PROJECT - ACTIVITY RECOGNITION

Team
Chidambaram Crushev
Preeti Maurya
JunYong Lee
Stephanie

Motivation

The data collected from activities recognized on smartphones can help in customizing the smartphone screen features accordingly. The dataset used in this project contains the records and features related to smartphone data which helps to predict or classify what activity an individual is currently doing based on the data. For instance, a user's activities involve whether he is standing, sitting, laying, walking, etc. Classification algorithms are one of the well known algorithms in supervised machine learning where the training dataset is labelled. Using classification algorithms on existing datasets, a model is trained to predict the classes new data can belong to. There are different classification algorithms such as, Perceptron, Adaline, Stochastic Gradient Descent, Decision Tree, SVM, Bayes classifier which can be applied to this dataset and recognize the activity of an individual.

Problem Description

The dataset size is more than 60 MB and has 562 columns. These features data are collected from the smartphone data and help us to recognize the activity of an individual whether he is Standing, Sitting, Laying, Walking, Walking_downstairs and Walking_upstairs. For data collection, accelerometer and gyroscope readings were taken when the individuals were performing one of the above activities. These large features(561 columns) are extracted based on the accelerometer and gyroscope readings and they underwent various mathematical operations(Fourier transform). With this huge data, we can address the following questions:

1. What is the activity of an individual based on the smartphone data?
   (We can perform binary and multi-class classification for the data with machine learning algorithms and then make predictions)

2. What is the accuracy of different machine learning models? How to improve those accuracies?

3. Since the data is huge, how will you improve the running time of the algorithm?
   (We need more computing power to process this data and we will be using the NMSU supercomputer (Discovery cluster) to improve the running time.)

Dataset

The dataset can be downloaded and found from the following link **https://www.kaggle.com/abhishektyagi001/activity-recognition-smartphone-merged-dataset** .The dataset has 10299 rows and 562 columns. The last column is the target label column which is named as "*Activity*". The Activity columns have different classes called "LAYING, SITTING, STANDING, WALKING, WALKING_DOWNSTAIRS, WALKING_UPSTAIRS". The feature columns(0 to 561) are extracted from the accelerometer and gyroscope readings when the individuals were subjected to doing one of these activities during the data collection process. The dataset is subjected to below preprocessing steps before they are fed into the classification model.

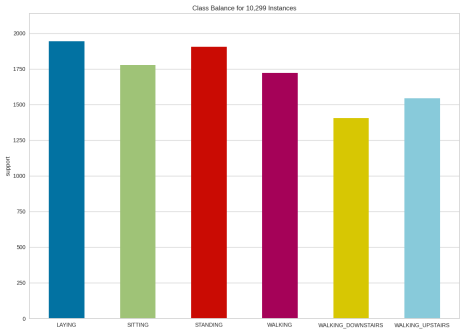
Data Preprocessing

The data preprocessing involved the below step

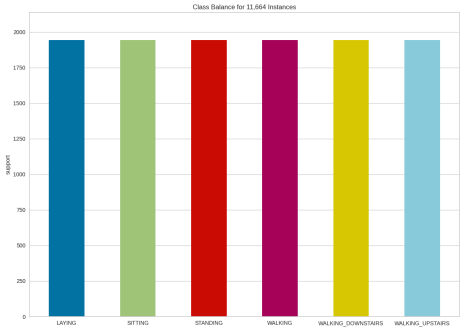
1. Replacing Null/NA values
2. SMOTE algorithm
3. Label encoding
4. Standardizing the dataset
5. Splitting the data into training and testing

The first step involved in the data preprocessing is to check for any null/NA values in the entire

dataset. The dataset doesn't have any null/NA value which is checked by the following command, **" df.isna().values.any(), df.isnull().values.any()"** The second step is to address the class imbalance problem in the dataset. The target classes had the following distribution of row samples.



From the figure, we can infer that the target classes have an unequal distribution of class samples. The classes "LAYING, SITTING, STANDING, WALKING, WALKING_DOWNSTAIRS, WALKING_UPSTAIRS" have 1944, 1176, 1905, 1722, 1406 and 1544 rows respectively. In order to address the problem, SMOTE(Synthetic Minority Oversampling Technique) is applied which solves the class imbalance problem. So, after applying the class imbalance problem, the distribution of class samples are,



From the above figure, we can infer that the classes are now balanced and can pass it to the next stage of preprocessing.

