Applied Machine Learning 1

JunYong Lee
Preeti Maurya
Chidambaram Crushev

Activity Recognition | Final Report

NM STATE UNIVERSITY

BE BOLD. Shape the Future.

Problem Description

- ➢ Using data collected from activities recognized on smartphones
- ➢ What kind of activities a person doing? (Standing, Sitting, Laying, Walking, etc.)
- ➢ Classification algorithms, which are supervised machine learning algorithms, can be done on existing dataset to train / predict the classes for new data

1. What is the activity of an individual based on the smartphone data?
   - We can perform binary and multi-class classification
2. What is the accuracy of different machine learning models? How to improve the accuracy?
3. Since the data is huge, how can we improve the running time of the algorithm?
   - We will be using NMSU supercomputer (Discovery cluster) for more computing power that could decrease the running time

DataSet Information

- Data size: 10299 X 562
- The last column is the target label column which is named as 'Activity' (LAYING, SITTING, STANDING, WALKING, WALKING_DOWNSTAIRS, WALKING_UPSTAIRS)
- The feature columns : Data of certain activity from accelerometer and gyroscope

tBodyAcc-mean()-	tBodyAcc-mean()-Z	tBodyAcc-std()-X	...	Activity
-0.43183	-0.47637	-0.38656	...	STANDING
-0.33836	-0.46501	-0.39758	...	STANDING
-0.28965	-0.33084	-0.27765	...	SITTING
-0.03894	-0.34411	-0.67423	...	WALKING

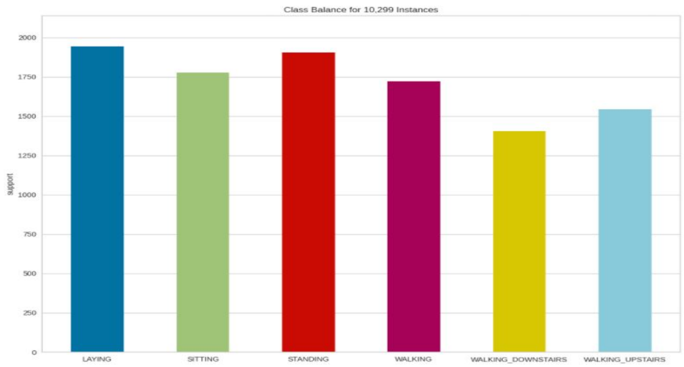
Data Preprocessing

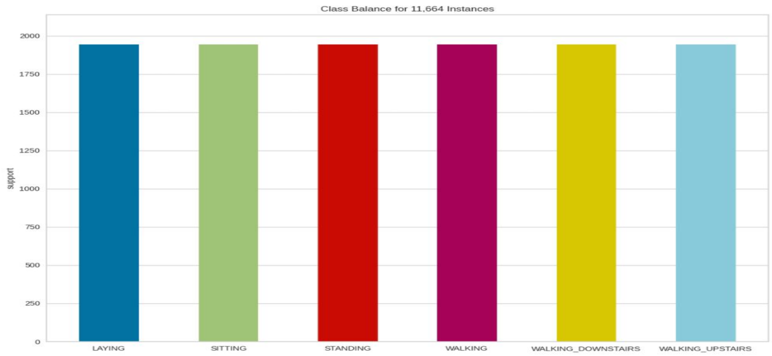
1. Replacing Null/NA values : **df.isna().values.any(), df.isnull().values.any()**
2. SMOTE (Synthetic Minority Oversampling Technique) algorithm

Original:

LAYING	SITTING	STANDING	WALKING	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS
1944	1176	1905	1722	1406	1544

LAYING	SITTING	STANDING	WALKING	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS
1944	1944	1944	1944	1944	1944





Data Preprocessing(Contd.)

