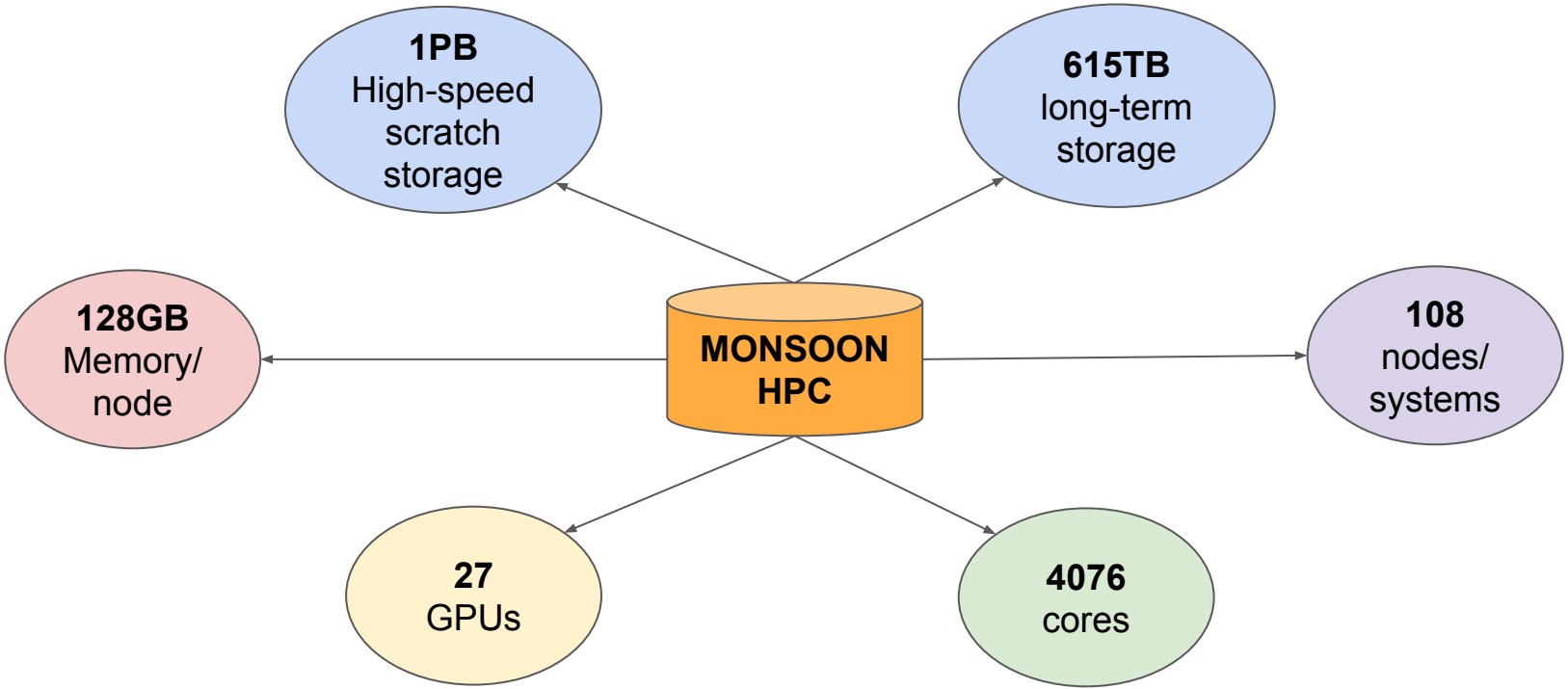