The third step is to do label encoding for the target classes. This converts the target labels into numeric form. So, "*LabelEncoder*" function is used to convert the target labels. So, after doing label encoding, the new target labels are,

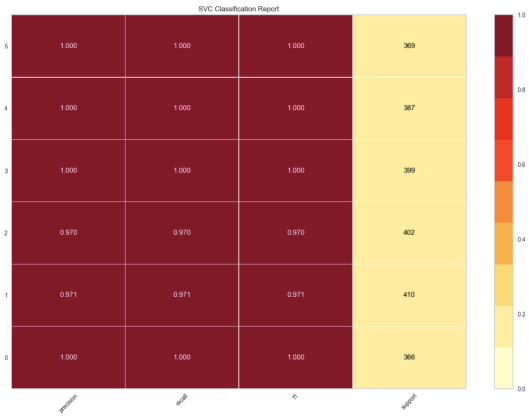
Before Label Encoding	**After Label Encoding**
LAYING	0
SITTING	1
STANDING	2
WALKING	3
WALKING_DOWNSTAIRS	4
WALKING_UPSTAIRS	5

The fourth step is to standardize the data. This is achieved using the *"*StandardScaler*"* function from the sklearn library. Once the data is standardized, the final step is to split the data into training and testing dataset. 80% of the data is reserved as the training data and 20% of the data is reserved as the testing data.

Implementation & Results

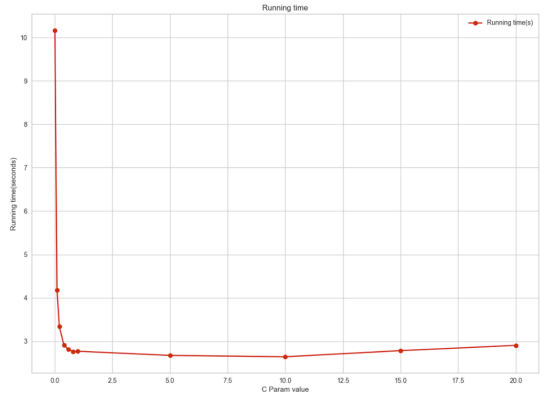
SVM - Linear

The training data is fed into the SVM - Linear Kernel model for various C parameter values with gamma value as 0.7. The values of C are as follows: [0.01, 0.1,0.2, 0.4, 0.6, 0.8, 1.0, 5, 10, 15, 20] . The SVM model is trained for different values of C and tested with the testing set side by side. Interestingly, it is found that the testing accuracy is higher with 0.99 when the corresponding C value is 1.0 The following plots denote the classification report, testing accuracy, running time of SVM model.

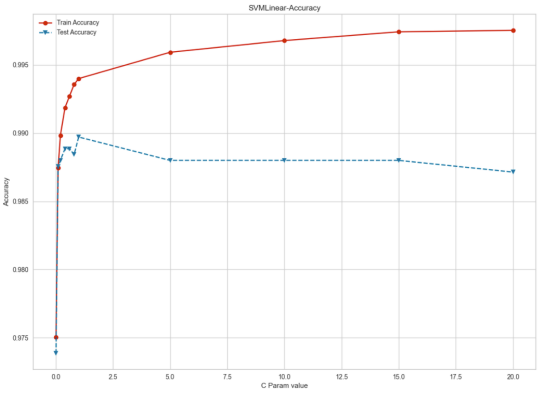


(Classification Report- SVM Linear)

The training data is fed into the SVM - Rbf Kernel model for various C parameter values with gamma value as 0.1. The values of C are as follows: [0.01, 0.1,0.2, 0.4, 0.6, 0.8, 1.0, 5, 10, 15, 20]. The Non Linear SVM model is trained for different values of C and evaluated with the testing set side by side. It is determined that the testing accuracy is higher with 0.99 when the corresponding C value is 5 for the Rbf kernel. The following plots denote the classification report, testing accuracy, running time of the non linear SVM model.



