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.)

Midterm report

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

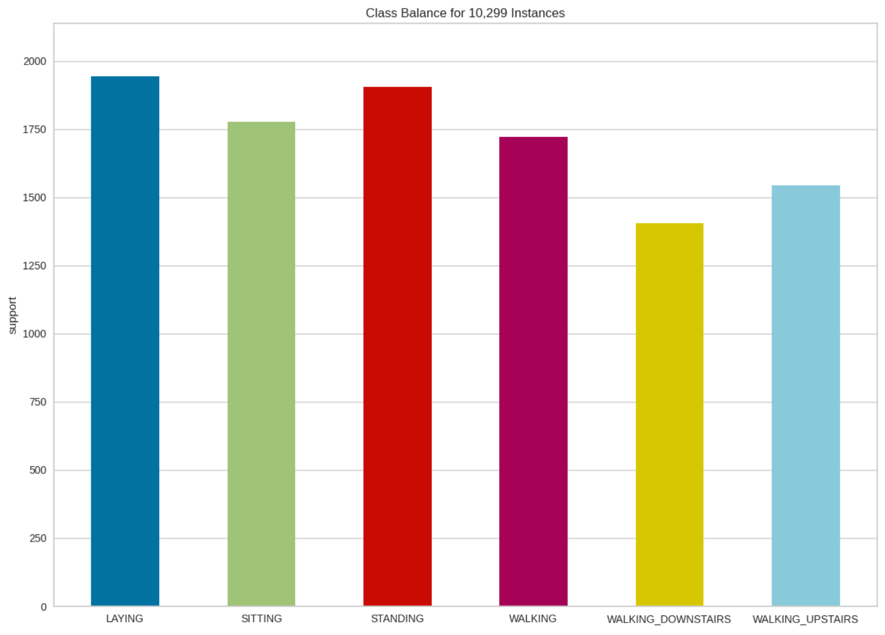
The data preprocessing involved the below step

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

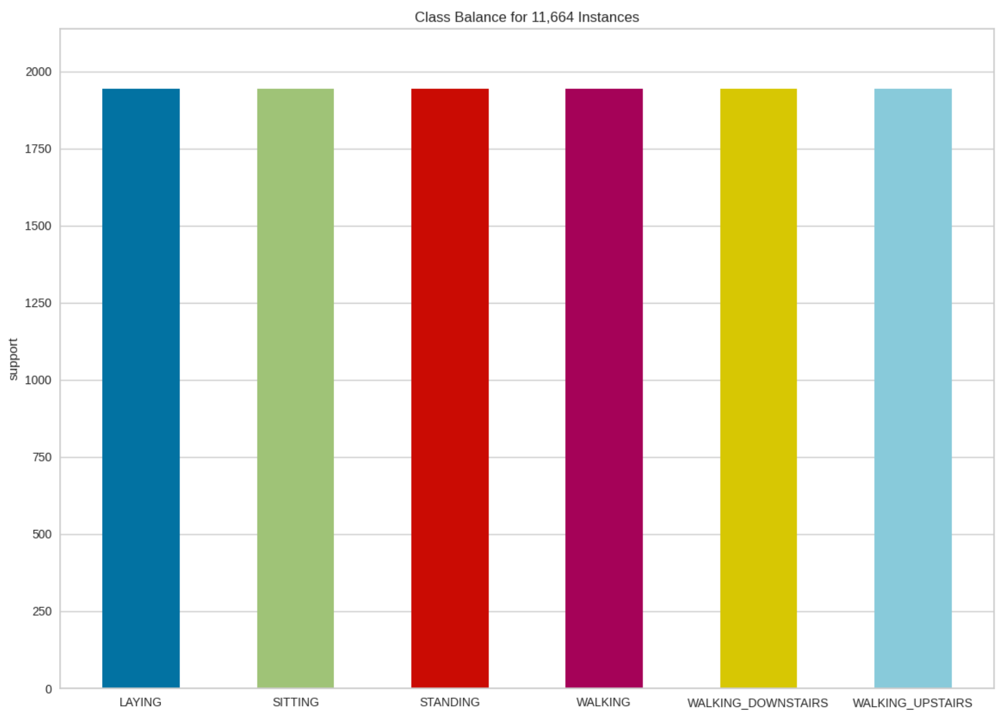
The first step involved in the data preprocessing is to check for any null/NA values in the entire dataset. The dataset doesn't have any null/NA value which is checked by the following command,

