USC University of Southern California

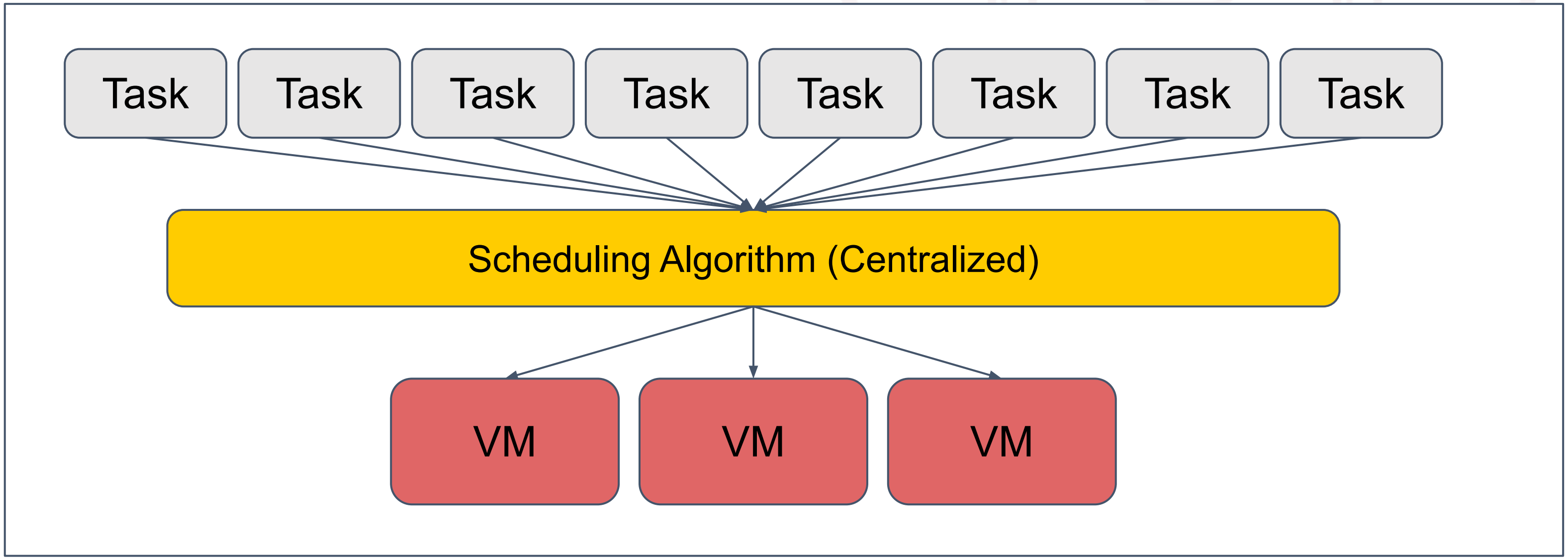
EE 454
Project Phase III

Steve Hornyak, Victor Hui, and Joshua Williams

Part I: Survey of Papers

Problem Statement

- Task Scheduling in the OS
- Many trade-offs
  - Resource utilization
  - Energy consumption
  - Operating cost
  - Scalability
  - Execution time
- Using AI/ML to provide better results
  - Moving away from simple algorithms (like RR)



Introduction to Papers Surveyed

[1] Deep and Reinforcement Learning for Automated Task Scheduling in Large-Scale Cloud Computing Systems

- By Rjoub Gaith, Jamal Bentahar, Omar Abdel Wahab, Ahmed Saleh Bataineh
- 4 different ML techniques for optimizing CPU and RAM utilization
  - Deep Reinforcement Learning with Long short-term Memory (DRL-LSTM)
  - Reinforcement Learning (RL)
  - Deep Q-Networks (DQN)
  - Recurrent Neural Network Long Short-Term Memory (RNN-LSTM)

[2] H2O-Cloud: A Resource and Quality of Service-Aware Task Scheduling Framework for Warehouse-Scale Data Centers - A Hierarchical Hybrid DRL (Deep Reinforcement Learning) based Approach

- By Mingxi Cheng, Ji Li, Paul Bogdan, and Shahin Nazarian
- Cloud computing for larger scale data centers
- Optimizing quality of service (QoS) and energy consumption
- Uses a Hierarchical Hybrid Deep Reinforcement Learning method