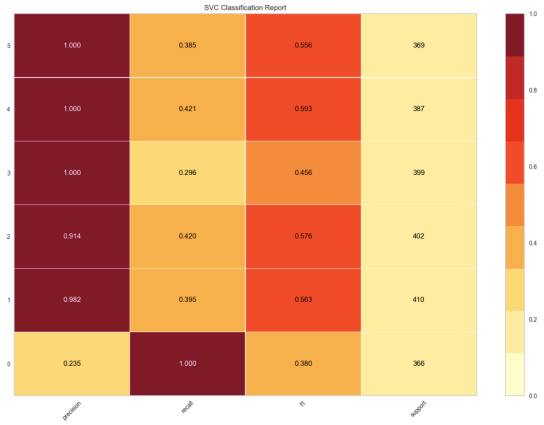
(Running Time)



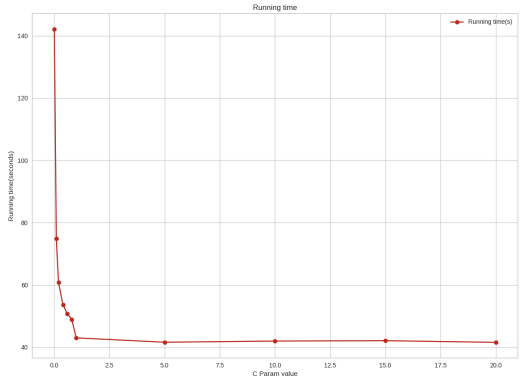
(Test and Training Accuracy)

SVM (Non-Linear)

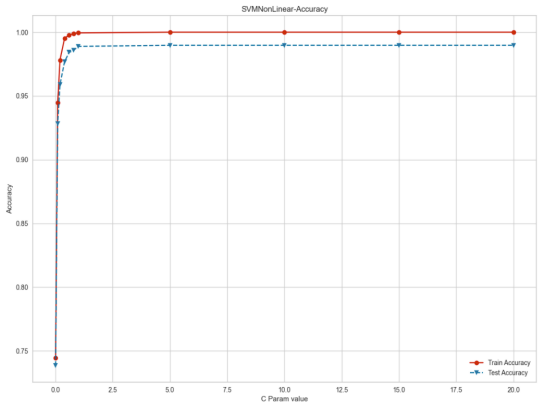
Kernel - Rbf Output



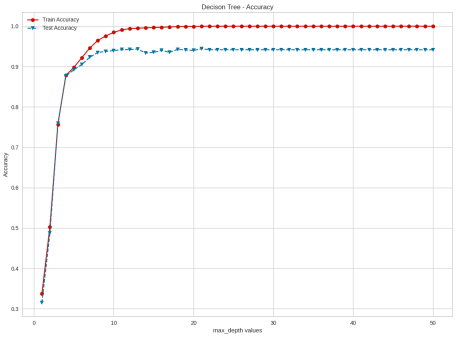
(NonLinear SVM-Rbf kernel-Classification Report)

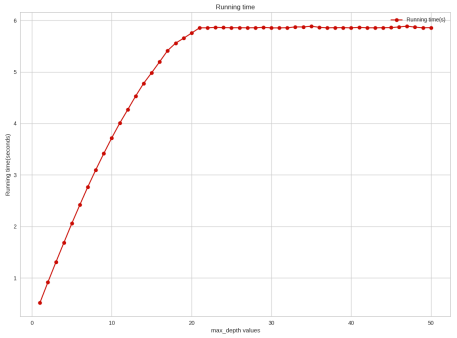


(Running Time for different values of C)



(Test Accuracies)





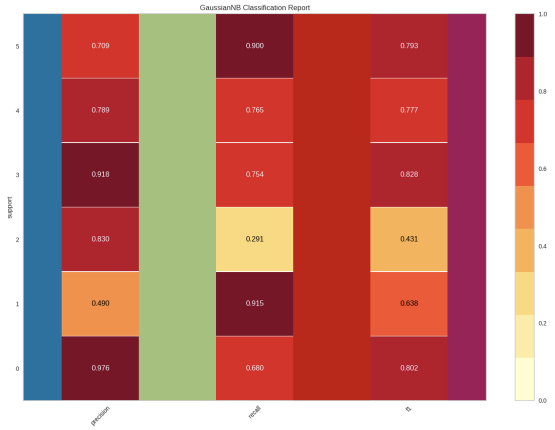
Decision Tree :

With DecisionTreeClassifier, we tested two criteria('gini' and 'entropy') with different max_depths.
We looked at the score on each target class, and the change of accuracy and running time as max_depth changes. The max_depths were 1 to 50.