" 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,

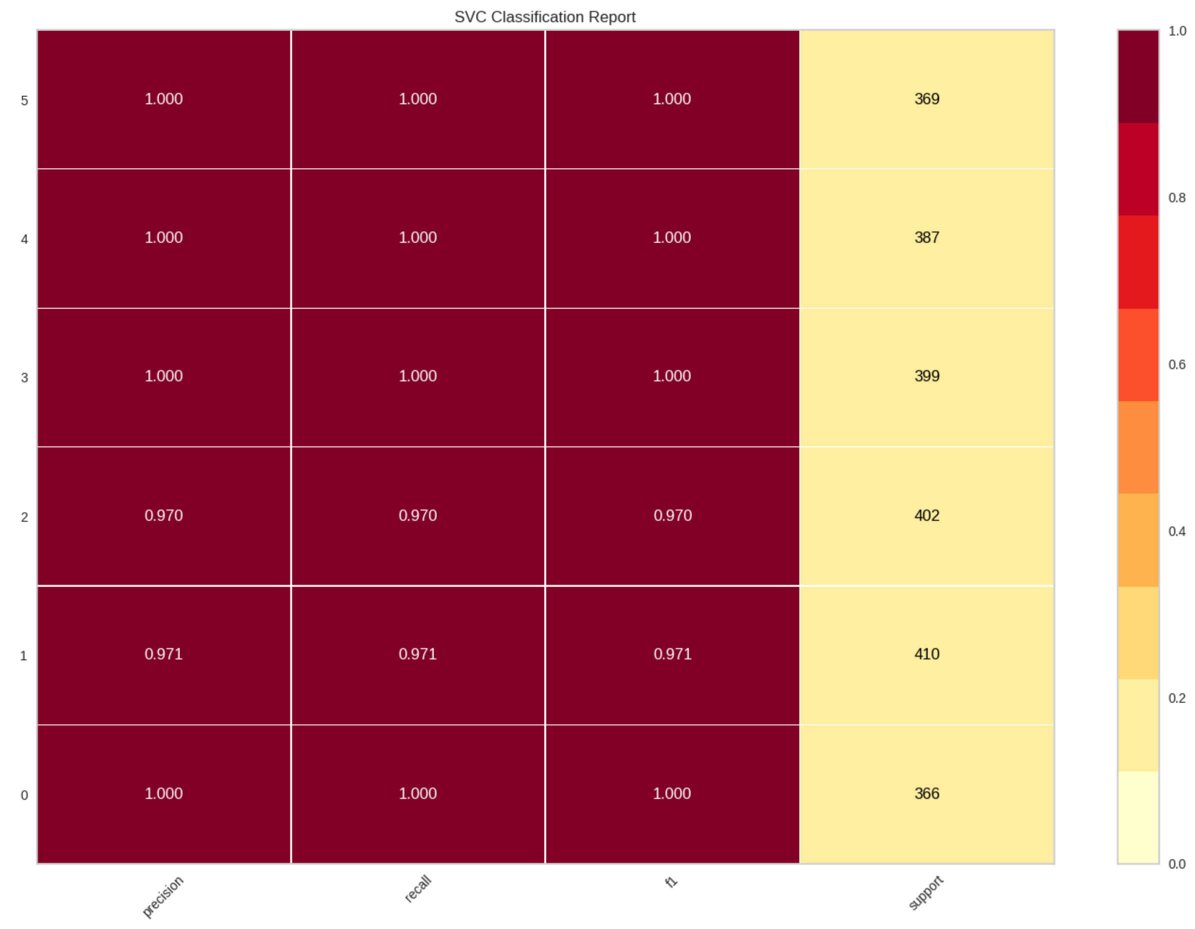
Before Label Encoding	**After Label Encoding**
LAYING	0
SITTING	1
STANDING	2
WALKING	3
WALKING_DOWNSTAIRS	4
WALKING_UPSTAIRS	5

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

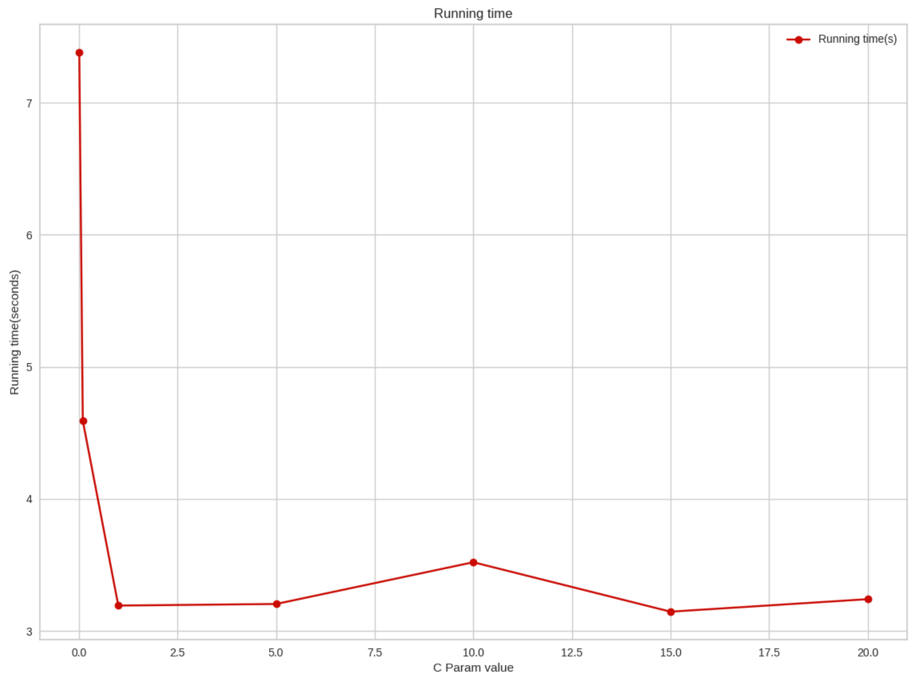
SVM - Linear

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, 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 when 0.99 when the corresponding C value is 1.0. The following plots denote the classification report, testing accuracy, running time of SVM model.

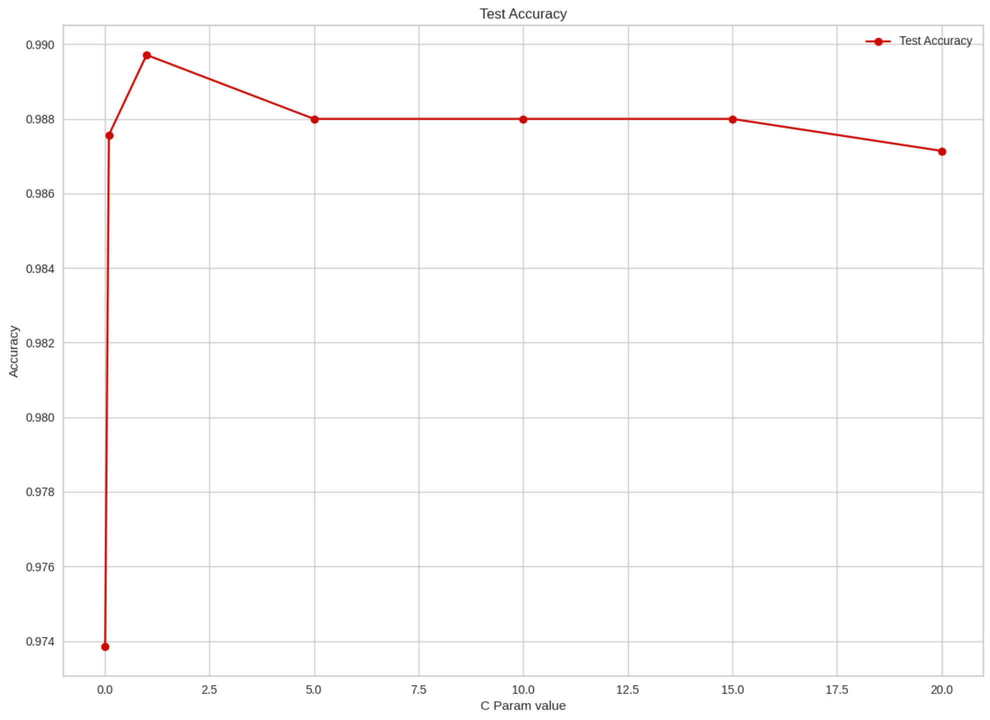


	precision	recall	f1	support
5	1.000	1.000	1.000	369
4	1.000	1.000	1.000	387
3	1.000	1.000	1.000	399
2	0.970	0.970	0.970	402
1	0.971	0.971	0.971	410
0	1.000	1.000	1.000	366