[3] DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers

- By Mingxi Cheng, Ji Li, and Shahin Nazarian
- Designed for larger scale data centers
- Optimize for energy consumption, rejection rate and runtime
- Uses the Novel Deep Reinforcement Learning method

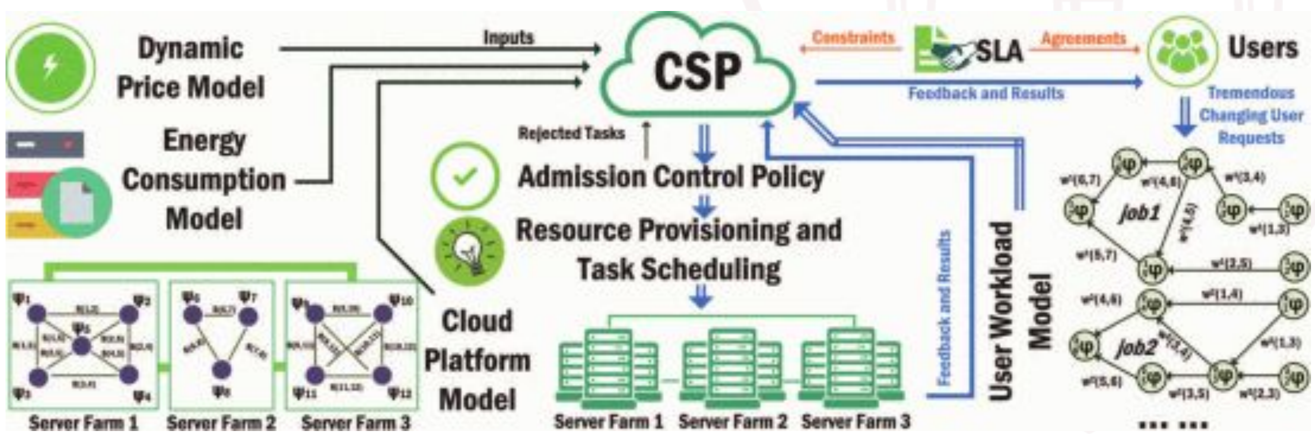
USC

Comparison of the Methods

	DRL-LSTM	H2O-Cloud	DRL-Cloud
Algorithm Type	Machine Learning	Machine Learning	Machine Learning
Learning Type	• Reinforcement Learning (RL) • Deep Q-Networks (DQN) • Recurrent Neural Network Long Short-Term Memory (RNN-LSTM) • Deep Reinforcement Learning with LSTM (DRL-LSTM)	• Hierarchical Hybrid Deep Reinforcement Learning • Deep Q-Network	• Novel Deep Reinforcement Learning
Metrics of Opt.	CPU usage cost, RAM usage cost	Energy cost efficiency, Energy efficiency, Reward rate	Energy cost efficiency, Rejection rate, Runtime
Runtime/Complexity	Worst	Best	Better
Data/Tool Used	Google cluster dataset	Google cluster usage traces	Google cluster usage traces
Trade-off	Longer Execution Time	Not exceptional in low variance, small sets of data	Not optimal for small data sets
Published	2020	2019	2018
Journal	Concurrency and Computation	TCAD	ASP-DAC
Paper Number	[1]	[2]	[3]

Weighing the Options

- Each method had its advantages and drawbacks
  - Dependent on what is being optimized in the study
- Input affects the output results, i.e. large vs small amount of data, large vs minimal variance, CPU dependent data vs I/O dependent data
- The best task scheduling implementation is dependent on circumstances:
  - What are the inputs?
  - What must be optimized?
  - What can be sacrificed (trade-offs)?



From Source [3]

Part II: ML Implementation

Our ML Implementation

- Many AI/ML scheduling papers are about burst time estimation
  - Difficult to determine how long a task needs to run
  - Tasks do not provide burst time in a RTOS
- Our greedy turnaround time algorithm is a function of burst time
  - Optimal average turnaround time: priority is the inverse of remaining burst time
  - Sorted next-fit Greedy approach
- Our ML algorithm: burst time by job type
  - Use ML to guess job type
- Fun way to apply what we learned in our limited time

