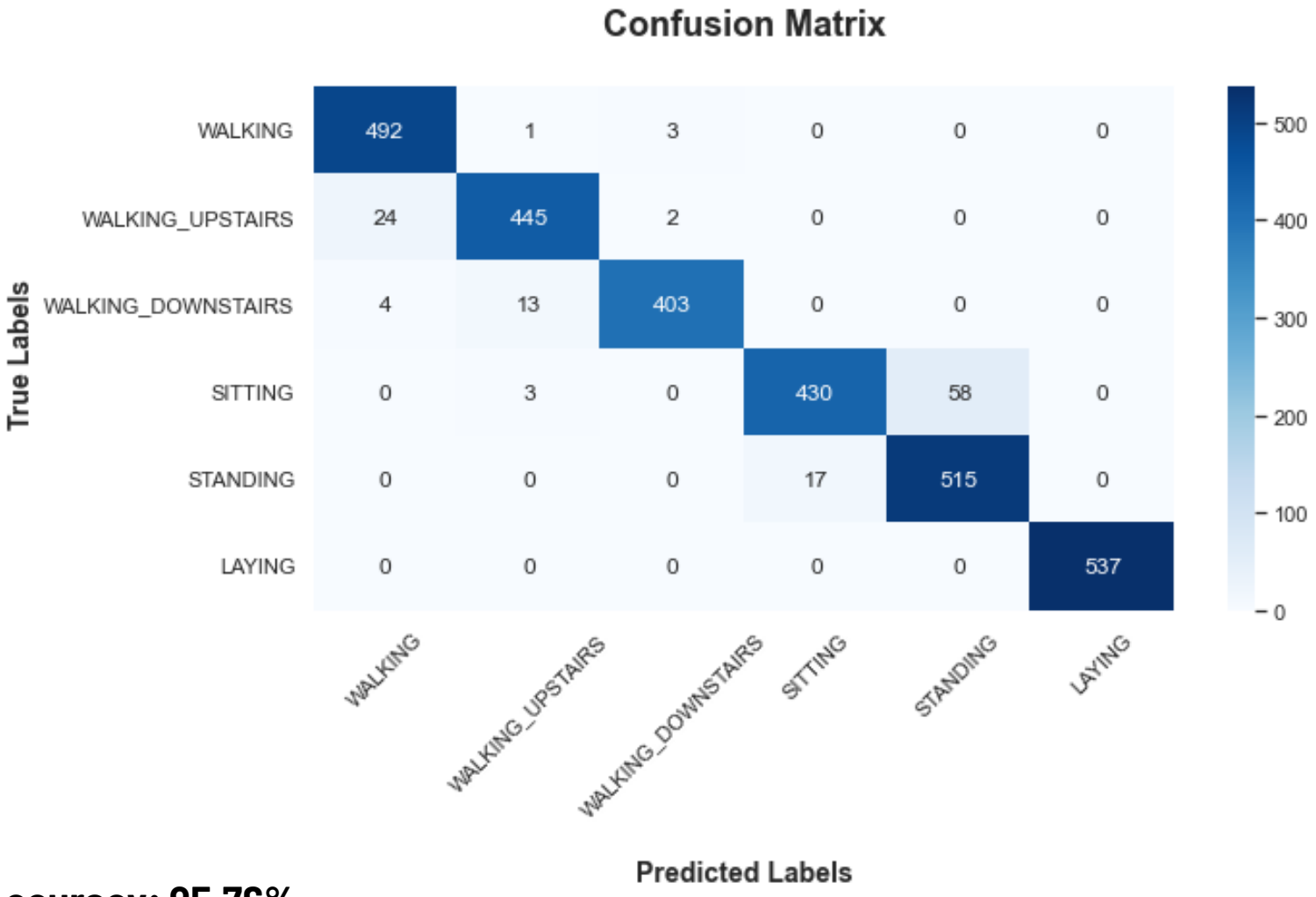
2. Softmax Regression

Hyperparameters:
C : [0.01, 0.1, 1, 10, 20, 30]
Penalty: ['l2','l1']