(Classification Report- SVM Linear)



(Running Time)

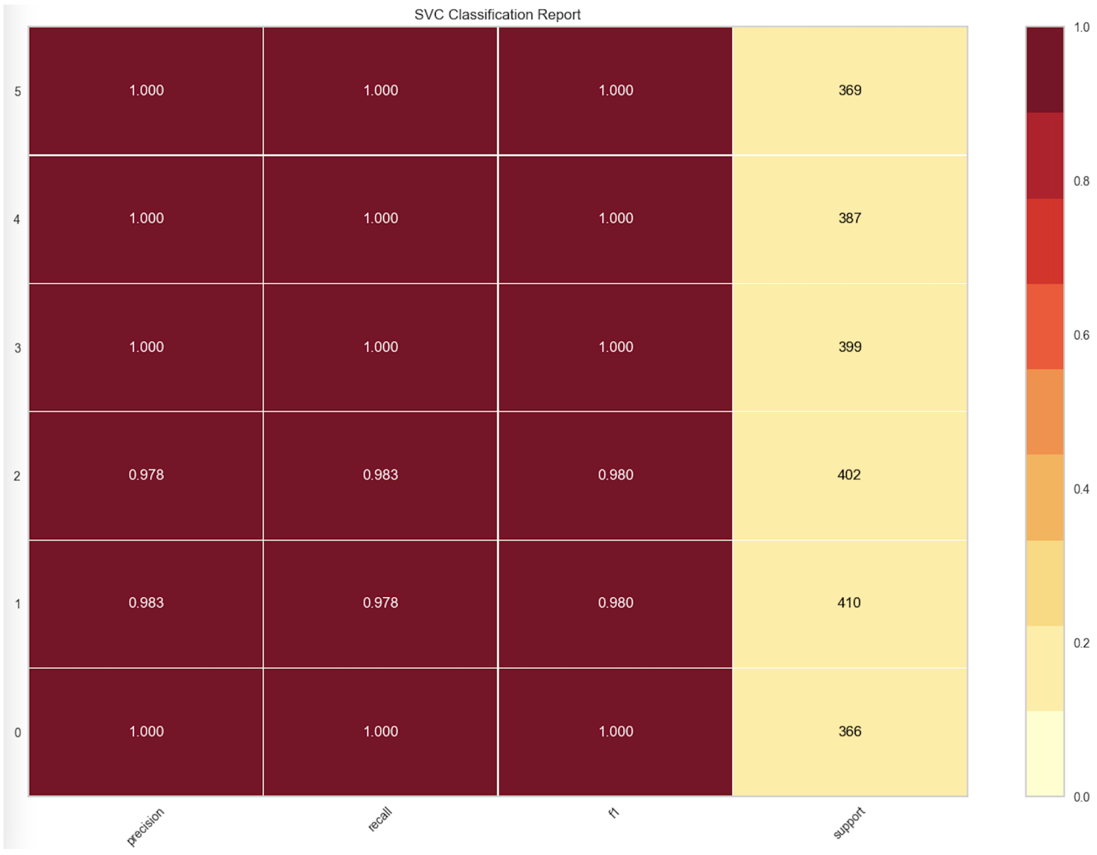


(Test Accuracy)

SVM (Non-Linear)

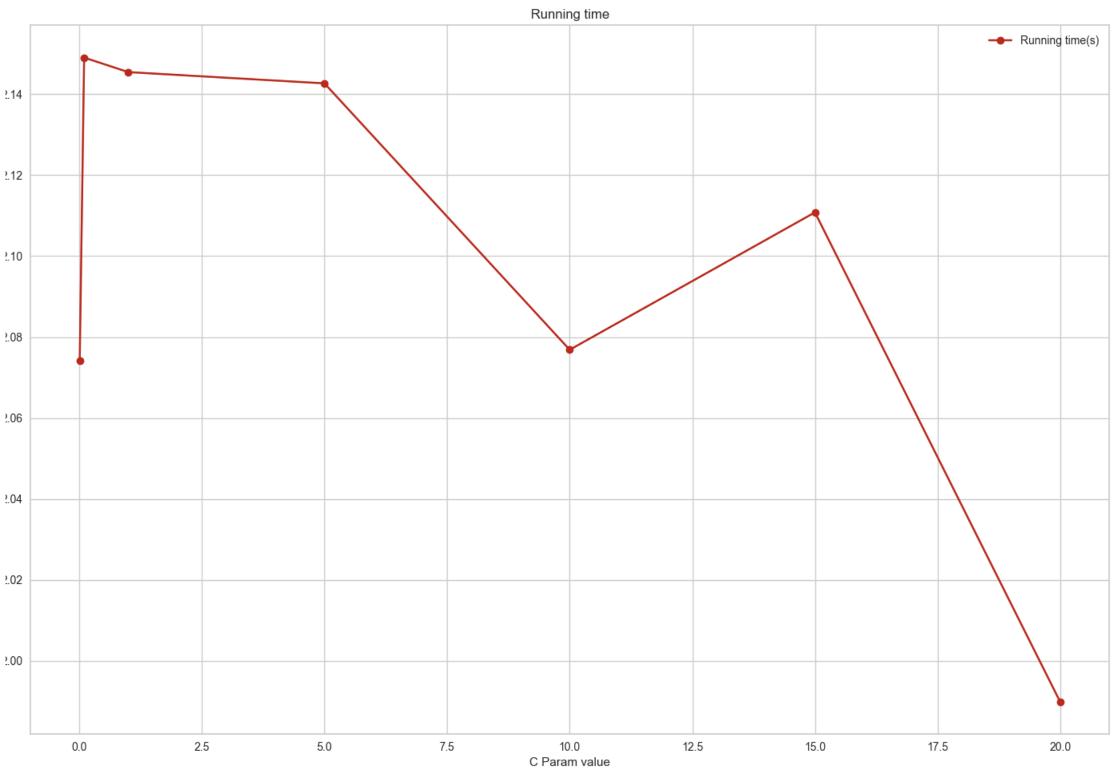
The training data is fed into the SVM - Rbf and Poly Kernel model for various C parameter values with gamma value as 0.7. The values of C are as follows: [0.01, 0.1, 1.0. 5, 10, 15, 20] . The Non Linear SVM model is trained for different values of C and evaluated with the testing set side by side. It is determined that the testing accuracy is higher with 0.480 when the corresponding C value is 5 for Rbf kernel, whereas the testing accuracy is higher with 0.993 when the corresponding C value is 0.01 for Poly kernel. The following plots denote the classification report, testing accuracy, running time of the non linear SVM models.