Examining different Classification Models

Dataset

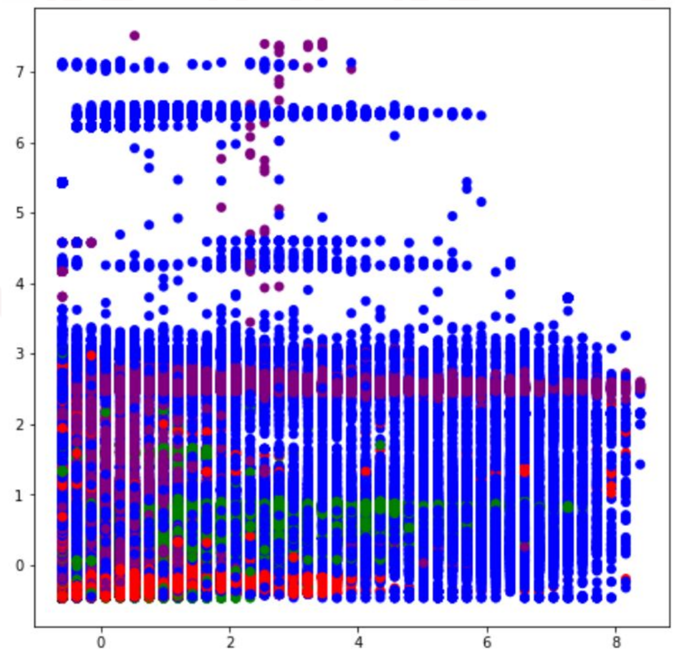
Time	ParentID	TaskID	JobType	NrmlTaskCores	NrmlTaskMem
90000	757745334	1488529826	0	0	0.0311296
90000	975992247	1488529821	0	0	0
90000	1468458091	1488529832	1	0.021875	0.00235309
90000	1460281235	1488529840	0	0	0
90000	1164728954	1488529835	0	0.003125	0.0016384
90000	1288997448	1488529848	0	0.003125	0.0049152
90000	1488529845	1488529847	1	0.003125	0.000719232
90000	1263655469	1488529844	2	0 0	

- The Google Cluster Data consists of 4 job types (0-3)
- Features: NrmlTaskCores and NrmlTaskMem
- New Burst Time Calculation:

(TaskID % 10 + 1) * 300

↓

(JobType + 1) * 300

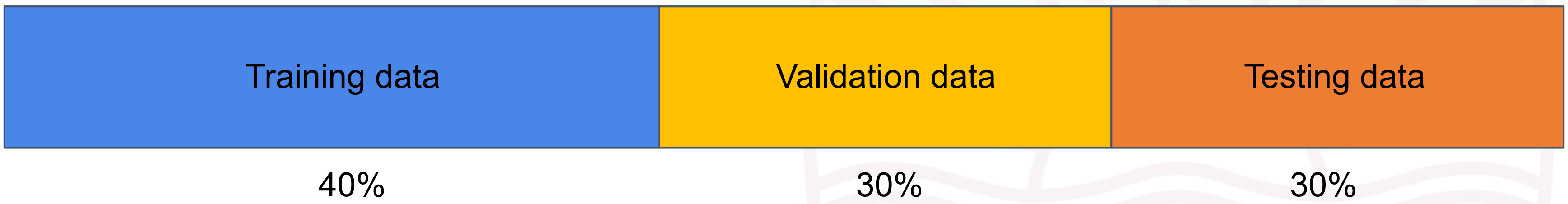


USC

Classification Models

- Examples: Logistic Regression, K Nearest Neighbors, State Vector Machines, Perceptron.
- Given a feature vector X and a qualitative response Y, the task of a classification model is to build a function C(X) that takes as input the feature vector X and predict its value for Y
- We need to perform Multiclass Classification.
- Models tested: Multiclass Logistic Regression, K Nearest Neighbors, Decision Tree Classifier
- Implemented using Sklearn