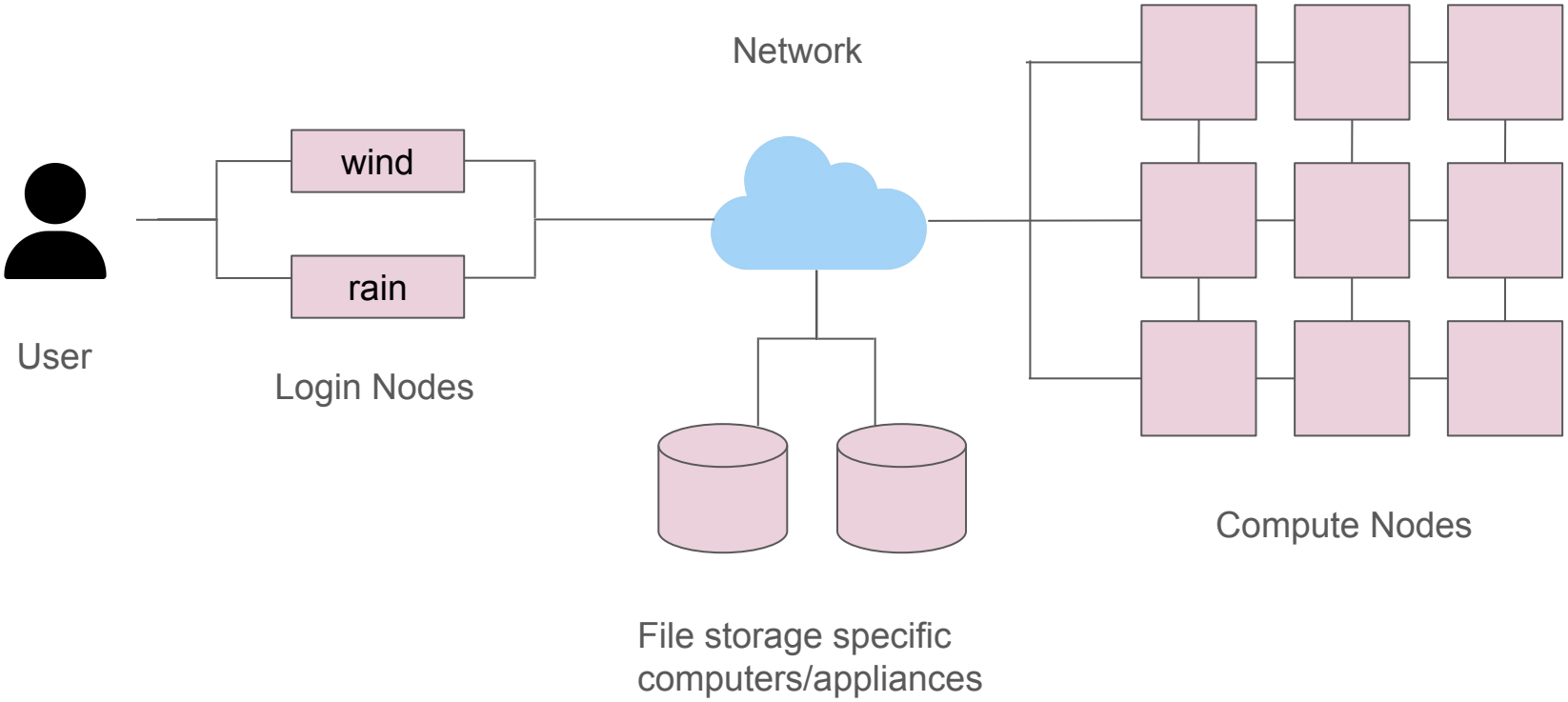
Using Monsoon in ML workflow