Kernel - Poly Ourput

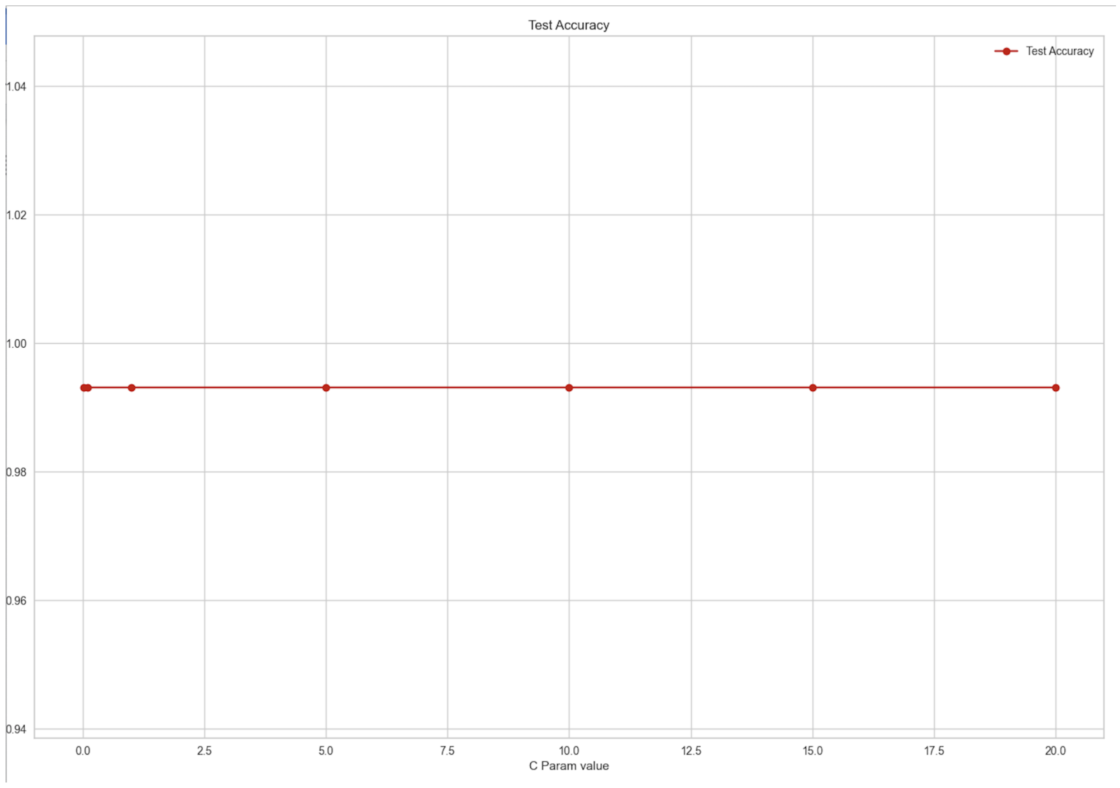


	precision	recall	f1	support
5	1.000	1.000	1.000	369
4	1.000	1.000	1.000	387
3	1.000	1.000	1.000	399
2	0.978	0.983	0.980	402
1	0.983	0.978	0.980	410
0	1.000	1.000	1.000	366

(NonLinear SVM-Kernel Poly-Classification report)

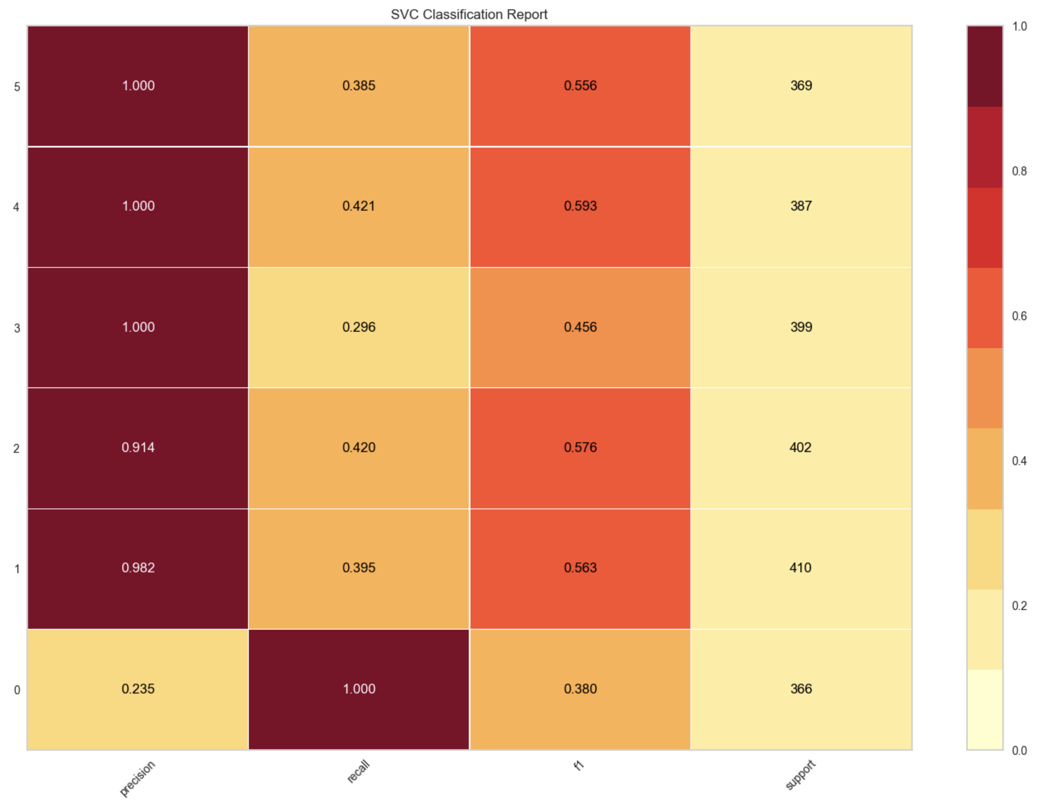


(Running time for different values of C)