3. Label encoding (LabelEncoder)

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

4. Standardizing the dataset : StandardScaler() from sklearn library

```
sc_X = preprocessing.StandardScaler()
X_trainscaled = sc_X.fit_transform(X_train)
X_testscaled = sc_X.transform(X_test)
```

5. Splitting the data into 80% of training set and 20% of testing set

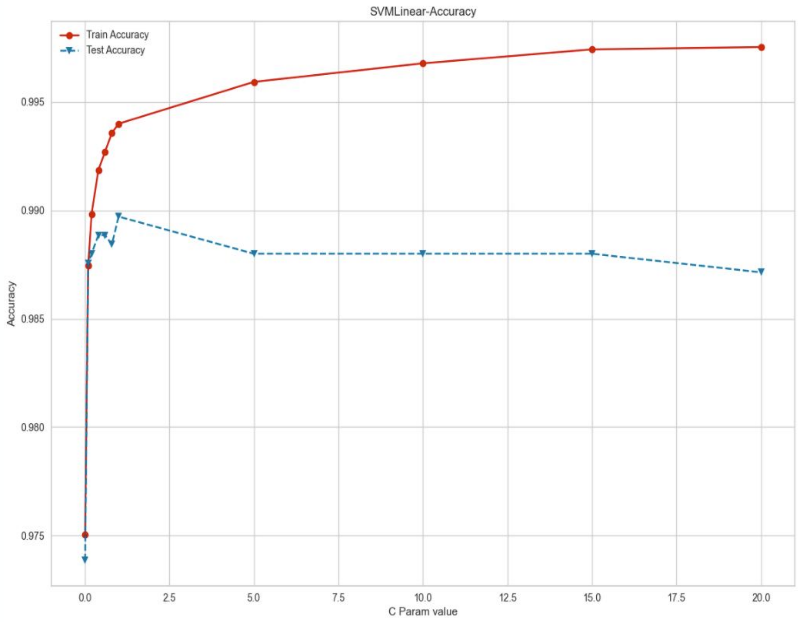
```
X_train, X_test, y_train, y_test = train_test_split(X_1_df,Y_1_df,random_state=1, test_size=0.2)
```

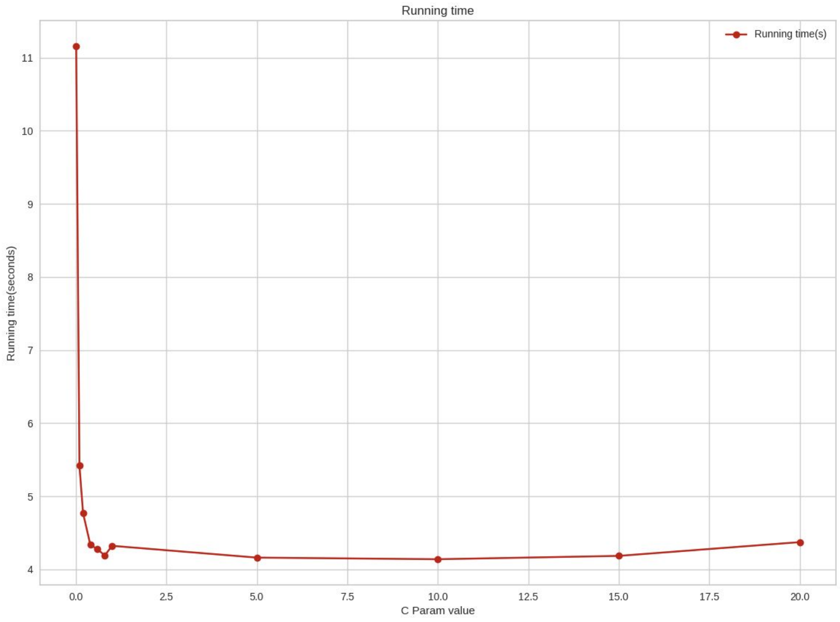
SVM (Linear)

- SVM (Linear) is a linear model for classification and regression problems. The SVM (linear) algorithm creates a line or a hyperplane which separates the data into classes.
- We ran the SVM model to get the test/train accuracies and running times for different values of C as [0.01, 0.1,0.2, 0.4, 0.6, 0.8, 1.0, 5, 10, 15, 20] .

```
c= [0.01, 0.1,0.2, 0.4, 0.6, 0.8, 1.0, 5, 10, 15,20]
for i in c:
    #capture the start time
    start = time.time()
    #fit the training dataset to linear kernel model
    svc = SVC(kernel = 'linear', gamma=0.7, C=i, random_state=1)
    svc.fit(X_train, y_train.values.ravel())
    y_pred_linear = svc.predict(X_test)
    y_train_pred_linear = svc.predict(X_train)
```

SVM Linear(Contd)