by Daniel Agyapong

Monsoon Today



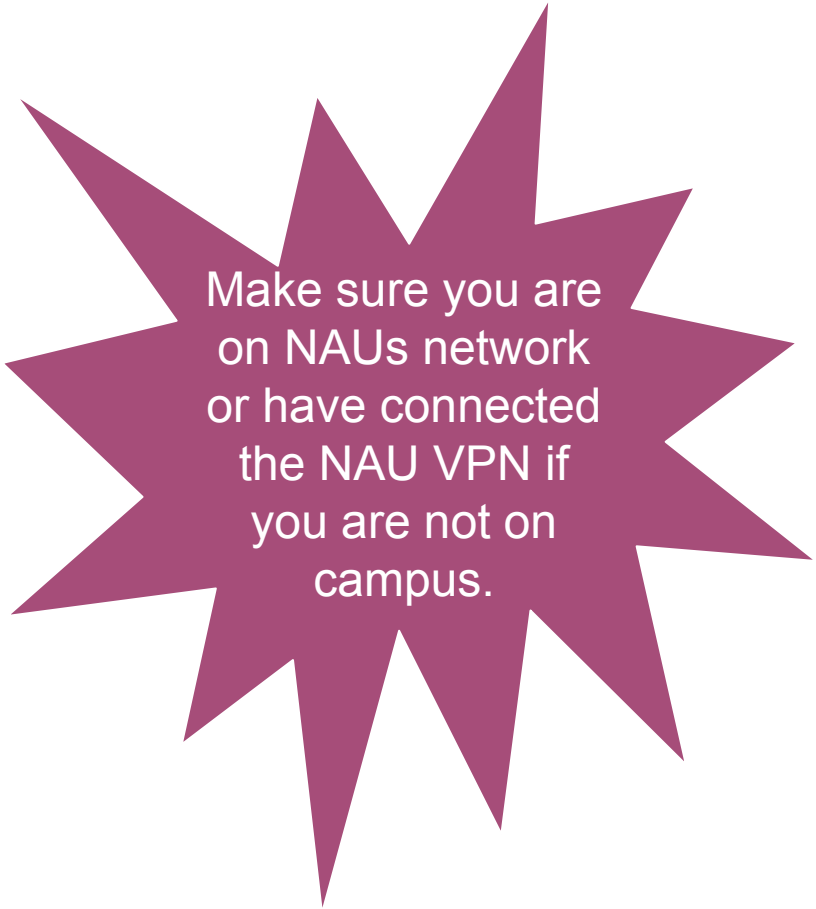
Cluster Topology



Connecting to Monsoon

- Access through the monsoon dashboard https://ondemand.hpc.nau.edu/pun/sys/dashboard/

- Access through the secure shell (ssh)
  On your terminal run: ssh -Y <username>@monsoon.hpc.nau.edu
  And type your password



Running code on Monsoon (Compute Node)

Interactive/Debug Work

srun -t 24:00:00 --mem=4GB --cpus-per-task=1 --pty bash

srun -t 1:00:00 --mem=8GB --cpus-per-task=1 python analysis.py

Submitting jobs

sbatch jobscript.sh

Example Job Script

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --output=/scratch/da2343/output.txt
#SBATCH --error=/scratch/da2343/error.txt
#SBATCH --time=20:00
#SBATCH --mem=1GB
#SBATCH --cpus-per-task=1
python  analysis.py
```

Machine Learning Overview

1 Supervised Learning

Goal is to learn a mapping function from the input data to the output data, such that it can predict the output for new input data.

Classification (Binary or Multiclass)
Regression

2 Unsupervised Learning

Goal is to learn the underlying structure of an input data without output labels.

Clustering
Dimensionality reduction

3 Reinforcement Learning

Goal is to train agents to make decisions in an environment to maximize cumulative rewards.

Game playing
Autonomous systems

Supervised Machine Learning

Parameters	Hyper-parameters
Learned by the model itself through optimizations like Stochastic Gradient Descent (SGD) or Adam E.g: Weights and biases	Selected before the model is trained. E.g Learning rate, epoch, number of layers, number of neurons per layer, activation function, etc

Optimum hyper-parameters help to learn model parameters.

Problem:
There are too many hyper-parameters to tune.

Some combinations work better than others.

Solution:
Parallelize the operation so that you can run multiple combinations at a time.

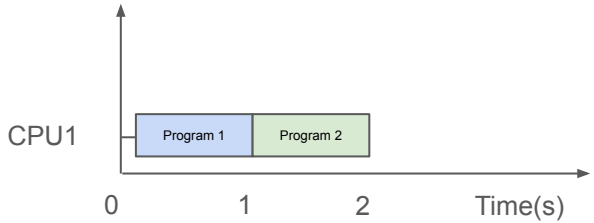
Featureless/Baseline

Featureless/Baseline model is a simple model which serves as reference.

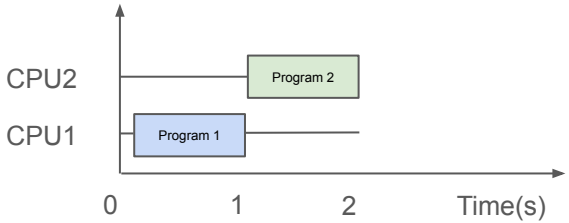
Classification - Just predict the **most frequent label** in train data.
Regression - Just predict the **mean label** in train data.

Your trained model should at least perform better than featureless.

Parallelism and Concurrency

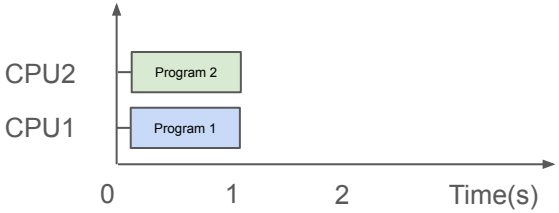


1 CPU, Sequential Execution



2 CPUs, Sequential Concurrent Execution

Typically, this is how the normal code we write is executed.



2 CPUs, Parallel Execution

This is how you want your code to be executed for higher speedups

Slurm Job Arrays for parallelism

Convenient way to run multiple similar jobs with one command, which can speed up your programs.

Works like a loop that performs a different task with the same resources and settings.

Slurm Job Arrays - Basic

test_one.sh

ml_project_3 > code > job_arrays_basic > test_one.sh