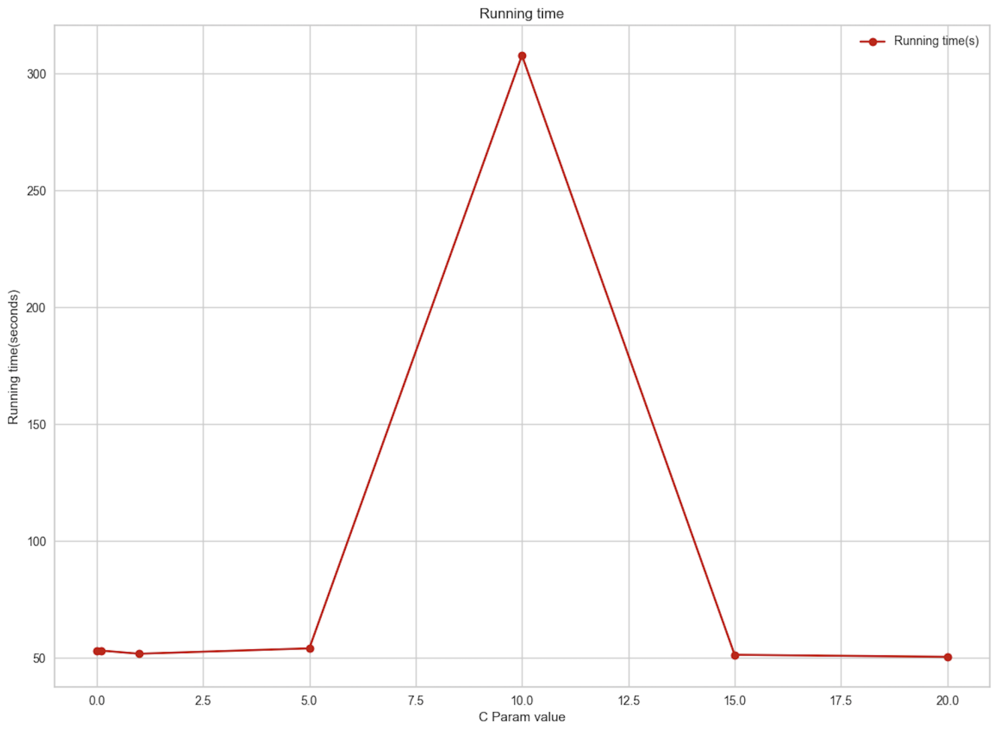
(Test Accuracies)

Kernel - Rbf Output

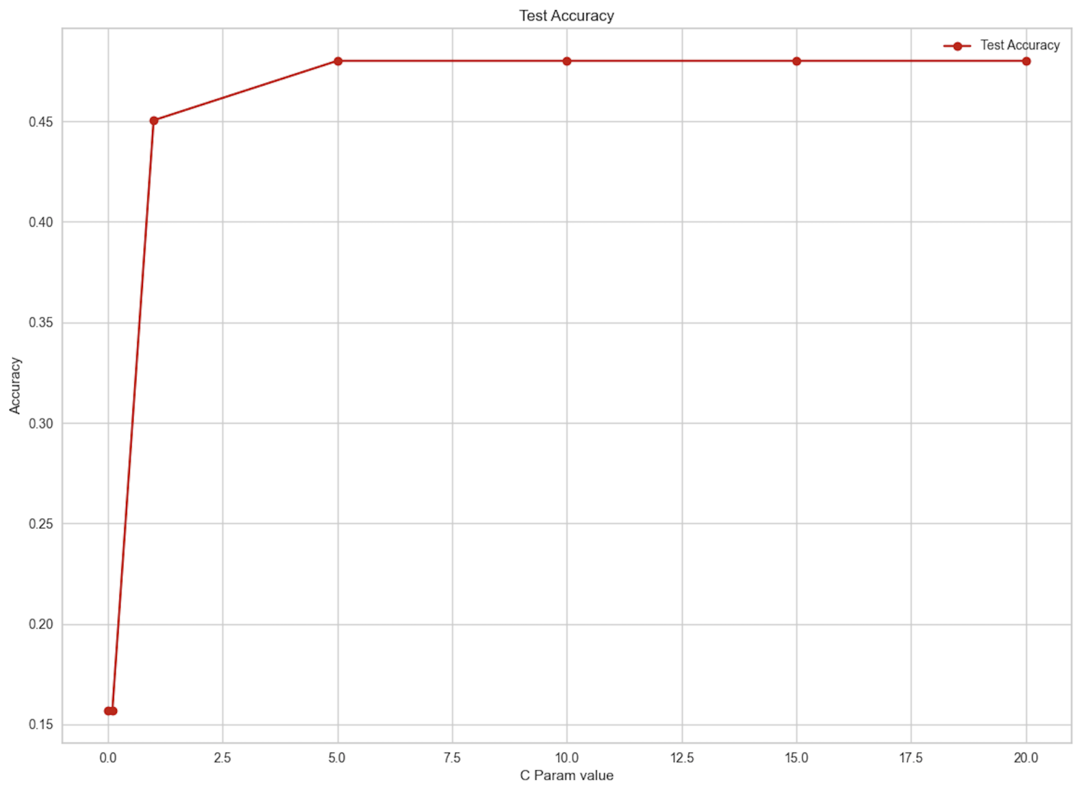


	precision	recall	f1	support
5	1.000	0.385	0.556	369
4	1.000	0.421	0.593	387
3	1.000	0.296	0.456	399
2	0.914	0.420	0.576	402
1	0.982	0.395	0.563	410
0	0.235	1.000	0.380	366