Max Accuracy = 0.990

Average Running Time = 5.029 seconds

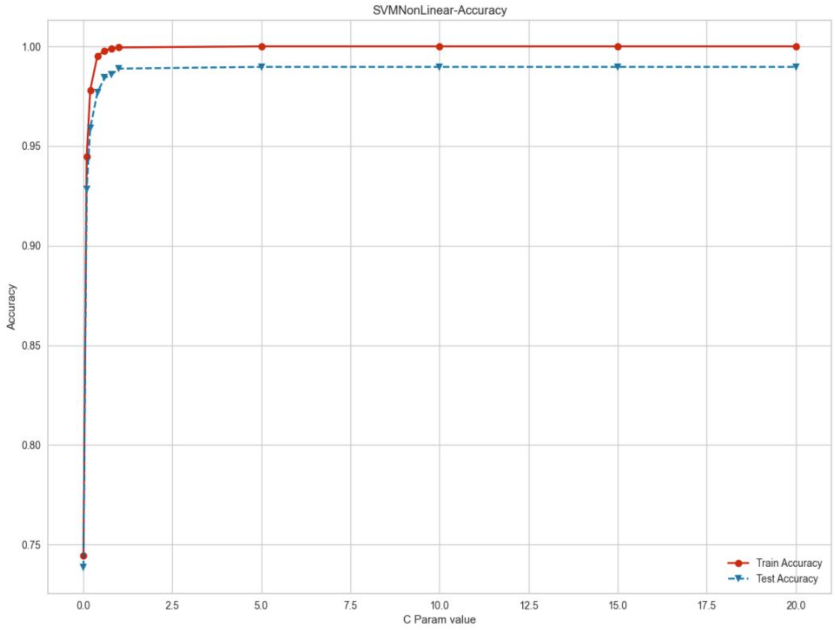
SVM (Non-Linear)

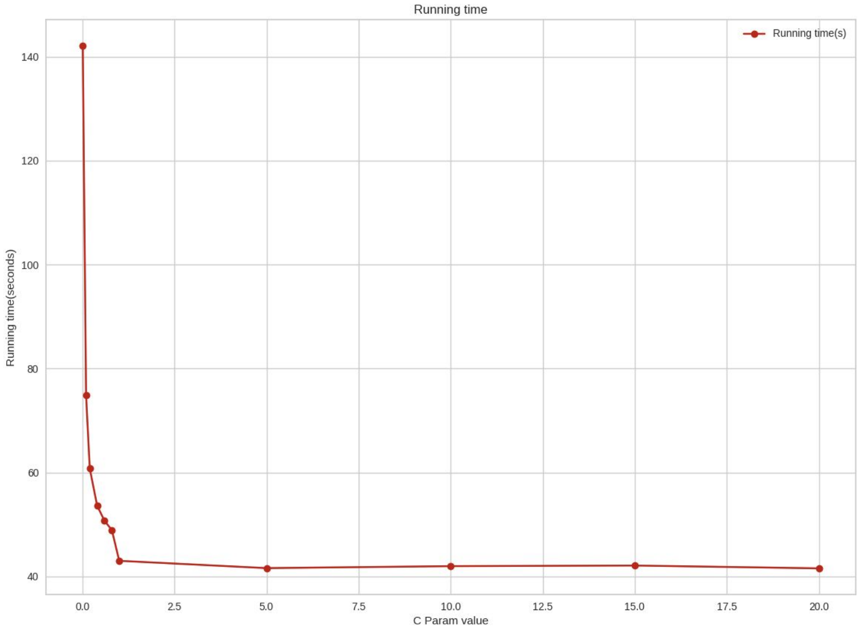
- SVM (Non-Linear) is a non-linear model for classification problems. The kernel function in non-linear model helps in transforming data from non-linear spaces into another dimension so that the data can be classified.
- We ran the SVM Non-Linear model to get the test/train accuracies and running times for different values of C as [0.01, 0.1,0.2, 0.4, 0.6, 0.8, 1.0, 5, 10, 15, 20] and kernel type “rbf” .

```
#hyperparameters test set
c= [0.01, 0.1,0.2, 0.4, 0.6, 0.8, 1.0, 5, 10, 15,20]
for i in c:
    # fit the training dataset to rbf kernel model
    # capture the start time
    start = time.time()
    rbf_svc = SVC(kernel = 'rbf', gamma=0.1, C=i, random_state=1)
    rbf_svc.fit(X_train, y_train.values.ravel())

    y_pred_rbf = rbf_svc.predict(X_test)
    y_train_pred_rbf = rbf_svc.predict(X_train)
```