Train, validation, test data split

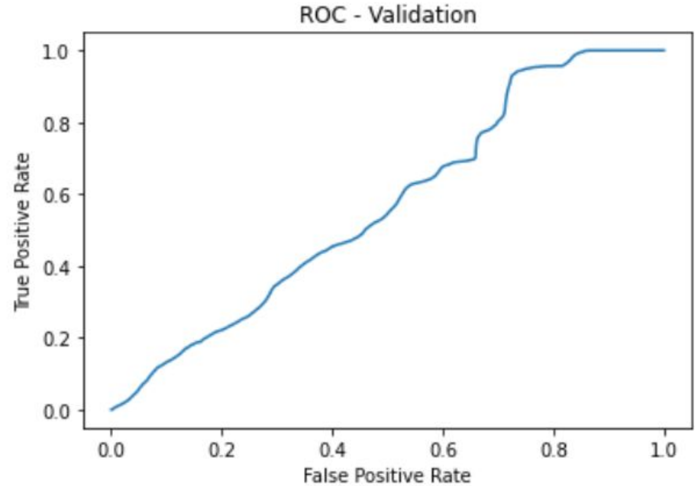


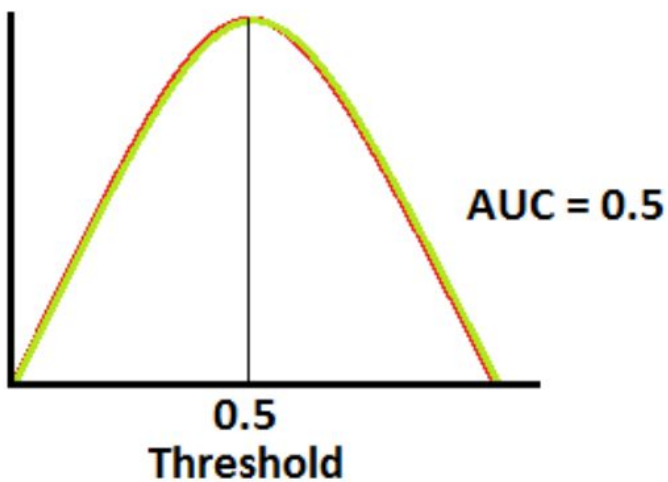
- Since we have an abundance of data points, we can afford to split our dataset three-way
- The dataset is split in the following way:
  - Denote “JobType 0” data as J0, “JobType 1” data as J1, etc. Select the first 40% of J0, 40% of J1, 40% of J2, and 40% of J3 and place it in the training set.
  - Split the remainder into the validation and testing set.
  - Note: this does not mean the classes are balanced in the sets. We will explore this later

Multinomial Logistic Regression

	Precision	Recall
Training	0.56575	0.9751997
Validation	0.5671757	0.9646386

- The ROC curve shows the model has no discrimination capacity to distinguish between positive class and negative class, i.e. we need to find a different model.