```
#!/bin/bash
#SBATCH --array=0-4
#SBATCH --time=1:00:00
#SBATCH --mem=8MB
#SBATCH --cpus-per-task=1
#SBATCH --error=/projects/genomic-ml/da2343/ml_project_3/code/job_arrays_basic/slurm-%A_%a.out
#SBATCH --output=/projects/genomic-ml/da2343/ml_project_3/code/job_arrays_basic/slurm-%A_%a.out
#SBATCH --job-name=job_arrays_basic
cd /projects/genomic-ml/da2343/ml_project_3/code/job_arrays_basic
python test_one.py $SLURM_ARRAY_TASK_ID
```

test_one.sh | test_one.py

ml_project_3 > code > job_arrays_basic > test_one.py

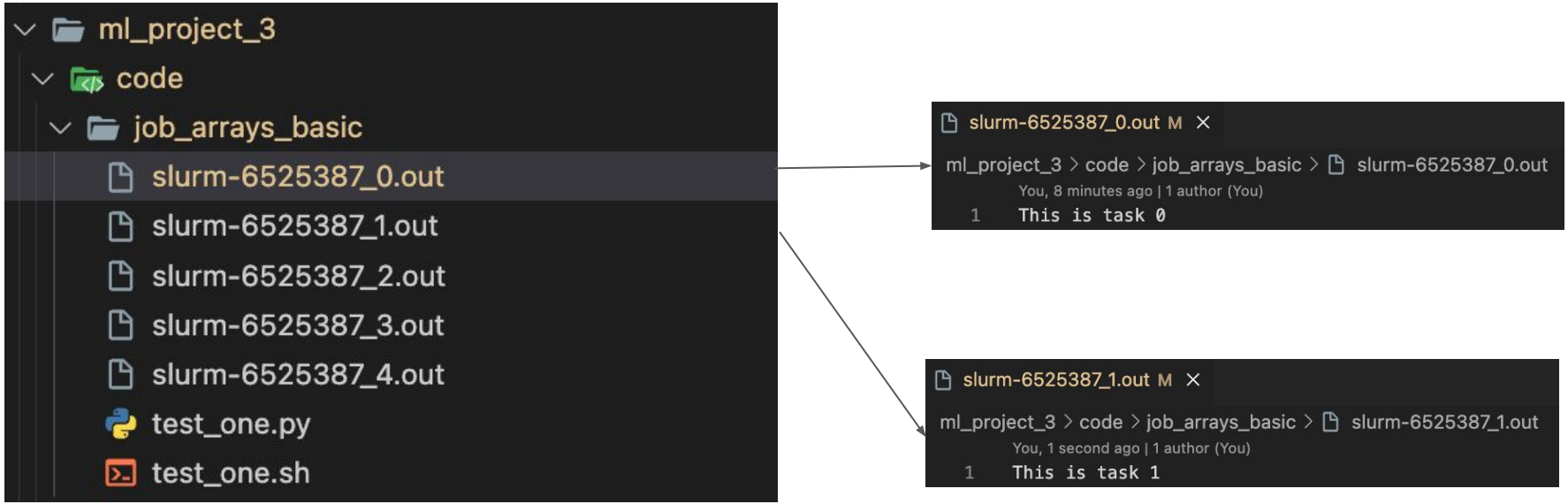
```
import sys