As you can see below, with gini criterion, all scores were higher than 0.9, and class '1' showed the highest support value. For accuracy, we were able to observe that max_depth 21 showed the highest accuracy. The running time gradually increased as max_depth increased.

Criterion = 'gini'

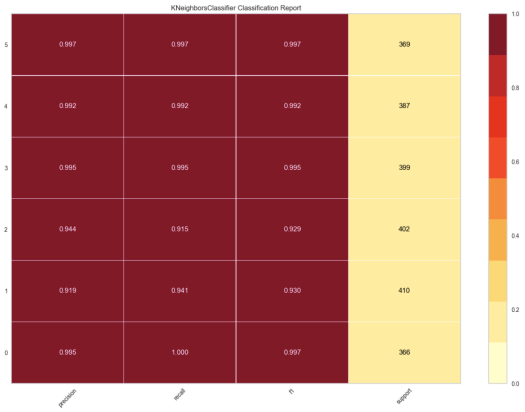
Naive Bayes classifier :
Below are the metrics:
Gaussian Naive Bayes model accuracy(in %): 71.58
Time taken to train model and prediction : 0.139 seconds



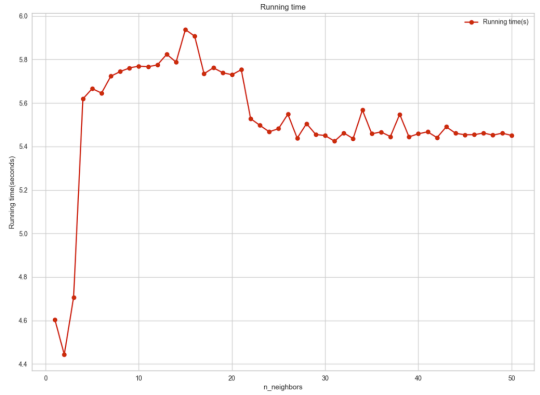
(Naive Bayes-Classification Report)

KNN :
With the KNN classifier, we tested n_neighbors of list(range(1, 51)).
We looked at the score on each n_neighbor, and the change of accuracy and running time as n_neighbor changes.

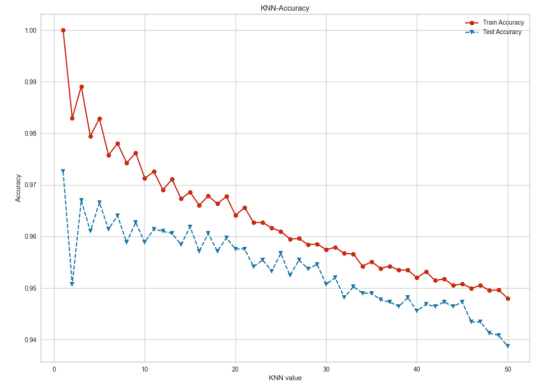
As you can see below, with n_neighbors = 1, we received the higher test accuracy = 0.973 due to unstable predictions. However, as we increase the value of n_neighbors the predictions become more stable upto a certain point.



(KNN-Classification Report)



(Running time for different values of n_neighbors)

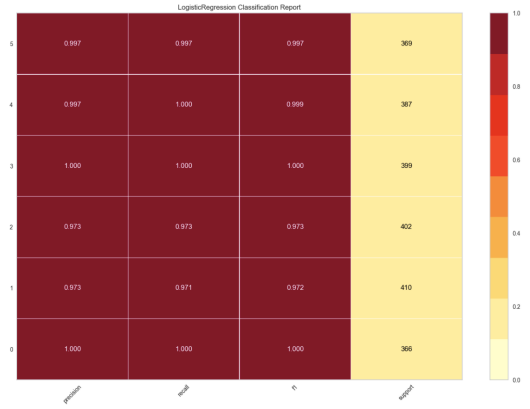


(Test Accuracies)