Training + Validation Time [Hours:Minutes.Seconds]:
00:09.23

Testing Time:
00:00.008

GridSearch Report:
Best Estimator: LogisticRegression(C=1, multi_class='multinomial', verbose=False)
Best parameters: {'C': 1, 'penalty': 'l2'}
Total number of CrossValidation sets: 3
Best Score: 0.9366175682839435



Accuracy: 95.76%

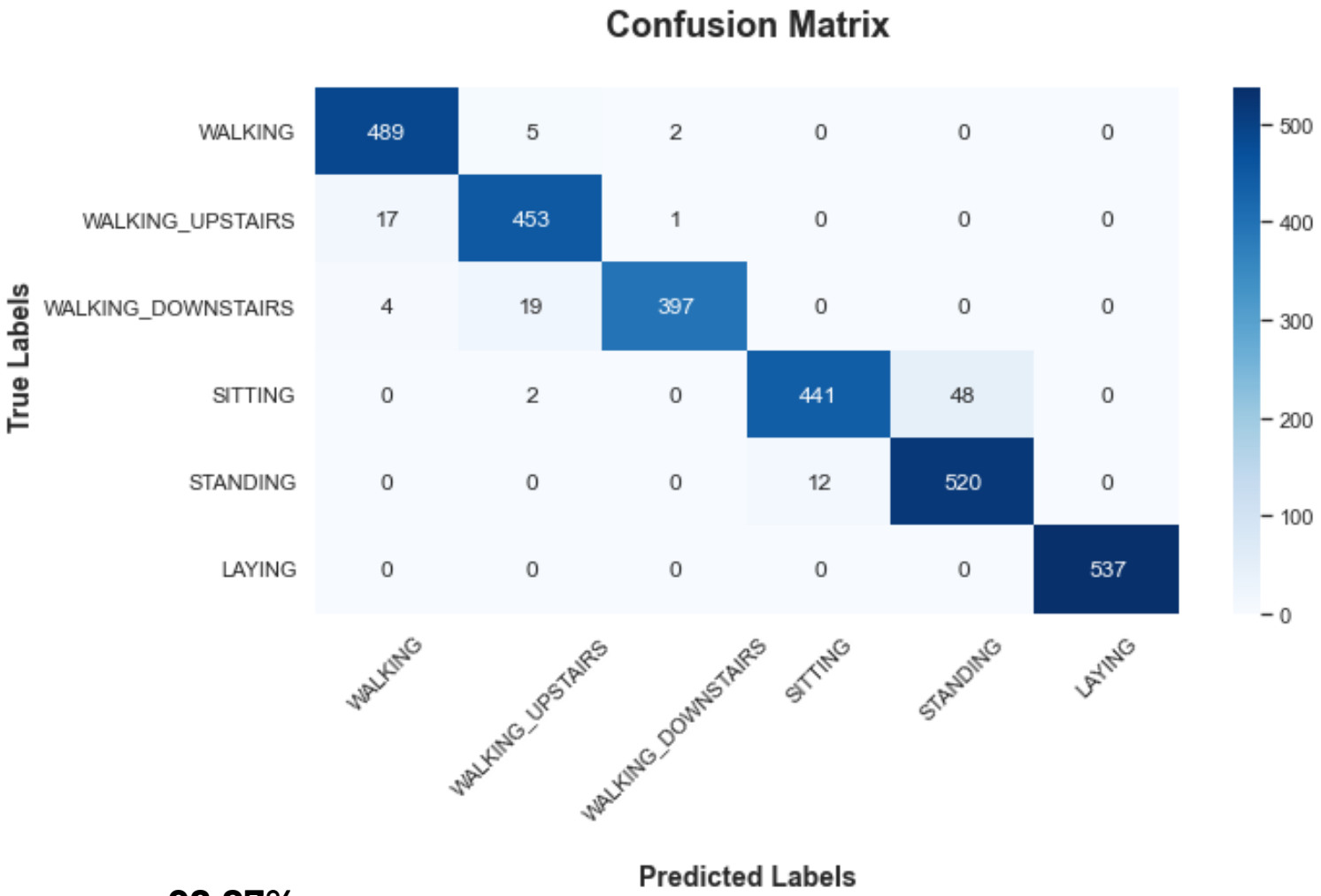
3. RBVF Kernel SVM

Hyperparameters:
C : [2,8,16]
Gamma:[0.0078125, 0.125, 2]