(NonLinear SVM-Rbf kernel-Classification Report)



(Running Time for different values of C)



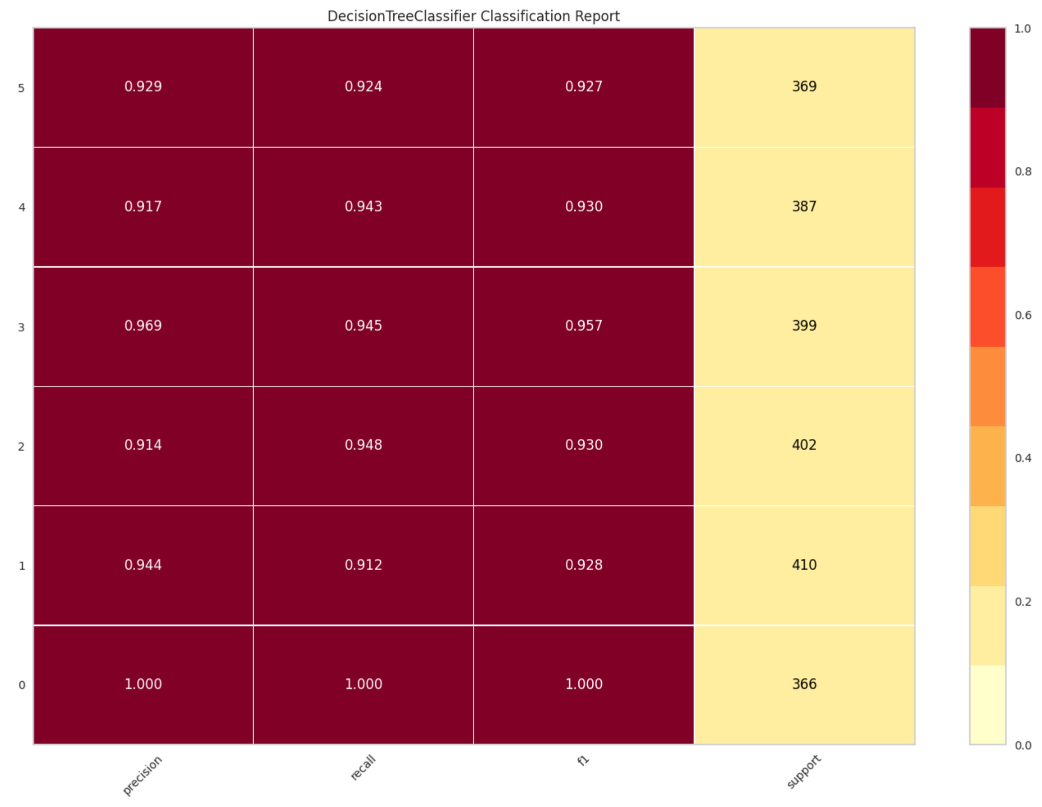
(Test Accuracies)

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 2, 5, 10, and 20.

As you can below, with gini criterion, all scores were higher than 0.9, and class ‘1’ showed the highest support value. For the accuracy, we were able to observe that max_depth 10 showed the highest accuracy. The running time gradually increased as max_depth increased.

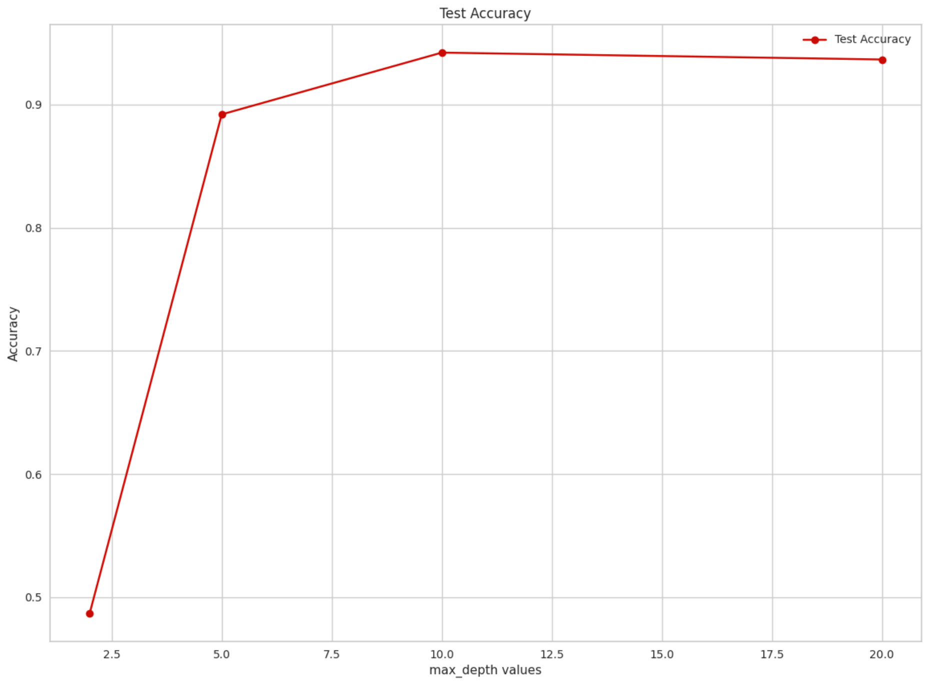
Criterion = ‘gini’