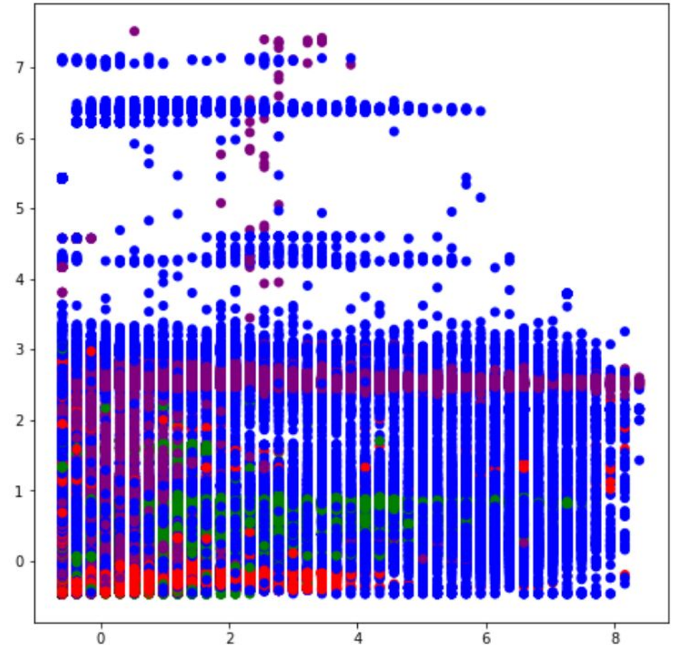


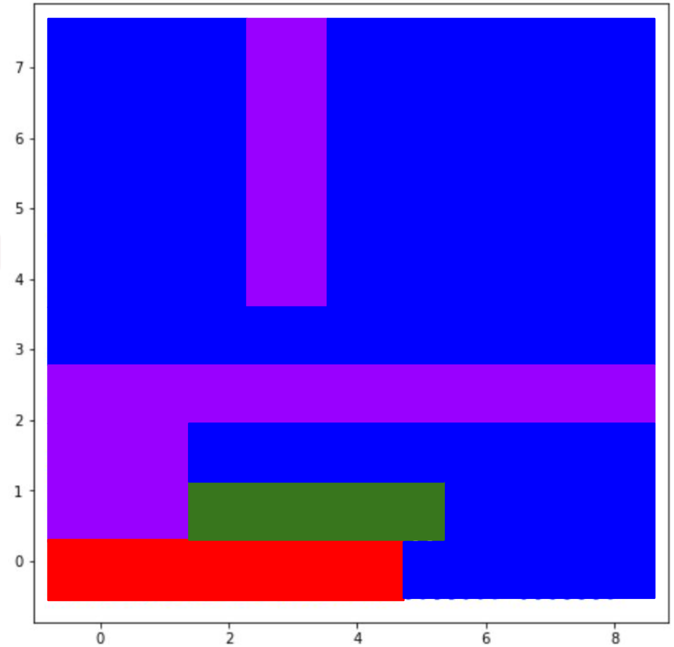


ROC curve is a curve of probability distribution

Decision Tree Classification

- By plotting features against each other, it seems like a decision tree classifier would be an ideal classification model
- Example by simply eyeballing

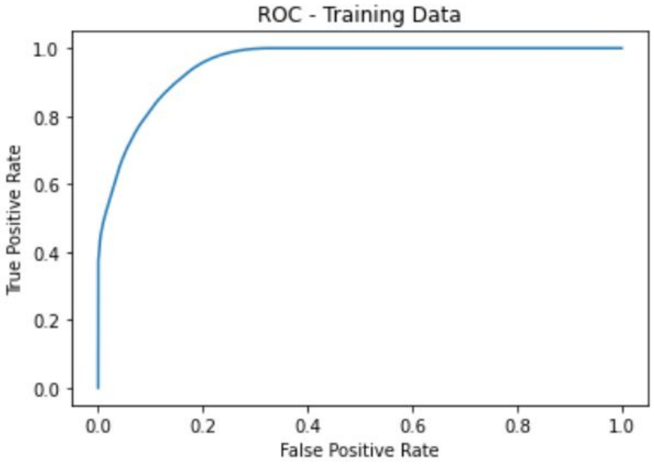


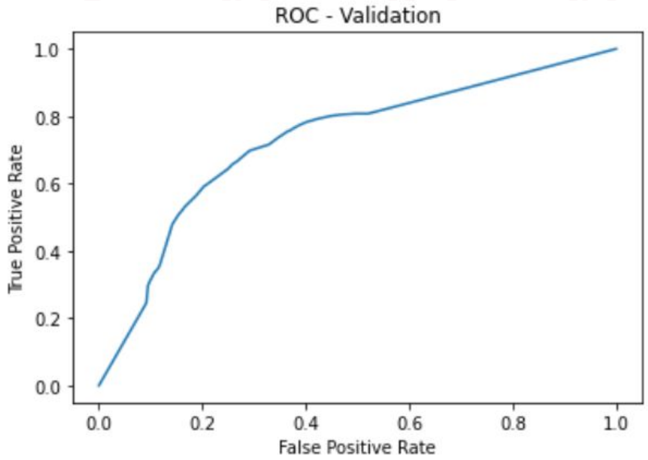


Decision Tree Classification

	Precision	Recall
Training	0.72941	0.8104
Validation	0.72008	0.61993

- Training ROC outperforms Multinomial Logistic Regression but Validation ROC is still undesirable
- It is known that trees generally do not have the same level of predictive accuracy as other classification approaches, since they tends to over simplify.