if len(sys.argv) == 2:
    prog_name, task_str = sys.argv
    print("This is task %s" % task_str)

```

Create a shell script file (.sh)
Set SBATCH parameters

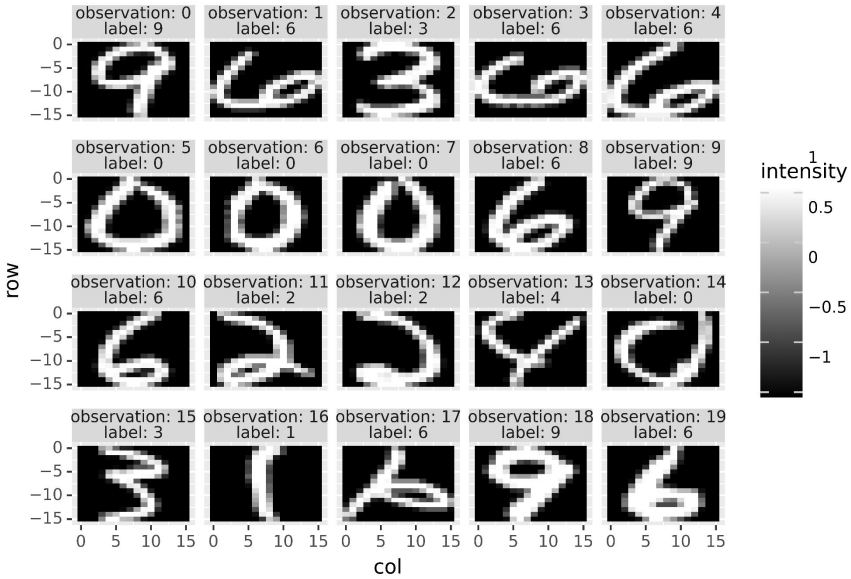
Notable Components:
#SBATCH –array=x-y
$SLURM_ARRAY_TASK_ID

Run:
sbatch test_one.sh

Slurm Job Arrays - Basic (Output)



Zip and Spam data



16x16 pixels for each zip
2007 rows x 256 columns

0.23	0	5.28	82	227	1
0	0.04	2.6	42	182	1
0.19	0	2.74	13	74	1
0	0	3.69	62	258	1
0	0	4.39	66	101	1
0.03	0.05	3.45	318	1003	1
0.02	0.02	3.48	5	5902	0
0	0	1.04	2	26	0
0	0	1	1	3	0
0	0	1.63	7	65	0
0	0.01	4.41	28	1866	0
0	0	2.5	11	35	0
0	0.14	3.57	28	150	0
0	0	1	1	1	0

Frequency of words
4601 rows x 56 columns

Labels, 1 for Spam, 0 for No Spam

Slurm Job Arrays - Advanced (1)

```
root_data_dir = "/projects/genomic-ml/da2343/ml_project_2/data"
params_df_list = []

algo_list = ["KNeighborsClassifier",
             "LinearModel",
             "Featureless"]
dataset_list = ["zip","spam"]

for dataset_name in dataset_list:
    params_dict = {
        'dataset_name': [dataset_name],
        'algorithm': algo_list,
    }

    params_df = pd.MultiIndex.from_product(
        params_dict.values(),
        names=params_dict.keys()
    ).to_frame().reset_index(drop=True)
    params_df_list.append(params_df)
params_concat_df = pd.concat(params_df_list, ignore_index=True)
n_tasks, ncol = params_concat_df.shape
```

Dataset_name	Algorithm
zip	KNeighborsClassifier
zip	LinearModel
zip	Featureless
spam	KNeighborsClassifier
spam	LinearModel
spam	Featureless

Create params.py script

Generate every possible combination of algorithm and dataset into a param.csv file

The `n_tasks` represents the number of rows in the param.csv file, which ultimately each row will be executed on a single CPU.

Slurm Job Arrays - Advanced (2)

```
date_time = datetime.now().strftime("%Y-%m-%d_%H:%M")
job_name = f"ml_project_3_{date_time}"
job_dir = "/scratch/da2343/" + job_name
results_dir = os.path.join(job_dir, "results")
os.system("mkdir -p " + results_dir)
params_concat_df.to_csv(os.path.join(job_dir, "params.csv"), index=False)

run_one_contents = f"""#!/bin/bash
#SBATCH --array=0-{n_tasks-1}
#SBATCH --time=1:00:00
#SBATCH --mem=4GB
#SBATCH --cpus-per-task=1
#SBATCH --error={job_dir}/slurm-%A_%a.out
#SBATCH --output={job_dir}/slurm-%A_%a.out
#SBATCH --job-name={job_name}
cd {job_dir}
python run_one.py $SLURM_ARRAY_TASK_ID
"""
run_one_sh = os.path.join(job_dir, "run_one.sh")
with open(run_one_sh, "w") as run_one_f:
    run_one_f.write(run_one_contents)