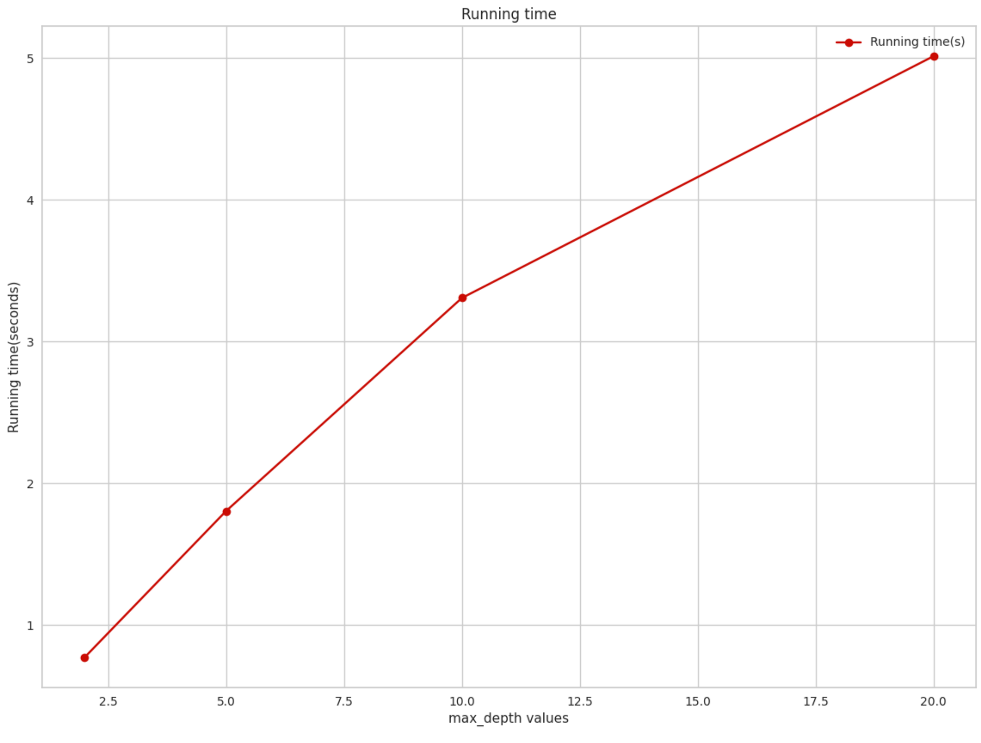
DecisionTreeClassifier Classification Report

	precision	recall	f1	support
5	0.929	0.924	0.927	369
4	0.917	0.943	0.930	387
3	0.969	0.945	0.957	399
2	0.914	0.948	0.930	402
1	0.944	0.912	0.928	410
0	1.000	1.000	1.000	366

1.0
0.8
0.6
0.4
0.2
0.0



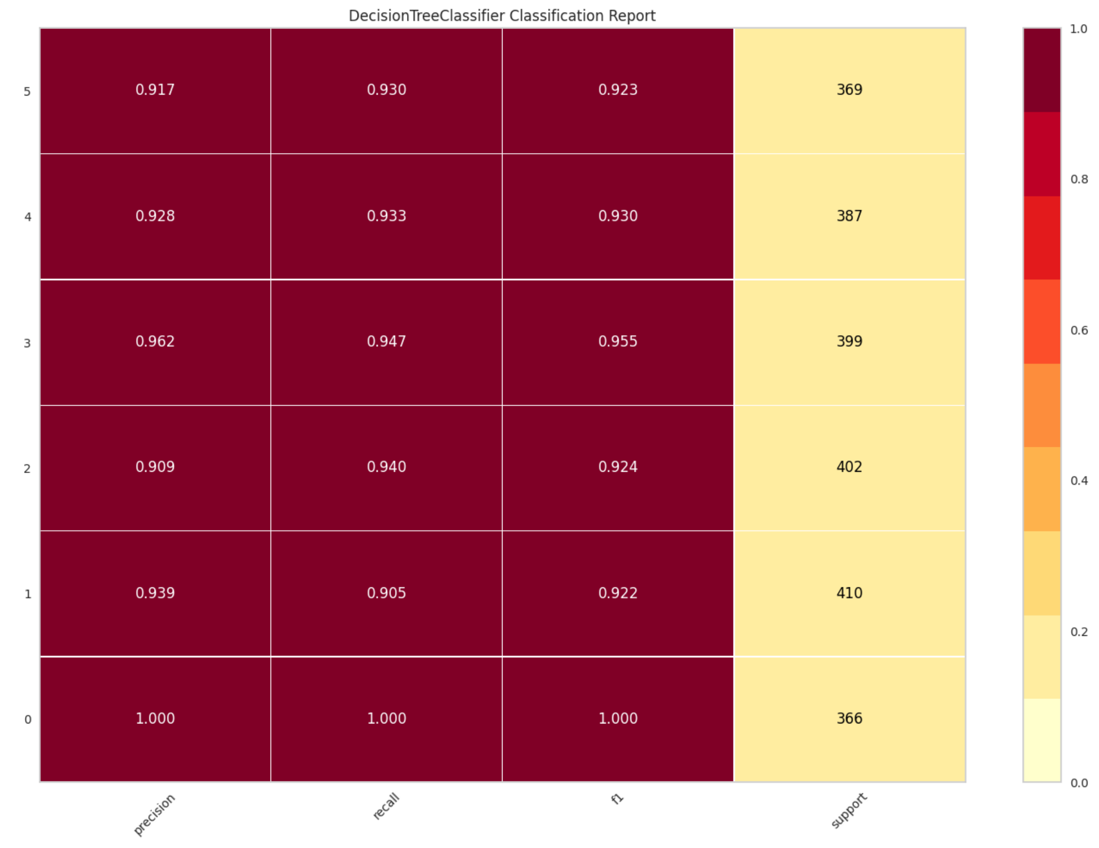
Test Accuracy
Test Accuracy
Accuracy
0.5
0.6
0.7
0.8
0.9
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
max_depth values