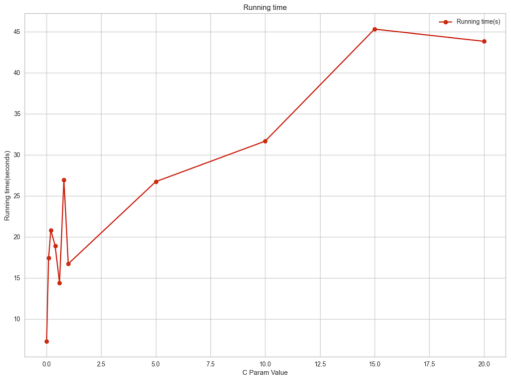
Logistic Regression :

With the Logistic Regression classifier, we tested the classifier with different values of Inverse of regularization strength C = [0.01, 0.1,0.2, 0.4, 0.6, 0.8, 1.0, 5, 10, 15,20].
We looked at the accuracy score on each value of C, and the change of accuracy and running time as C changes.

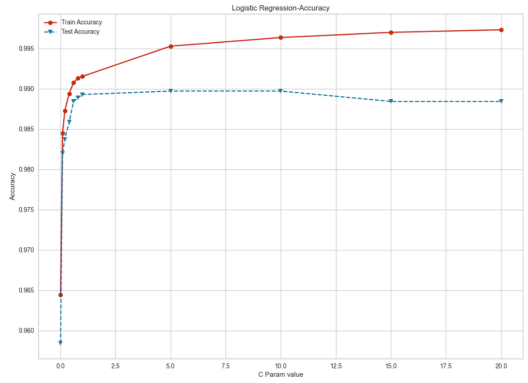
As you can see below, with C value 5.0, we received the highest test accuracy of 0.990 ie. However, running logistic regression with C = 15 consumed the highest running time compared to other C values.



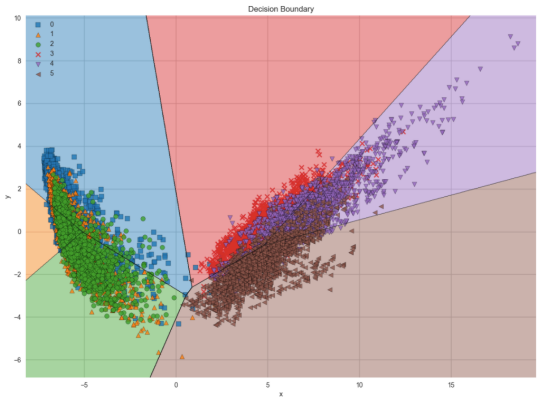
(Logistic Regression-Classification Report)



(Running time for different values of C Param)

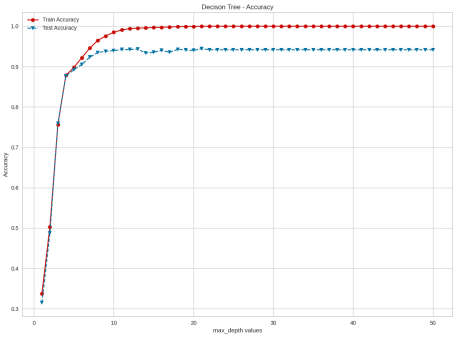


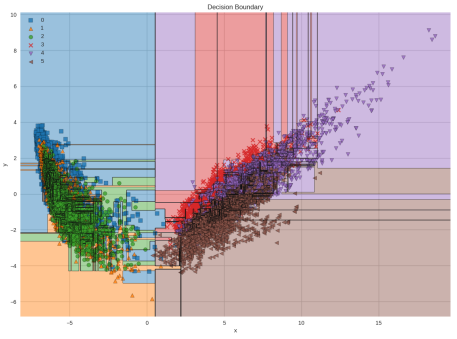
(Test Accuracies)

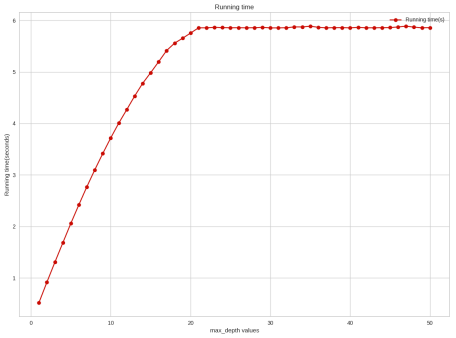


Decision tree

The decision tree classifier is implemented for the activity recognition dataset. It ran for 50 iterations with max Depth value ranging from 1 to 50. Gini index was used as the splitting criterion. After running 50 iterations, the maximum test accuracy was found to be 0.944 for maxdepth value 21. Also, the average running time for all the iterations was found to be 4.958 seconds. The running time gradually increased for increasing maxdepth values.