run_orig_py = "parallel.py"
run_one_py = os.path.join(job_dir, "run_one.py")
shutil.copyfile(run_orig_py, run_one_py)
orig_dir = os.path.dirname(run_orig_py)
orig_results = os.path.join(orig_dir, "results")
os.system("mkdir -p " + orig_results)
orig_csv = os.path.join(orig_dir, "params.csv")
params_concat_df.to_csv(orig_csv, index=False)

msg = f"""created params CSV files and job scripts, test with
python {run_orig_py}
SLURM_ARRAY_TASK_ID=0 bash {run_one_sh}"""
print(msg)
```

Saves the params.csv file in the scratch directory.

Dynamically creates an sbatch script with job array from 0 to the number of rows in the params.csv

Copies the python script as “run_one.py”,

Slurm Job Arrays - Advanced (3)

```python
params_df = pd.read_csv("params.csv")

if len(sys.argv) == 2:
    prog_name, task_str = sys.argv
    param_row = int(task_str)
else:
    print("len(sys.argv)=%d so trying first param" % len(sys.argv))
    param_row = 0

data_path = "/projects/genomic-ml/da2343/ml_project_3/data"
param_dict = dict(params_df.iloc[param_row, :])

dataset_name = param_dict["dataset_name"]
algorithm = param_dict["algorithm"]
```

Inside the main python script, read the params.csv file

Get the SLURM_ARRAY_TASK_ID from sys.argv.

Use that task_ID to get the row param dictionary.

Slurm Job Arrays - Advanced (4)

```
# Read the zip file into a pandas dataframe
zip_df = pd.read_csv(
    f"{data_path}/zip.test.gz",
    header=None,
    sep=" ")
is01 = zip_df[0].isin([0, 1])
zip01_df = zip_df.loc[is01, :]
zip01_shuffled_df = zip01_df.sample(frac=1, random_state=1).reset_index(drop=True)

# Read the spam.csv data into a pandas dataframe
spam_df = pd.read_csv(
    f"{data_path}/spam.data",
    sep=" ",
    header=None)
spam_df_shuffled = spam_df.sample(frac=1, random_state=1).reset_index(drop=True)

data_dict = {
    "zip": (zip01_shuffled_df.loc[:, 1:].to_numpy(), zip01_shuffled_df[0]),
    "spam": (spam_df_shuffled.iloc[:, :-1].to_numpy(), spam_df_shuffled.iloc[:, -1])
}

algo_dict = {
    "KNeighborsClassifier": GridSearchCV(estimator=KNeighborsClassifier(),
                           param_grid=[{'n_neighbors': [x]} for x in range(1, 21)], cv=5),
    "LinearModel": make_pipeline(StandardScaler(), LogisticRegressionCV(cv=5, max_iter=1000)),
    "Featureless": DummyClassifier(strategy="most_frequent"),
}

classifier = algo_dict[algorithm]
data_set = data_dict[dataset_name]
```

Inside the main python script, read the zip and spam dataframes.

Create data and algorithm dictionaries.

Get algorithm and data_set with keys from the row param dictionary.

Slurm Job Arrays - Advanced (5)

```
input_mat, output_vec = data_set
test_acc_df_list = []
kf = KFold(n_splits=3, shuffle=True, random_state=1)
for fold_id, indices in enumerate(kf.split(input_mat)):
    index_dict = dict(zip(["train", "test"], indices))
    set_data_dict = {}
    for set_name, index_vec in index_dict.items():
        set_data_dict[set_name] = {
            "X": input_mat[index_vec],
            "y": output_vec.iloc[index_vec]
        }
    classifier.fit(**set_data_dict["train"])
    pred_vec = classifier.predict(set_data_dict["test"]["X"])
    accuracy = accuracy_score(set_data_dict["test"]["y"], pred_vec)
    test_acc_df_list.append(pd.DataFrame({
            "test_accuracy_percent": accuracy * 100,
            "data_set": dataset_name,
            "fold_id": fold_id,
            "algorithm": algorithm
        }, index=[0]))

