Comparing Machine Learning Algorithms Across Data Systems

Grace Davenport, Hayden French, Silas Hayes, Ryan Lipps

Motivation

- As **data scientists**, we are interested in using big data systems for the implementation of **scikit-learn** machine learning algorithms
- We are now familiar with several **distributed computing frameworks,** but have not directly applied them to the data science pipeline
- We are interested in using **Spark**, **Dask**, and **Ray** to implement some popular machine learning algorithms—**regression**, **classification**, and **clustering**—and compare compute times between each system

Data Preprocessing

- Dataset: **20 million flight searches** between June, 2022 and August, 2022 and is approximately **2 GBs**

- Variables: **(1)** flight search on a specific date, **(2)** returning base fare, **(3)** distance, **(4)** time traveled, and **(5)** number of seats remaining

- Prior to uploading our data to our cluster, we **normalized each variable** via **z-score scaling** and performed a random **80-20** train/test split

- We are more interested in the **overall compute times** than **model performance**, but still aim to mirror the traditional machine learning pipeline as closely as possible

Experimental Design

- We implemented each algorithm on **Dask**, **Spark**, and **Ray** hosted on **Amazon's EC2** instances
- For all algorithms, we used all **quantitative** variables as predictors. For KNN and Random Forest, our response variable was the **base price** of the flight. For K-Means, we performed **unsupervised clustering**
- We ran each algorithm on **differently-sized subsets** of our data and recorded the execution time for each

Sklearn + Joblib

- Parallelize sklearn with this **one neat trick!!**

```
with joblib.parallel_backend('dask'):
    rf_model = RandomForestRegressor(max_depth=2)
    rf_model.fit(X_reg_train_subset, y_reg_train_subset)
    y_pred = rf_model.predict(X_reg_test_subset)
    metric = mean_squared_error(y_reg_test_subset, y_pred)
```

dask

Dask | Distributed | Dask ML | Examples | Ecosystem | Community

Pipelines and Composite Estimators

Generalized Linear Models

Naive Bayes

Parallel Meta-estimators

Incremental Learning

Clustering

API Reference

INTEGRATION

To use the Dask backend to Joblib you have to create a Client, and wrap your code with `joblib.parallel_backend('dask')`.

```
from dask.distributed import Client
import joblib

client = Client(processes=False)             # create local cluster
# client = Client("scheduler-address:8786")  # or connect to remote cluster

with joblib.parallel_backend('dask'):
    # Your scikit-learn code
```

Quickstart

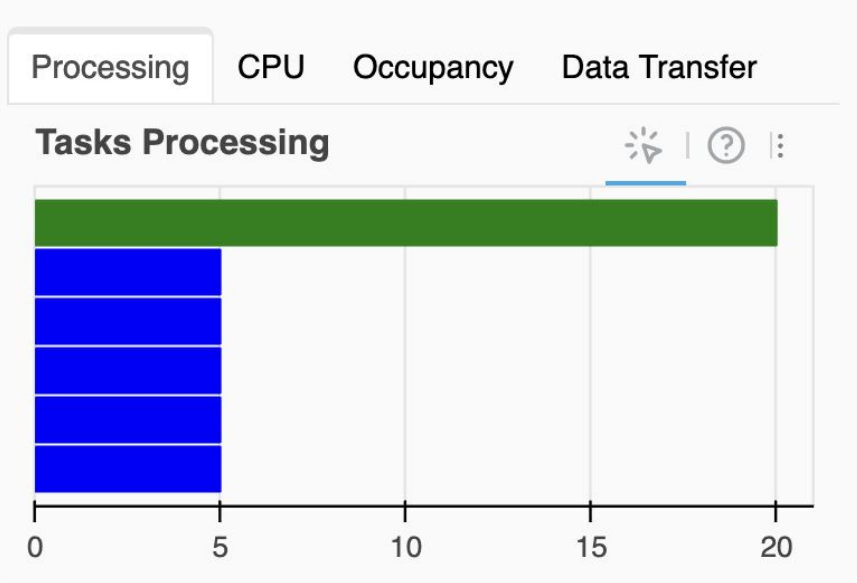
To get started, first install Ray, then use `from ray.util.joblib import register_ray` and run `register_ray()`. This will register Ray as a joblib backend for scikit-learn to use. Then run your original scikit-learn code inside `with joblib.parallel_backend('ray')`. This will start a local Ray cluster. See the Run on a Cluster section below for instructions to run on a multi-node Ray cluster instead.

```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVC
digits = load_digits()
param_space = {
    'C': np.logspace(-6, 6, 30),
    'gamma': np.logspace(-8, 8, 30),
    'tol': np.logspace(-4, -1, 30),
    'class_weight': [None, 'balanced'],
}
model = SVC(kernel='rbf')
search = RandomizedSearchCV(model, param_space, cv=5, n_iter=300, verbose=10)

import joblib
from ray.util.joblib import register_ray
register_ray()
with joblib.parallel_backend('ray'):
    search.fit(digits.data, digits.target)
```

It’s not that simple.