SVM (Non-Linear)(Contd)





Max Accuracy = 0.990

Average Running Time= 58.285 seconds

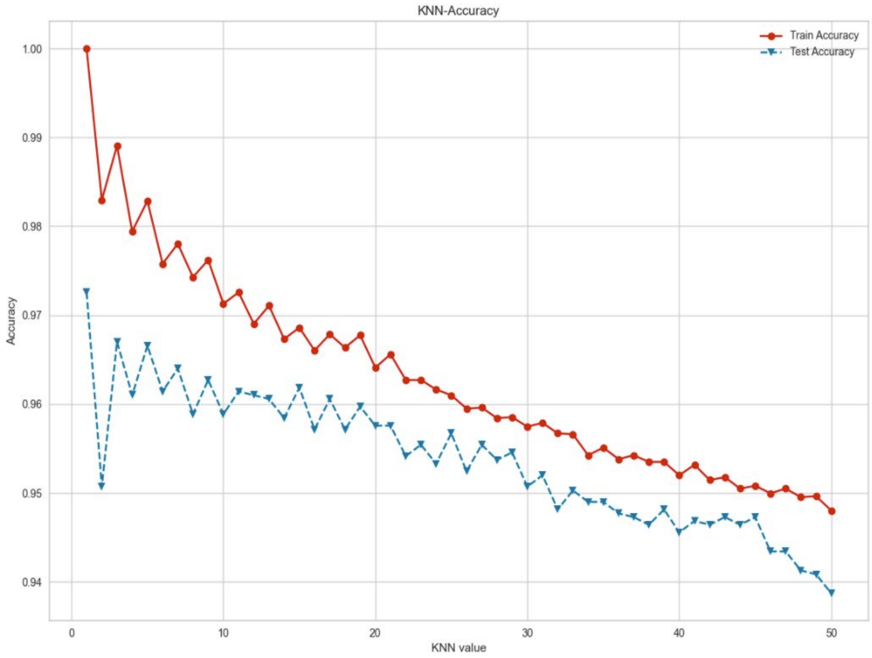
KNN

- KNN (K-Nearest Neighbors) assumes the related data-points reside in closest proximity of each other. KNN algorithm can be used to solve classification and regression problems.
- We ran the KNN model to get the test/train accuracies and running times for different values of n_neighbors as count from 1 to 51. K best value is found by using the Cross Validation method.

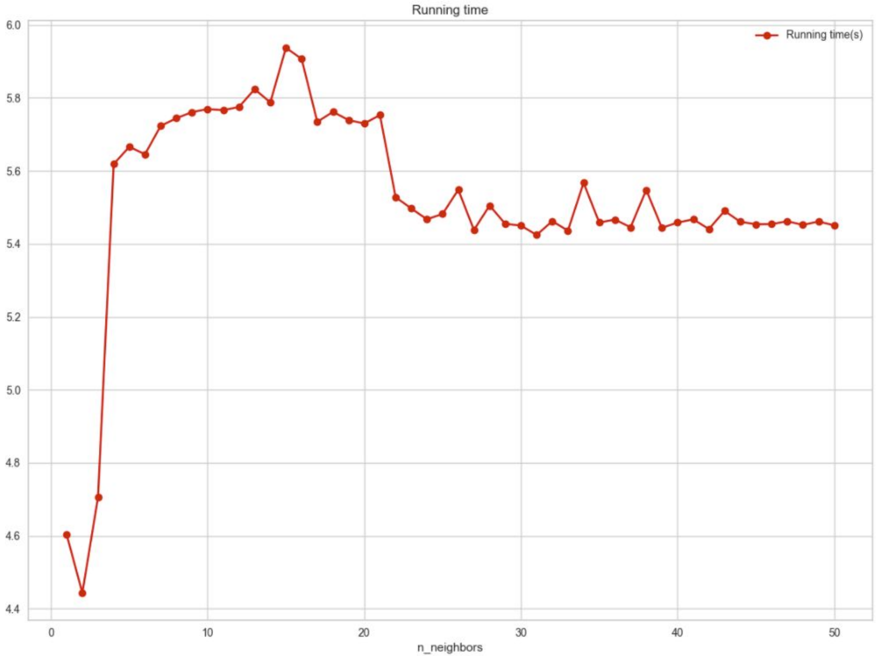
```
c = list(range(1, 51))

for i in c:
    # capture the start time
    start = time.time()
    neigh = KNeighborsClassifier(n_neighbors=i)
    scores = cross_val_score(neigh, X_1_df, Y_1_df.values.ravel(), cv=10, scoring='accuracy')
```

KNN (Contd.)