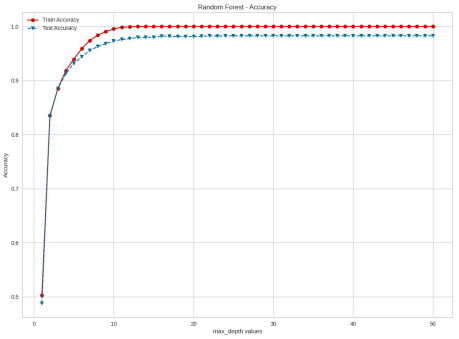


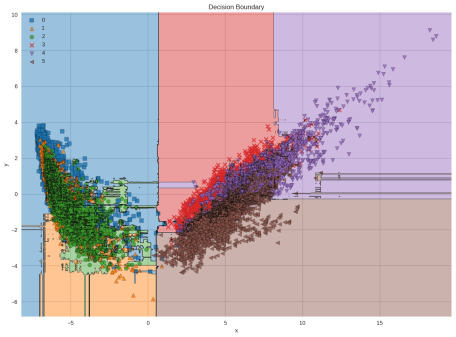


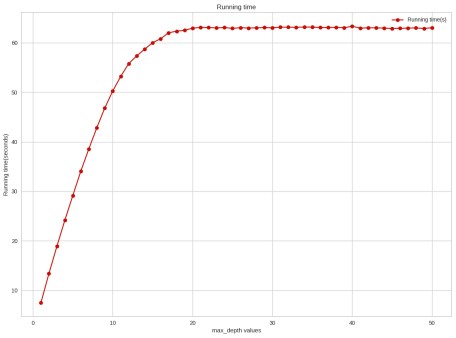


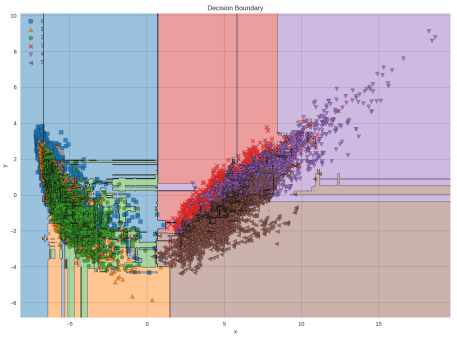
Random Forest

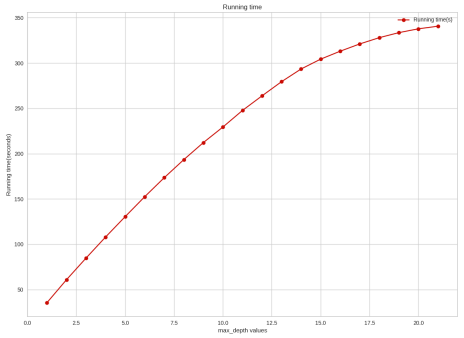
The random forest classifier is tested with the following parameters - Maxdepths ranging from 1 to 50, number of estimators to be 500 and criterion to be ‘gini’. From the below charts, we can infer that the maximum text accuracy using random forest was found to be 0.982 for maxdepth value 22. But if you compare the average running times of decision trees and random forest, random forest took a lot of time which is 56.245 seconds because the number of estimators is 500 and constructs a lot of decision trees. The accuracy increase using random forest comes at the expense of running time.







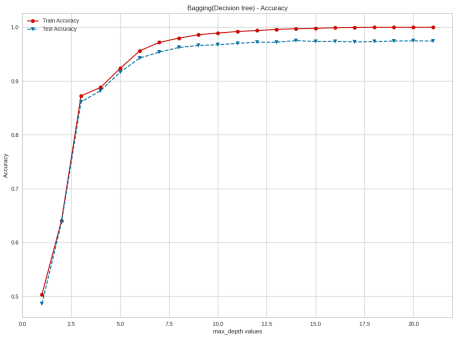




Bagging

The bagging is tested with the following parameters - max_depth ranging from 1 to 23, random_state = 1, number of estimators = 1000. From the below chart, we can infer that the test accuracy was improved from 0.944 to 0.954 for the decision tree classifier.