test_acc_df = pd.concat(test_acc_df_list)
# print(test_acc_df)
# Save dataframe as a csv to output directory
out_file = f"results/{param_row}.csv"
test_acc_df.to_csv(out_file, encoding='utf-8', index=False)
print("Done!!")
```

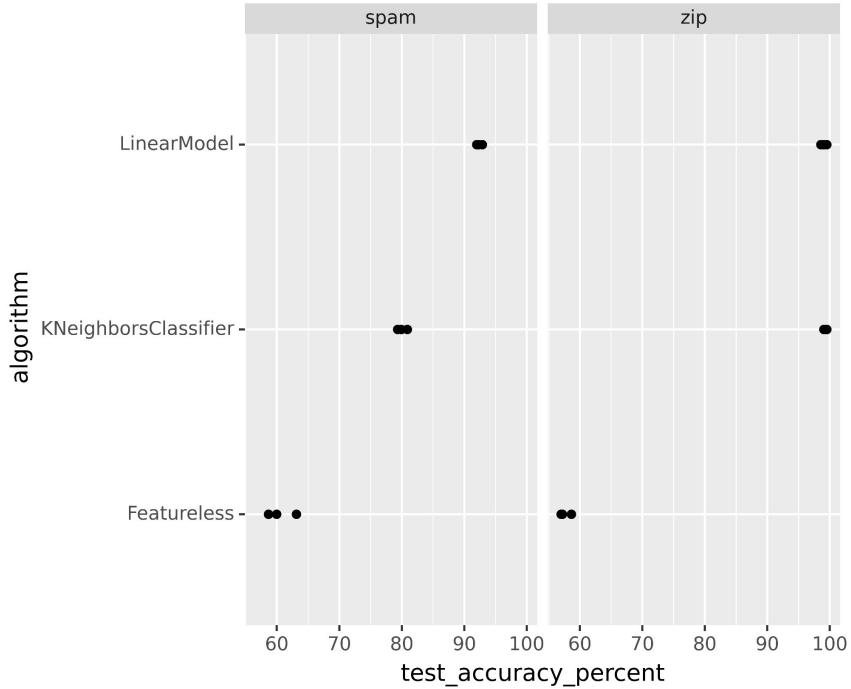
Run K-Fold cross-validation analysis.

Create test accuracy dataframe.

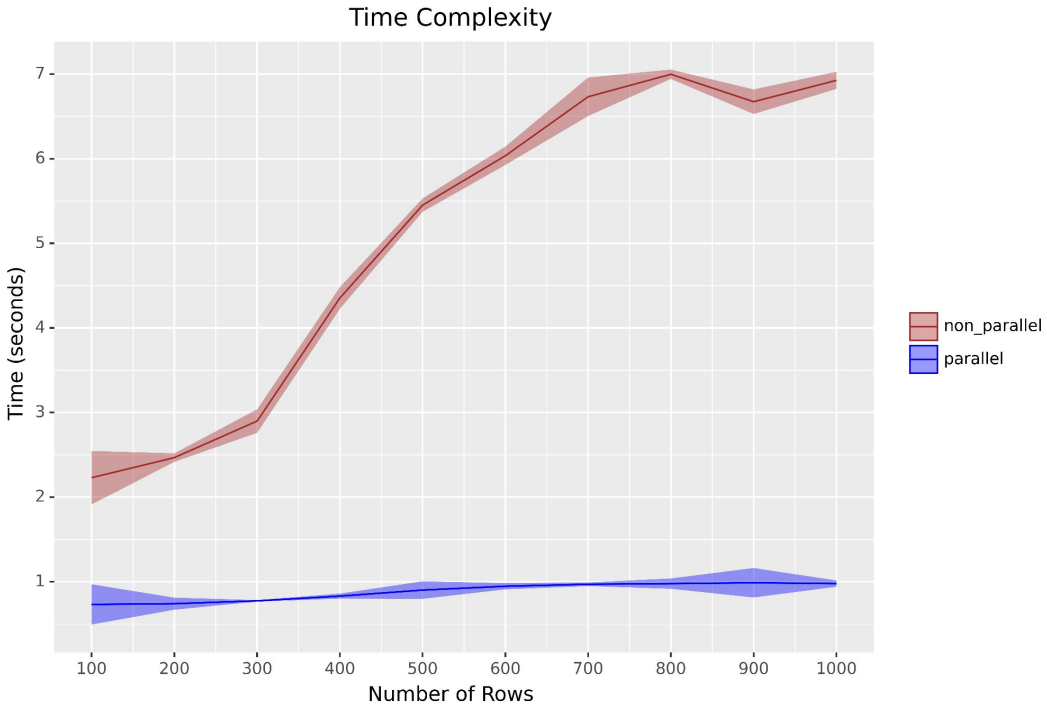
Save the test accuracy dataframe in the scratch results directory.

Slurm Job Arrays - Advanced (Output)