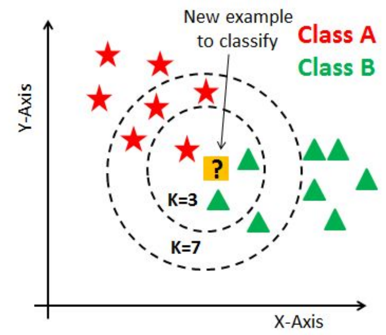


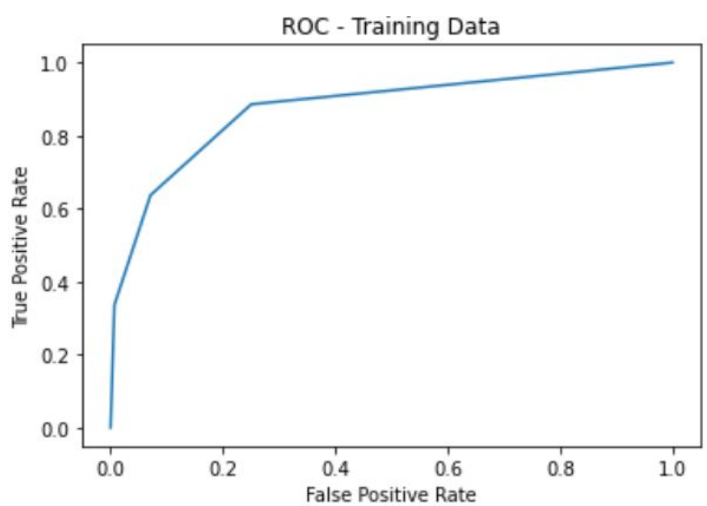
Higher area under curve = better

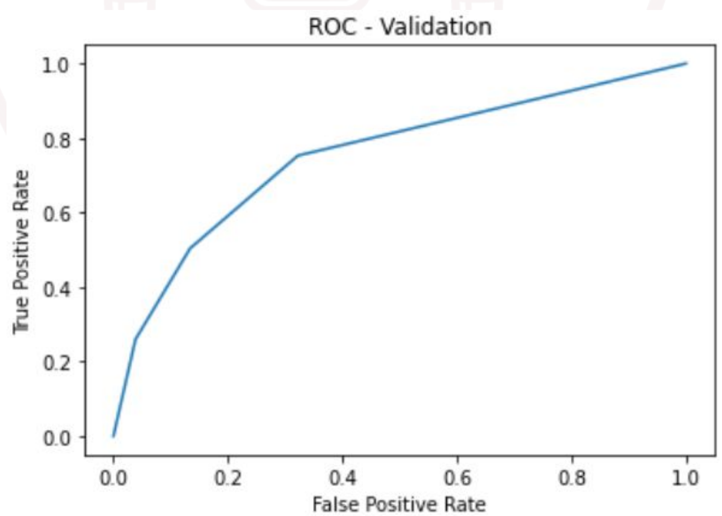
K Nearest Neighbors K = 3

	Precision	Recall
Training	0.74759	0.82414
Validation	0.72354	0.6270

- Since we are only using 2 features, we do not need to worry about the curse of dimensionality when dealing with high dimensional data
- K = 3 performance comparable to Decision Tree Classifier, slightly superior performance with Validation data