However, the bagging took a lot of time to complete the execution. The average running time for all the iteration is 225 seconds. So, the accuracy increase comes at the expense of running time.



AdaBoost

Adaboost is tested with following parameters: number of estimators = 100, base estimator = DecisionTree, max_depth ranging from 1 to 23. As a result, the maximum accuracy is 0.954 with running time 24.273. The DecisionTree accuracy has been improved from 0.944 to 954 using adaboost classifier.

Ensemble Vote Classifier

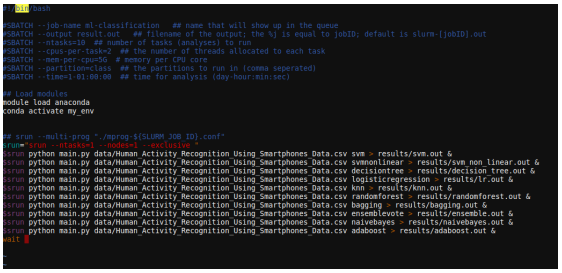
The ensemble vote classifier was used to make predictions because it combines the prediction of various classification models and makes the prediction by using Majority voting technique.

For the ensemble classifier, decision tree, svm linear, logistic regression, k nearest neighbors were used. For the decision tree, maxdepth is set to 1. For svm linear and logistic regression, the C param value is set to 1.0 and 5.0 respectively. For knn, the number of neighbors is set to 1.

By using ensemble, the accuracy achieved was 0.98. This is higher than the decision tree accuracy which is 0.944. Interestingly, it took a lot of time to run and the running time was estimated to be 2885 seconds.

How did we execute the models?

We(Chidambaram, Jun Yong, Preeti) have created our accounts in NMSU supercomputer Discovery cluster(https://hpc.nmsu.edu/home/) after completing the Discovery onboarding course in Canvas. We are allocated a project space (**ml-classification**) of 500 GB in the cluster by the supercomputer team to run and test our models using SLURM which is a scheduling system in the cluster. By running our algorithms, we can reduce the running time of our model and get the results faster. To achieve that, we are planning to use the Anaconda environments and then create our own custom environment and install the packages that we need in order to run and test our models. By having anaconda environments, we can mitigate the risk of dependency issues. A sample slurm script to request resources in the cluster and run our jobs(testing our code) will be as follows:



The above script shows how we can request resources in the cluster and run our jobs. So the SBATCH scripts are the ones which request resources in the cluster. The n tasks are set to 10 which means that we have 10 tasks corresponding to each model. Also, the number of CPUs allocated are 2 for each task and memory per cpu is 5GB. Hence, 10 Gb of memory is allocated for the task which is sufficient for the dataset. Also, in the cluster we have different partitions which basically are work queues. They actually have a set of computational nodes which does the actual job. So, we used the partition “class” which is a new partition with better processors. Then, the maximum time limit for the job is 1 day and 1 hour for the job.

Once the resource request code is written, you need to have the necessary python and machine learning libraries to run your python scripts. So, we used anaconda module and created our own custom environment. In the custom environment “my_env”, all the necessary python and machine learning packages are installed. Once the custom environment is ready, we activate the particular conda environment.

Then, we have the “srun” command which is responsible for running the python script and writing the output of each model to a separate output file. Once all the models run successfully, the job gets finished and the results are written to the output file.

Running time/ Accuracy table

Model Name	**Avg Running Time(sec)**	**Max Accuracy**
SVM - Linear	5.029	0.990
SVM - Non Linear	58.285	0.990
KNN	5.521	0.973
Logistic Regression	36.292	0.990
Decision Tree	4.958	0.944
Random Forest	56.245	0.982
Bagging	225.824	0.975
Boosting	24.273	0.954
Ensemble Vote	2885	0.980

Conclusion

Implemented several classification models on the Activity Recognition SmartPhones Data to make predictions in order to determine the activity of an individual. SVM and Logistic Regression performed really well with high accuracies (0.99). Though Decision tree accuracy was around 0.94, it was improved by using the ensemble methods like adaboost, bagging and ensemble vote classifier. The major challenge in this project is running the models which consumes a lot of time and space in a single computer. It is addressed by running on the supercomputer. Also, parallelism is achieved by running all the models at the same time.

Thank you!!!