Max Test Accuracy = 0.973



Average Running Time= 5.521 seconds

Logistic Regression

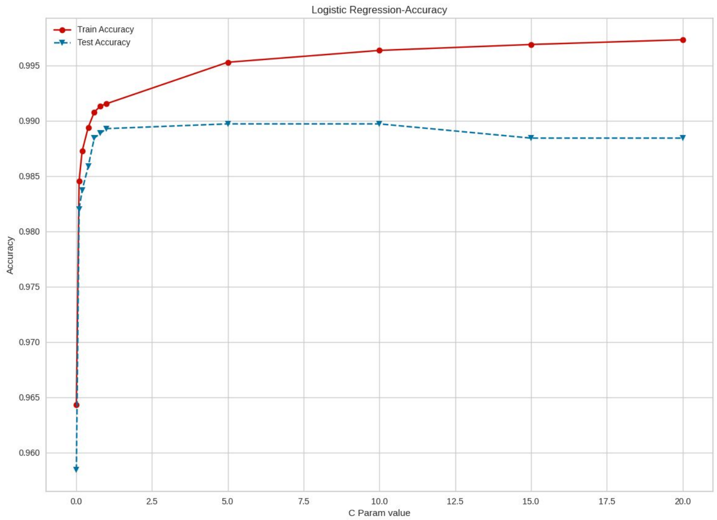
- Logistic Regression was implemented for the following parameter values, c=[0.01, 0.1,0.2, 0.4, 0.6, 0.8, 1.0, 5, 10, 15,20] , max_iteration = 10000, class_weight = “balanced”

- **Output**

  Average Running time = 36.292 seconds,

  Corresponding C param value = 5.0,

  max Accuracy = 0.990



Decision Tree Classifier

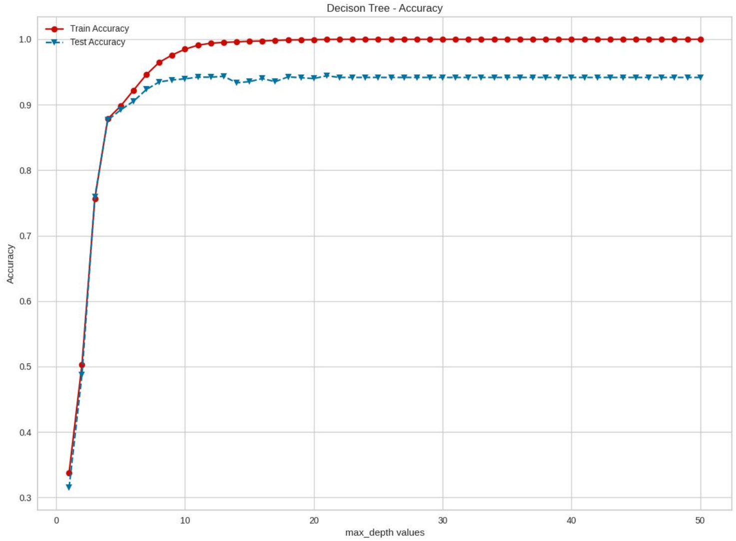
- Decision Tree classifier was implemented with the following params
- Parameters set - criterion = “gini”, MaxDepth = 1 to 50