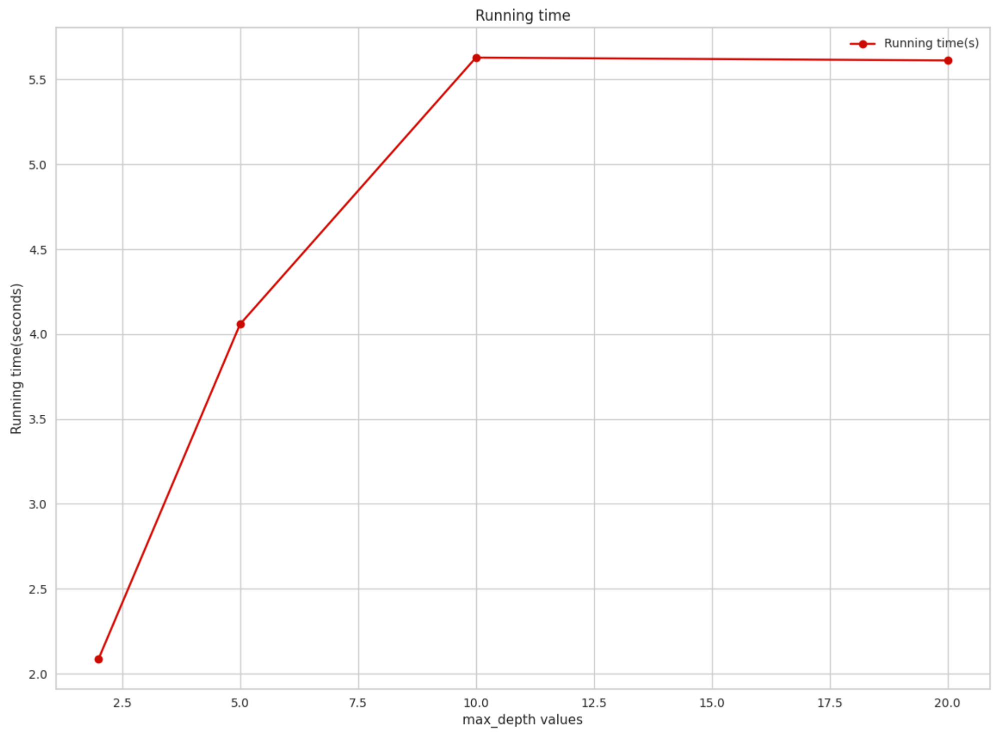
Running time
Running time(s)
Running time(seconds)
1
2
3
4
5
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
max_depth values

Criterion = ‘entropy’

With entropy criterion, it generally showed the scores higher than 0.9, and the class ‘1’ showed the highest support value. For the accuracy, max_depth = 10 showed the highest accuracy and 20 showed the lower accuracy. For the running time, it increased as max_depth increased, however, there was no big difference in running time with 10 and 20 max_depth. In addition, the running time reached 5 second with max_depth 10, which is faster than ‘gini’ criterion did.


DecisionTreeClassifier Classification Report

	precision	recall	f1	support
5	0.917	0.930	0.923	369
4	0.928	0.933	0.930	387
3	0.962	0.947	0.955	399
2	0.909	0.940	0.924	402
1	0.939	0.905	0.922	410
0	1.000	1.000	1.000	366

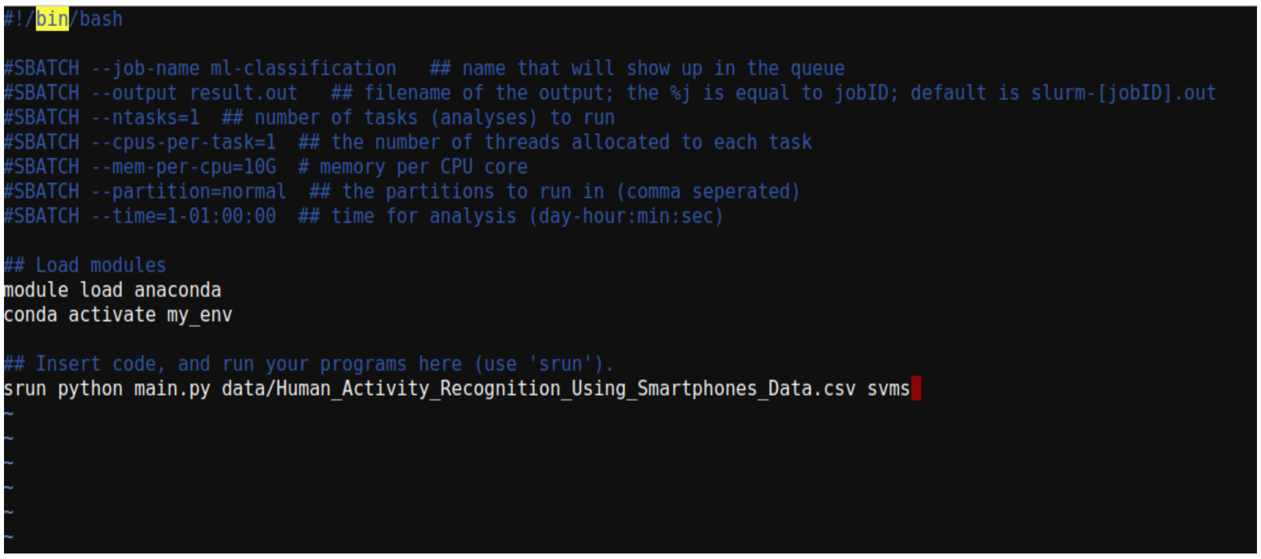


.

Things yet to be implemented(Future plan)

Existing models(SVM-Linear, SVM-NonLinear, Decision tree)	Consider overfitting issue, find more optimum value for the models(maxDepth in case of decision tree, C value in case of SVM)
Ensemble methods	Bagging, Boosting, Random Forest classifier(Boost the accuracy further)
More classification models	Naive Bayes classifier, Logistic regression, KNN
Improving the running time	Implementing it in NMSU supercomputer Discovery cluster by using anaconda environments

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:



```
#!/bin/bash

#SBATCH --job-name ml-classification   ## name that will show up in the queue
#SBATCH --output result.out   ## filename of the output; the %j is equal to jobID; default is slurm-[jobID].out
#SBATCH --ntasks=1  ## number of tasks (analyses) to run
#SBATCH --cpus-per-task=1  ## the number of threads allocated to each task
#SBATCH --mem-per-cpu=10G  # memory per CPU core
#SBATCH --partition=normal  ## the partitions to run in (comma seperated)
#SBATCH --time=1-01:00:00  ## time for analysis (day-hour:min:sec)

## Load modules
module load anaconda
conda activate my_env

## Insert code, and run your programs here (use 'srun').
srun python main.py data/Human_Activity_Recognition_Using_Smartphones_Data.csv svms
~
~
~
~
~
~
```

The above script shows how we can request resources in the cluster and run our jobs. We are still working on optimizing the slurm script and figuring out how to run in the cluster and get the results by testing our models via parallelism. For the final report and demo, we want to accomplish this and run our project in the supercomputer successfully.

Thank you!!!Thank you!!!