Training + Validation Time:
04:20.63

Testing Time:
00:01.15

GridSearch Report:
Best Estimator: SVC(C=16, gamma=0.0078125)
Best parameters: {'C': 16, 'gamma': 0.0078125}
Total number of CrossValidation sets: 5
Best Score: 0.9447834551903698



Accuracy: 96.27%

4. Linear SVC

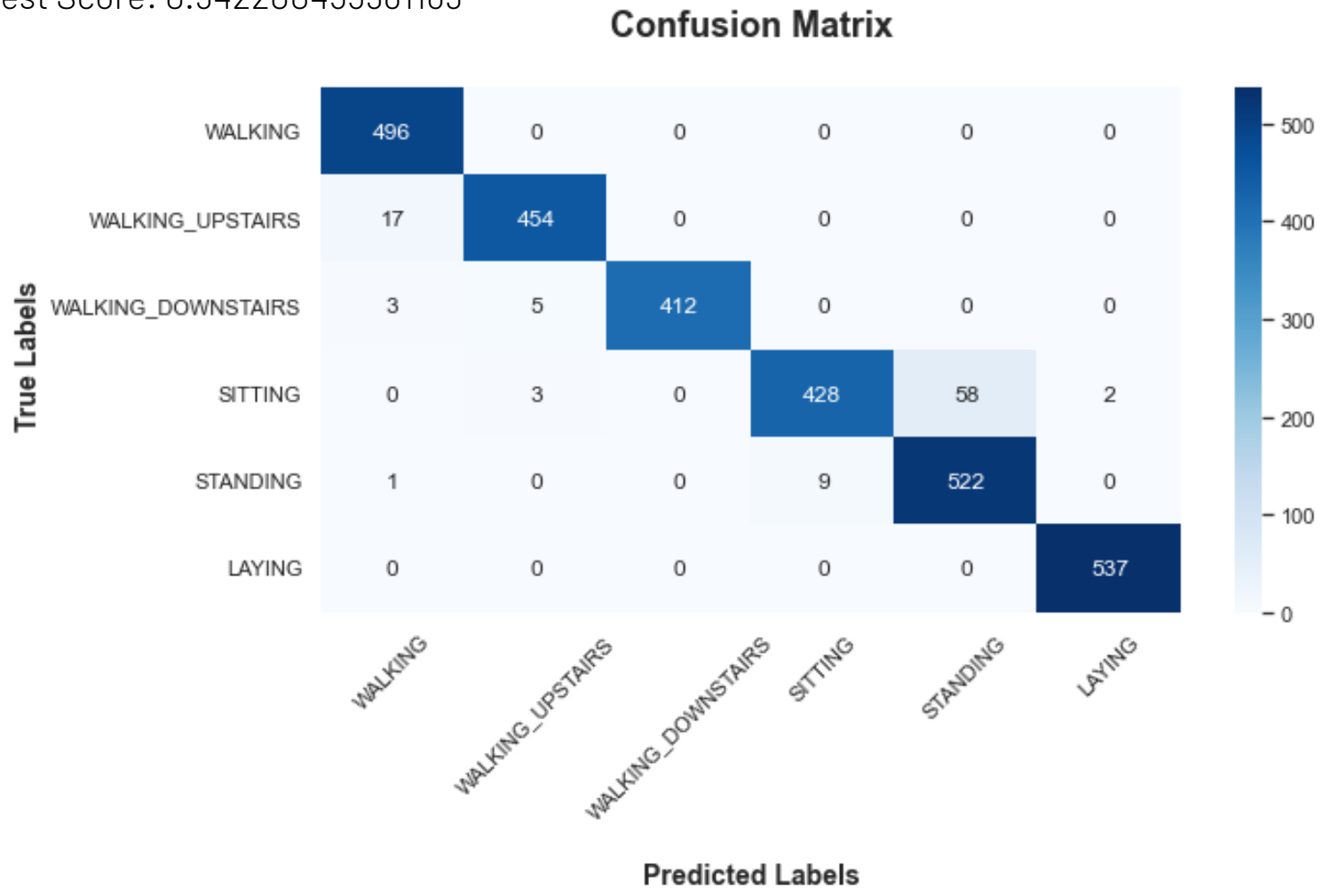
Tolerance:
0.00005

Hyperparameters:
C : [0.125, 0.5, 1, 2, 8, 16]

Training + Validation Time:
00:29.82

Testing Time:
00:00.01

GridSearch Report:
Best Estimator: LinearSVC(C=0.5, tol=5e-05)
Best parameters: {'C': 0.5}
Total number of CrossValidation sets: 5
Best Score: 0.942200455981169