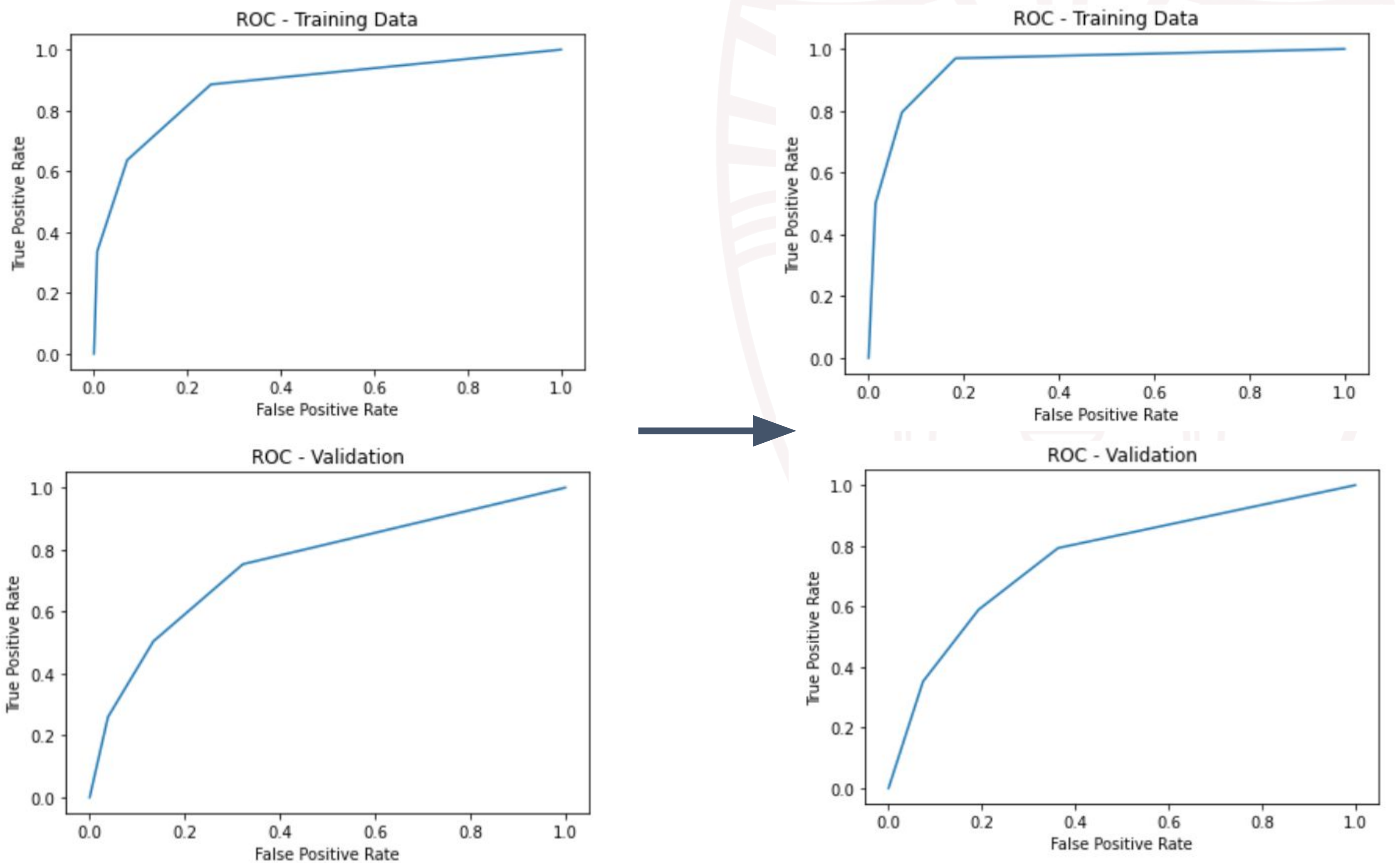


Higher area under curve = better

KNN after resampling

- As we mentioned previously, classes are imbalanced, since 0.4J0 is significantly larger than 0.4J2 and 0.4 J3.
- We can attempt to resample the data by either downsampling or upsampling.
- Here, we downsamples J0, J1, and J3 to the size of J2/2 (which has the lowest number of samples).
- Since the size of the dataset is so large, J2 has also been down sized to J2/2 to avoid overfitting.
- There are other sampling techniques such as SMOTE which could be explored in the future