- **Output**

  Max Accuracy = 0.944,

  Corresponding MaxDepth = 21

  Average Running time = 4.958 seconds



Random Forest

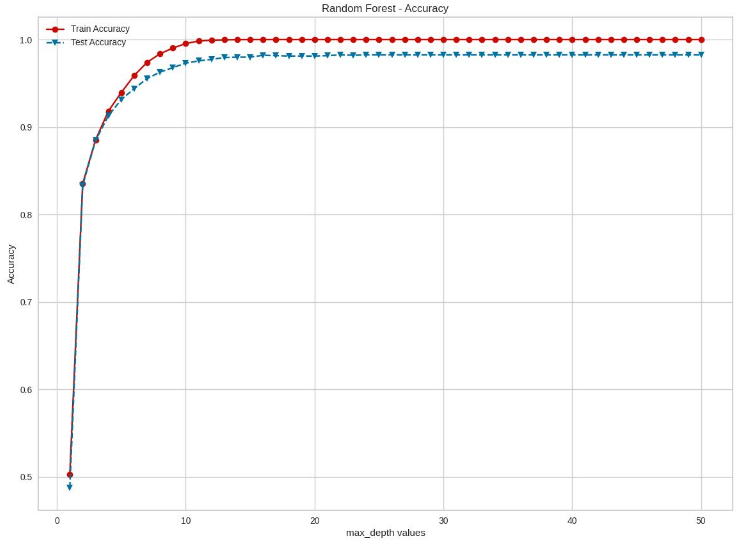
- To improve the accuracy of the decision tree further, random forest classifier was implemented
- Parameters set - Number of estimators = 500, criterion = “gini”, MaxDepth = 1 to 50

- **Output**

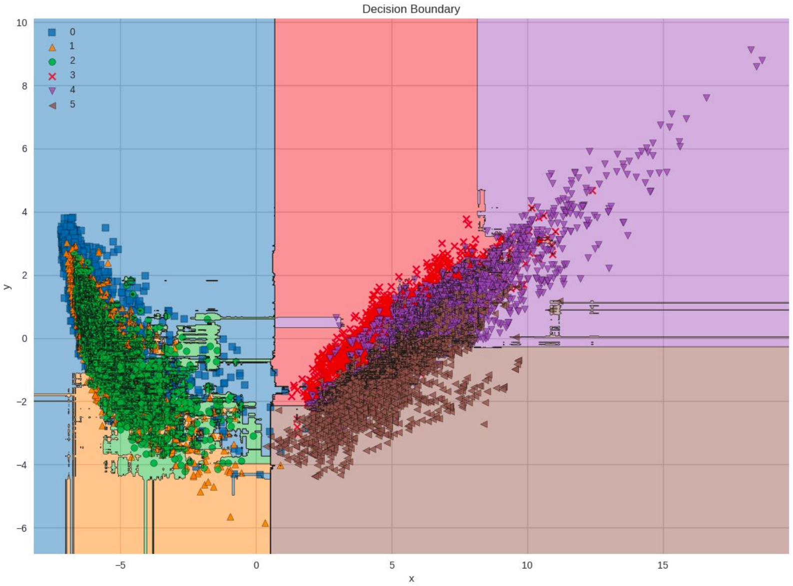
  Max Accuracy = 0.982,

  Corresponding MaxDepth = 22

  Average Running time = 56.245 seconds



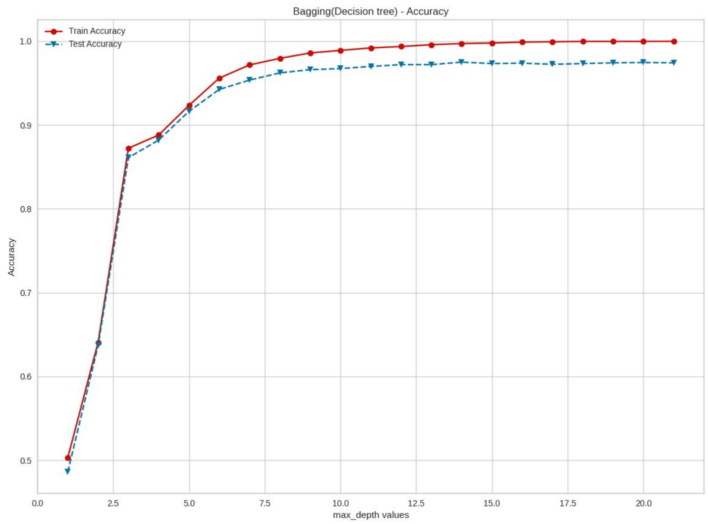
Random Forest(Contd.)