Accuracy: 96.67%

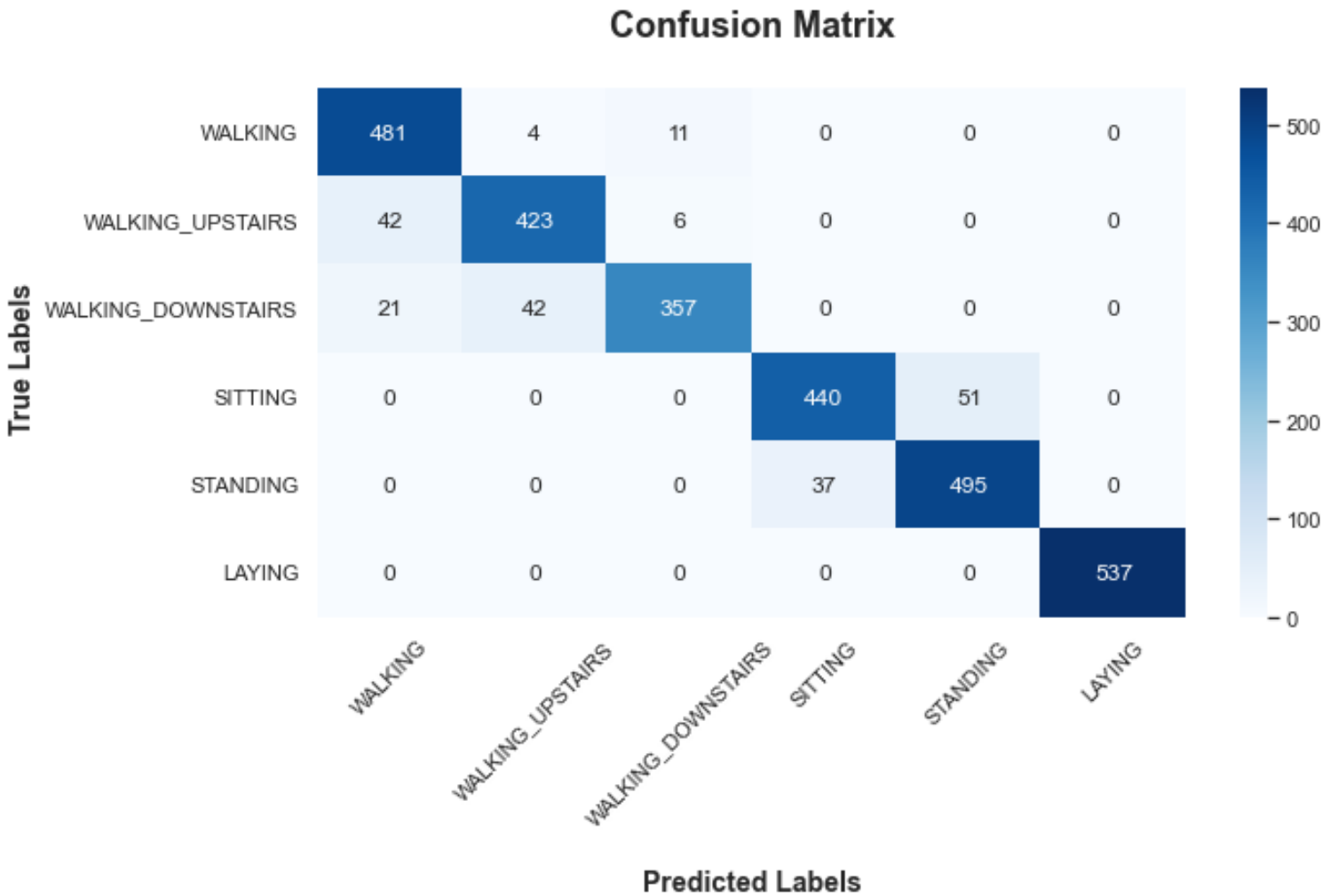
5. Random Forest Classifier

Hyperparameters:
N_estimators : np.arange(10,201,20)
Max_depth : np.arange(3,15,2)

Training + Validation Time:
04:11.09

Testing Time:
00:00.06

GridSearch Report:
Best Estimator: RandomForestClassifier(max_depth=11, n_estimators=110)
Best parameters: {'max_depth': 11, 'n_estimators': 110}
Total number of CrossValidation sets: 5
Best Score: 0.9226103765775514