3 Nearest Neighbors (Downsampled)



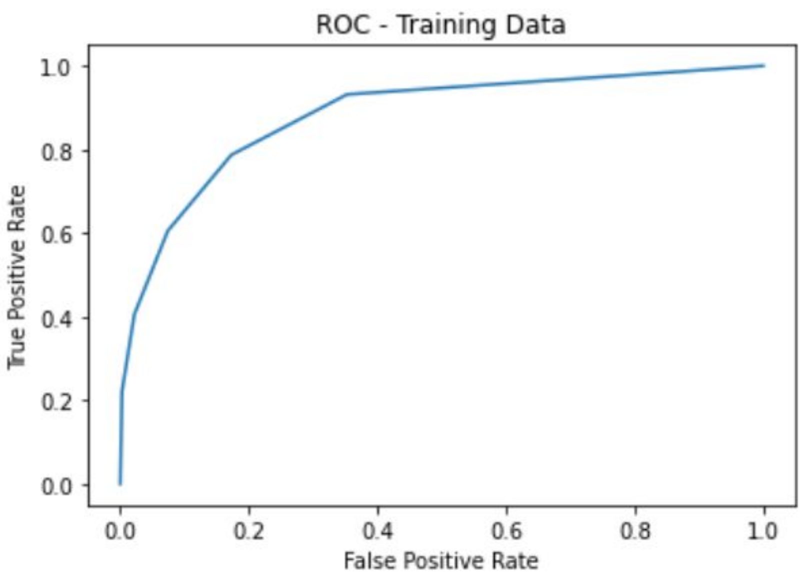
Higher area under curve = better

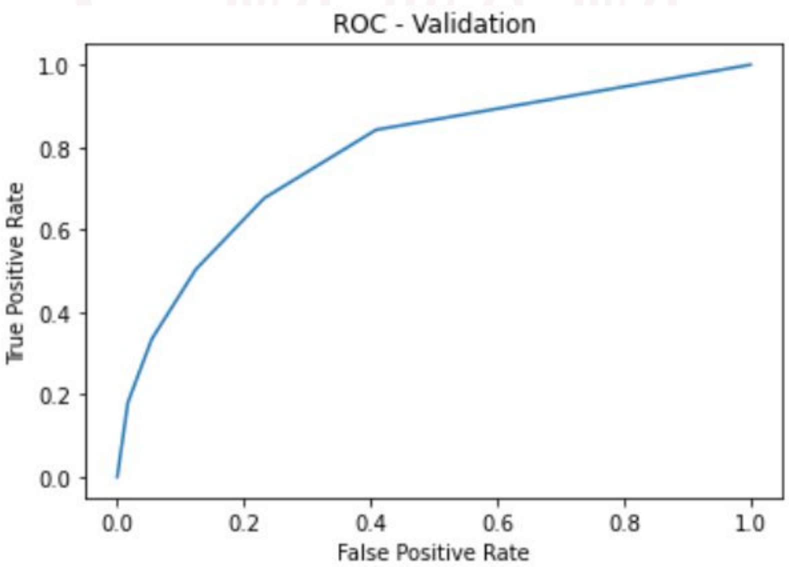
USC

K Nearest Neighbors K = 5

- Finally, we opted for a K Nearest Model with 5 neighbors.

	Precision	Recall
Training	0.74779	0.89924
Validation	0.68185	0.80725
Testing	**0.96367**	0.80805





Higher area under curve = better

Combining with Greedy Algorithm

Changes to Code

- Add the KNN ML to predict job type
- Modified the burst time calculation
- Changed the VM resources
  - Original resources had no rejections
  - One setup with a small number of soft rejections
  - One setup with a large number of soft rejections
- Modified the turnaround time Greedy Algorithm
  - Priority based on predicted burst time remaining

```python
# Function that initializes the task data using ML
# Using 5 Nearest Neighbors
def init_df_ml(df):
    #train test split
    train0 = df[df["JobType"]==0].head((int)(len(df[df["JobType"]==0])*.4))
    train1 = df[df["JobType"]==1].head((int)(len(df[df["JobType"]==1])*.4))
    train2 = df[df["JobType"]==2].head((int)(len(df[df["JobType"]==2])*.4))
    train3 = df[df["JobType"]==3].head((int)(len(df[df["JobType"]==3])*.4))
    train = train0.append(train1)
    train = train.append(train2)
    train = train.append(train3)

    xtrain = train[["NrmlTaskCores", "NrmlTaskMem"]]
    ytrain = train["JobType"]

    #training a 5NN Classifier with 0.4J
    KNN = KNeighborsClassifier(n_neighbors=5)
    KNN.fit(xtrain, ytrain)

    X = df[["NrmlTaskCores", "NrmlTaskMem"]]
    y = df["JobType"]
    new_y = KNN.predict(X)
    df["JobType_pred"] = new_y

    job_types = df['JobType_pred'].values
    task_burst_time = [(job_type + 1) * TIME_QUANTUM for job_type in job_types]
    df['ML_burst_time'] = task_burst_time

    return df
```

USC

Results

```
Round Robin Stats:
        Total Energy: 127582064.5843506
        Total Cost: 75819357.26554874
        Total Turn Around Time (sum of the turn around time of every task): 2639356500
        Total Soft Rejections: 2103451
        Algorithm Execution Time (in seconds): 49.051570415496826
Greedy Turn Around Time Stats:
        Total Energy: 127458970.2850342
        Total Cost: 75725799.40269472
        Total Turn Around Time (sum of the turn around time of every task): 2194308000
        Total Soft Rejections: 619956
        Algorithm Execution Time (in seconds): 27.50543713569641
Naive Classification + Greedy Turn Around Time Stats:
        Total Energy: 127420566.91589357
        Total Cost: 75697179.36676027
        Total Turn Around Time (sum of the turn around time of every task): 2183164500
        Total Soft Rejections: 582811
        Algorithm Execution Time (in seconds): 107.00152063369751
```

	Setup 1	Setup 2
CPU Units	16	12
VM Memory Units	22	18
Total Turnaround Time	2183164500	3687246300
Total Soft Rejections	582811	5596417
Execution Time	107.001	188.721
TTT Improvement (RR)	17.28%	61.69%
Soft Rejection Impr. (RR)	72.29%	77.95%
TTT Improvement (Greedy)	0.51%	4.31%
Soft Rejection Impr. (Greedy)	5.99%	9.00%

Thank you!

Questions?