Sklearn + Joblib Actually Parallel?



Joblib

Navigation

User manual

Why joblib: project goals
Installing joblib
On demand recomputing:

joblib.parallel_backend

`class joblib.parallel_backend(backend, n_jobs=-1, inner_max_num_threads=None, **backend_params)`

Change the default backend used by Parallel inside a with block.

> **Warning:**
>
> It is advised to use the `parallel_config` context manager instead, which allows more fine-grained control over the backend configuration.

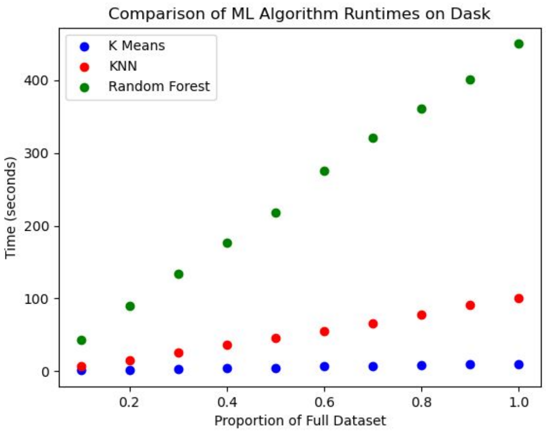
If `backend` is a string it must match a previously registered implementation using the `register_parallel_backend()` function.

You can also use the Dask joblib backend to distribute work across machines. This works well with scikit-learn estimators with the `n_jobs` parameter, for example:

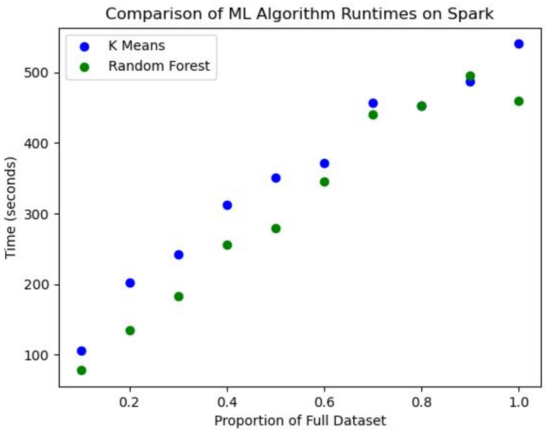
```
>>> import joblib
>>> from sklearn.model_selection import GridSearchCV
>>> from dask.distributed import Client, LocalCluster
```

```
>>> # create a local Dask cluster
>>> cluster = LocalCluster()
>>> client = Client(cluster)
>>> grid_search = GridSearchCV(estimator, param_grid, n_jobs=-1)
...
>>> with joblib.parallel_backend("dask", scatter=[X, y]):
...     grid_search.fit(X, y)
```

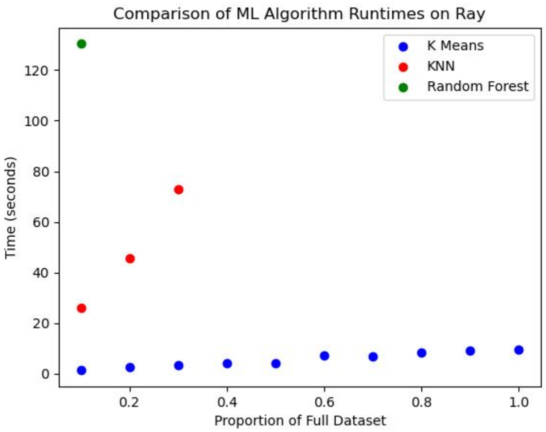
Sklearn Results



Dask



Spark



Ray

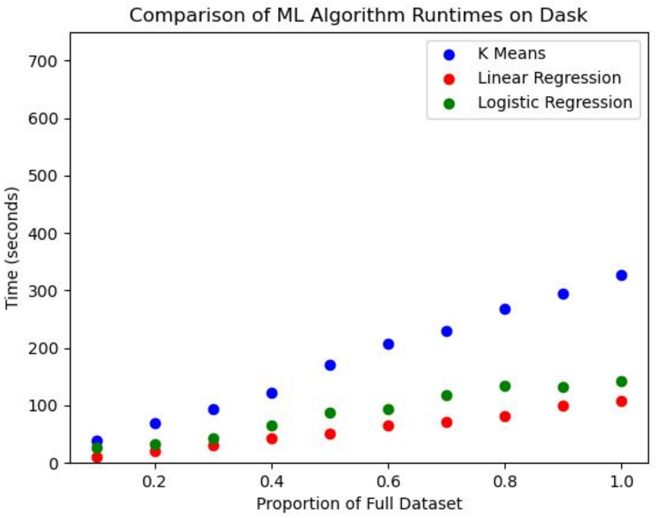
Modified Experimental Design

- We modified our approach to use Dask and Spark’s distributed computing frameworks **own machine learning backend** instead of **sklearn**
- We selected algorithms which were supported by **both libraries** and representative of **different machine learning methods** (clustering vs. regression, e.g.)
- Again, we ran each algorithm on **differently-sized** subsets of the data to compare runtimes
- **Ray** does not have its own ML implementation for these algorithms though it is still useful for **RL** and **parallelized tuning**