```python
import pandas as pd
from glob import glob
import plotnine as p9

date_time = "2023-11-17_20:57"
out_df_list = []
for out_csv in glob(f"/scratch/da2343/ml_project_3_{date_time}/results/*.csv"):
    out_df_list.append(pd.read_csv(out_csv))
test_acc_df = pd.concat(out_df_list)

root_results_dir = "/projects/genomic-ml/da2343/ml_project_3/code/job_arrays_advanced/results"
test_acc_df.to_csv(f"{root_results_dir}/{date_time}_results.csv", index=False)


# create the plot
gg = p9.ggplot() +\
    p9.geom_point(
        p9.aes(
            x="test_accuracy_percent",
            y="algorithm"
        ),
        data=test_acc_df) +\
    p9.facet_wrap("data_set")
gg.save("parallel_algo_acc.png", width=5, height=5, dpi=1000)
```



Slurm Job Arrays - Time Complexity



About 6 times speed-up

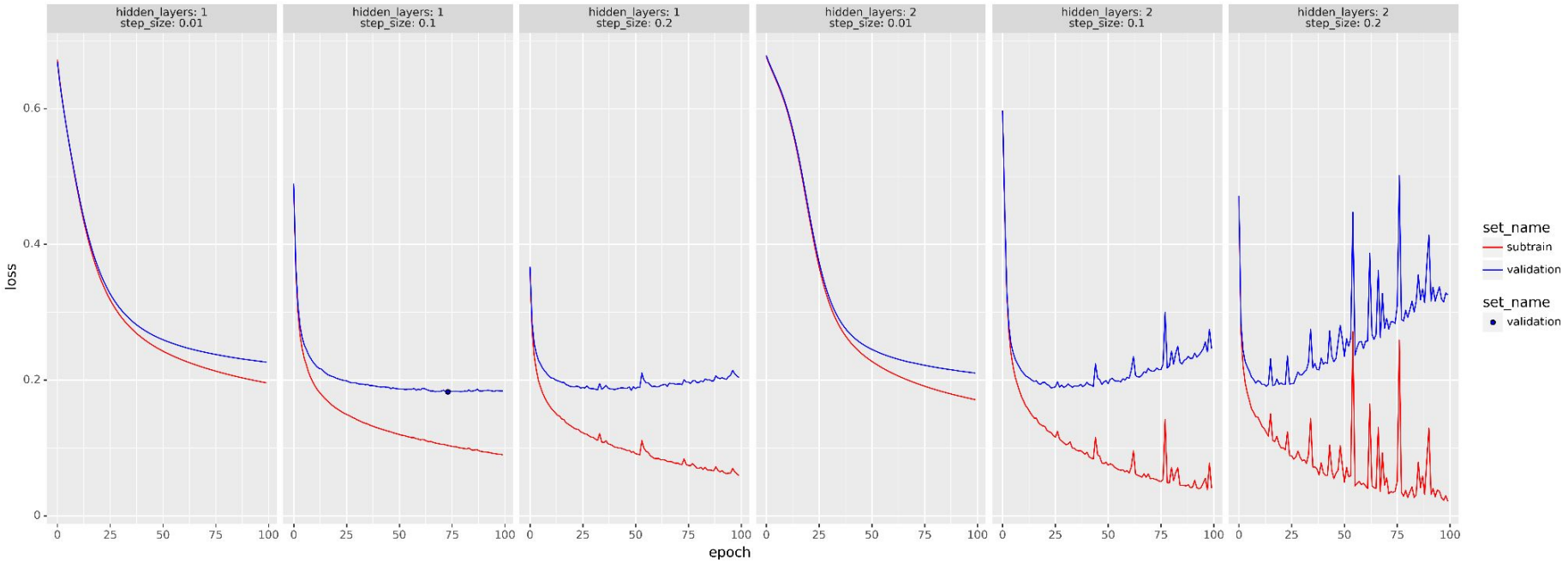
Non-parallel

- O(n_rows*n_data*n_algo) O(n_rows*k) time complexity.
- Time increases as number of rows increases

Parallel

- O(n_rows*1) time complexity
- Time increases slightly as number of rows increase

Slurm Job Arrays - Hyper-Parameter Tuning



Optimum hyper-params = hidden_layers: 1, step_size: 0.1

Thank You !!

Code and figures are available
https://github.com/EngineerDanny/ml_with_monsoon