Bagging

- Bagging was also implemented to see whether accuracies can be improved for Decision tree classifier
- Bagging Classifier Parameters, n_estimators = 1000
  base classifier = Decision Tree

- **Output**
  Max Accuracy - 0.975 (improved from 0.944)
  Average Running Time - 225.824 seconds



AdaBoost

- AdaBoost method was chosen to improve the accuracy for Decision Tree Classifier.
- Parameters - number of estimators = 100, base estimator = Decision Tree, maxDepth value = 21(chosen from Decision tree)
- **Output**

  Max Accuracy - 0.954, Running Time = 24.273 s
- The decision tree accuracy has been improved from 0.944 to 0.954 using adaboost classifier

```
clf = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=i), random_state=1, n_estimators=100)
clf.fit(X_train, y_train.values.ravel())

y_pred = clf.predict(X_test)
y_train_pred = clf.predict(X_train)
```

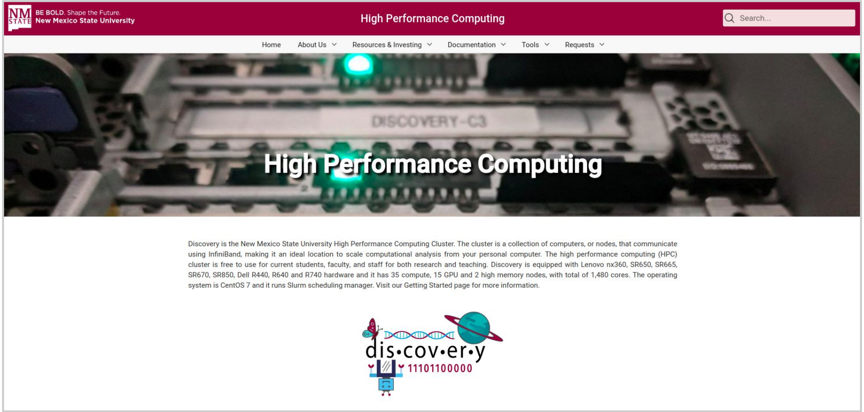
Ensemble Vote Classifier

- ➢ Ensemble Vote Classifier combines classifiers such as “Logistic Regression”, “SVM - Linear”, “K Nearest Neighbors”, “Decision Tree”
- ➢ The classifiers then predict the values by a “**Majority Voting**” technique
- ➢ **Output**

```
Ensemble Vote Classifier was Called. Wait...
Accuracy: 0.98 (+/- 0.02) [Logistic Regression]
Accuracy: 0.98 (+/- 0.02) [Support Vector Machine]
Accuracy: 0.95 (+/- 0.02) [K Nearest Neighbor]
Accuracy: 0.91 (+/- 0.04) [Decision Tree]
Accuracy: 0.98 (+/- 0.02) [Ensemble]
```

How did we execute ?

- Since our dataset is large and there are lot of classification models with so many iterations to run, we used the **NMSU Supercomputer Cluster(Discovery)**
- Completed a course to get our accounts created in the cluster.
- We ran the models using the custom-created **Anaconda environments** which solved many dependency issues
- Finally, we submitted it to **Slurm(Job Scheduler)** in the cluster to run our machine learning job, get the results.



How did we execute(Contd.) ?

```
#!/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=10  ## number of tasks (analyses) to run
#SBATCH --cpus-per-task=2  ## the number of threads allocated to each task
#SBATCH --mem-per-cpu=5G  # memory per CPU core
#SBATCH --partition=class  ## 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



srun="srun --ntasks=1 --nodes=1 --exclusive "
$srun python main.py data/Human_Activity_Recognition_Using_Smartphones_Data.csv svm > results/svm.out &
$srun python main.py data/Human_Activity_Recognition_Using_Smartphones_Data.csv svmnonlinear > results/svm_non_linear.out &
$srun python main.py data/Human_Activity_Recognition_Using_Smartphones_Data.csv decisiontree > results/decision_tree.out &
$srun python main.py data/Human_Activity_Recognition_Using_Smartphones_Data.csv logisticregression > results/lr.out &
$srun python main.py data/Human_Activity_Recognition_Using_Smartphones_Data.csv knn > results/knn.out &
$srun python main.py data/Human_Activity_Recognition_Using_Smartphones_Data.csv randomforest > results/randomforest.out &
$srun python main.py data/Human_Activity_Recognition_Using_Smartphones_Data.csv bagging > results/bagging.out &
$srun python main.py data/Human_Activity_Recognition_Using_Smartphones_Data.csv ensemblevote > results/ensemble.out &
$srun python main.py data/Human_Activity_Recognition_Using_Smartphones_Data.csv adaboost > results/adaboost.out &
$srun python main.py data/Human_Activity_Recognition_Using_Smartphones_Data.csv naivebayes > results/naivebayes.out &
wait
```

Running Time & Accuracy

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!