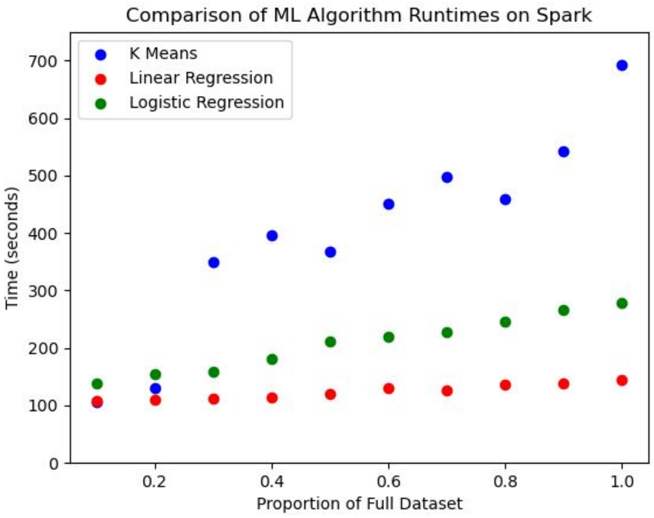
Results

	Dask (s)	Spark (s)
Linear Regression	108.04	144.91
Logistic Regression	142.80	279.03
K-Means	326.70	691.83

Discussion



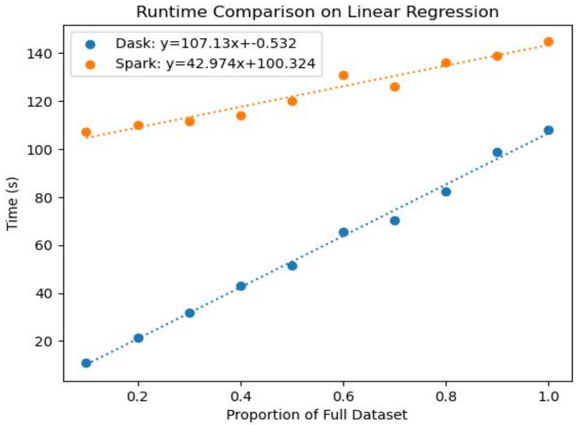
Dask



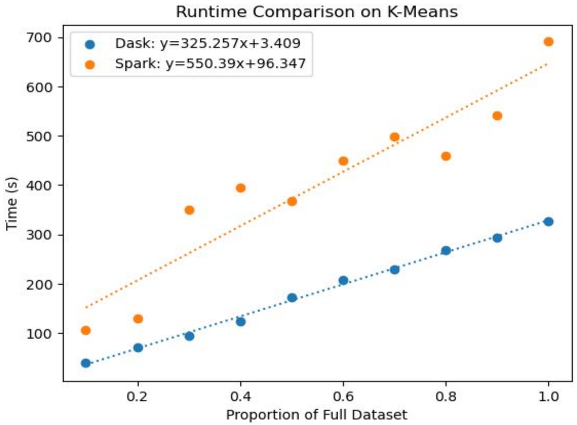
Spark

Discussion

Linear Regression



K-Means



Logistic Regression

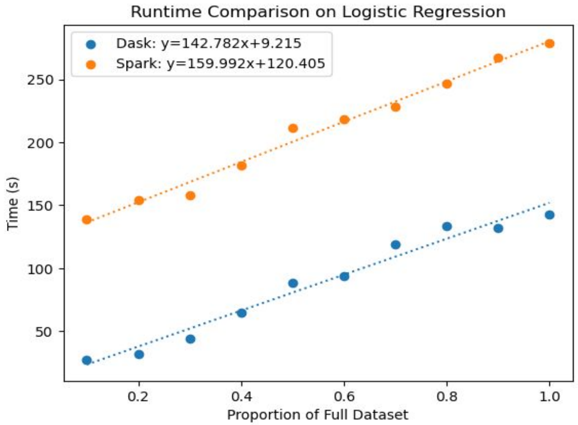


Table 2: R^2 Values for Regression Equations

Model	Dask R^2	Spark R^2
Linear Regression	0.996	0.956
Logistic Regression	0.971	0.987
K-Means	0.997	0.883

Future Work

Custom Distribution

Assess these systems using custom sklearn distributing

Vary VM Numbers

Each system different number of VMs

More Precise Metrics

Calculate the mean and standard deviation

Thank you!