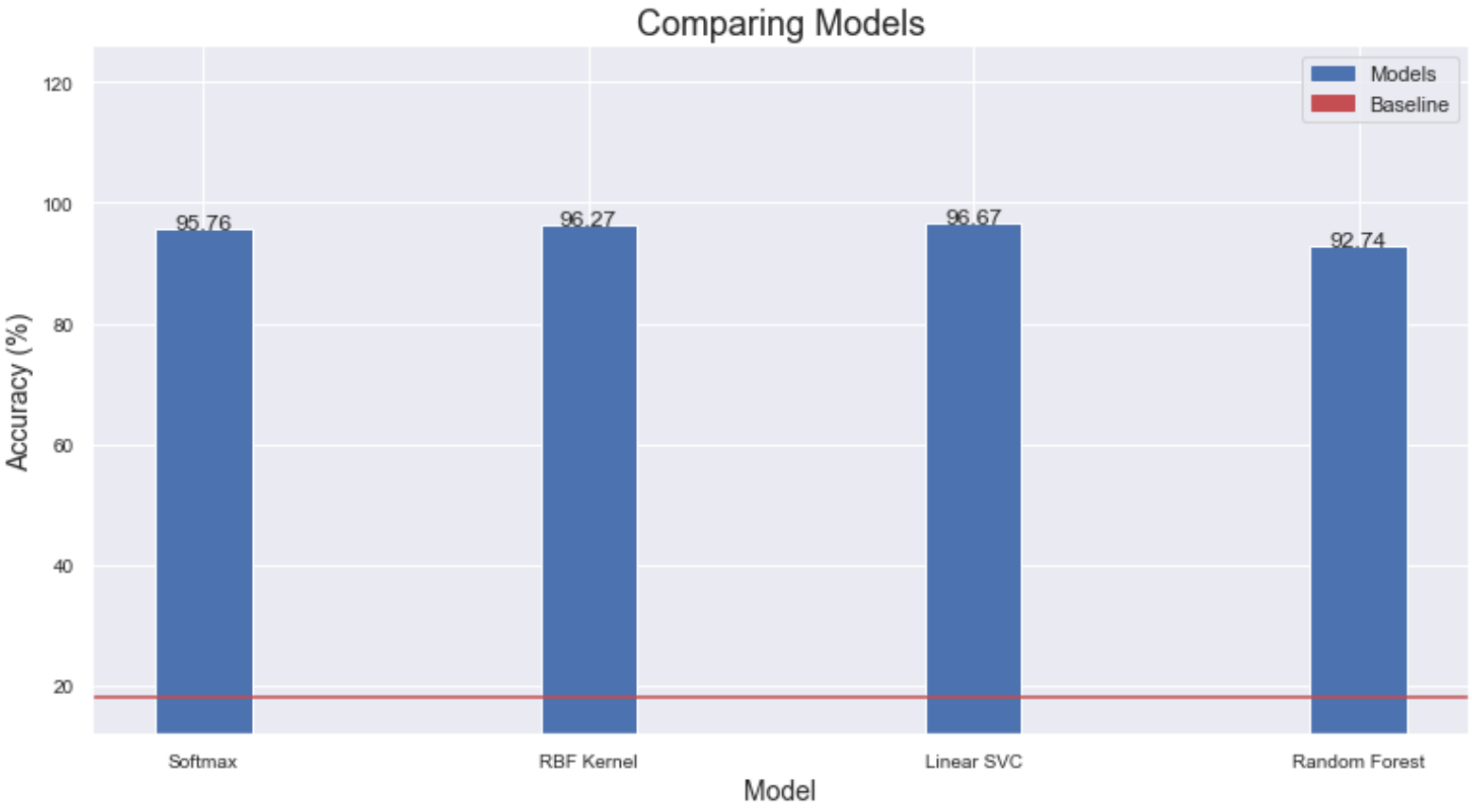
Accuracy: 92.74%

Results

Comparing all the models with their accuracies, we can see that Linear SVC has the best accuracy compared to every other model. Also, compared to baseline, every model has a substantially higher accuracy.

We can also observe that Linear SVC and RBF SVM had very similar results i.e. their accuracies are very close. So we can can also choose the RBF kernel instead of the linear kernel for our classification objective. But we can conclude that the support vector approach performs the best than all the other approaches as it is able to find good hyperplanes to separate the feature space and is also able to generalize well.

Model	Accuracy
Baseline Majority-Guess	18.22
Softmax Regression	95.76
RBF Kernel SVM	96.27
Linear SVC	96.67
Random Forest Classifier	92.74



References

1. https://www.kaggle.com/datasets/uciml/human-activity-recognition-with-smartphones?resource=download
2. https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones
3. https://www.youtube.com/watch?v=XOEN9W05_4A
4. https://www.w3schools.com/python/matplotlib_pie_charts.asp
5. https://stackoverflow.com/questions/59696944/density-plot-python-pandas
6. https://towardsdatascience.com/softmax-function-simplified-714068bf8156
7. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
8. https://www.ibm.com/docs/it/spss-modeler/SaaS?topic=models-how-svm-works
9. https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a
10. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
11. https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html
12. https://dzone.com/articles/using-jsonb-in-postgresql-how-to-effectively-store-1
13. https://towardsdatascience.com/svm-with-scikit-learn-what-you-should-know-780f1bc99e4a
14. https://www.datatechnotes.com/2020/07/classification-example-with-linearsvm-in-python.html
15. https://pythonprogramming.net/linear-svc-example-scikit-learn-svm-python/
16. https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
17. https://www.vebuso.com/2020/03/svm-hyperparameter-tuning-using-gridsearchcv/
18. https://www.datacamp.com/community/tutorials/random-forests-classifier-python
19. https://towardsdatascience.com/a-practical-guide-to-implementing-a-random-forest-classifier-in-python-979988d8a263
20. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
21. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74
22. https://medium.com/fintechexplained/what-is-grid-search-c01fe886ef0a
23. https://towardsdatascience.com/grid-search-for-hyperparameter-tuning-9f63945e8fec
24. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
25. https://www.stackvidhya.com/plot-confusion-matrix-in-python-and-why