

Big Data Analytics - SAT5165
Report for the First Large Project
By: Andrea Wroblewski and Olivia Gette

[GitHub Link](#)

Title: Predicting ICU length of stay using Spark Based Regression

The primary goal of this project is to use Apache Spark for distributed data processing and regression modeling to predict ICU(Intensive Care Unit) length of stay (LOS). Accurate LOS prediction is important for optimizing hospital resource allocation and improving patient care strategies. We will be working with a subset of a publicly available ICU dataset obtained from Kaggle titled "HOSP & ICU Datasets (100,000 med-data from 2001–2019)" dataset. This is a large-scale clinical database that has detailed patient demographics, admission records, and ICU metrics.

To manage the scale and complexity of the data, we will use Apache Spark across four virtual machines (VMs), one Master Node and three Worker Nodes. This lets us use parallel processing and efficient computation. This setup will allow us to explore the benefits of distributed computing when handling high-dimensional clinical data.

We expect that using four VMs will significantly reduce preprocessing and training time compared to a single-node setup, demonstrating the scalability and efficiency of Spark in healthcare analytics. This project will showcase the power of distributed machine learning for clinical decision support and provide insights into factors influencing ICU stay duration.

Code Explanation and Method

Each team member used the same preprocessing method but different modeling pipelines to explore different statistical methods and transformations.

Python Code

Load Data and Preprocess:

The first part of the ICU LOS pipeline Python program loads a subset of records from the Kaggle "HOSP & ICU Datasets (100,000 med-data from 2001-2019)". The subset was created by extracting potential CSVs needed for modeling, since the original file was too large to process in the virtual machine. The file used was ICUSTAYS.csv, which contains patient identifiers, ICU admission and discharge times, and calculated length of stay. It was the only CSV used. The file was uploaded to the Hadoop Distributed File System (HDFS) at `hdfs://master:9000/user/sat3812/mimic/raw/ICUSTAYS.csv` and read into a Spark DataFrame using PySpark's `read.csv()` function.

For preprocessing, LOS was cleaned by removing negative or null values.. The preprocessed LOS column was cast to a double data type to be compatible with SparkMLlib regression

models. A log-transformed version of LOS (`los_log`) was created as the regression label to reduce skewness in the target distribution, and missing numeric values were handled using median imputation. Categorical features were processed with one-hot encoding.

All preprocessed data were combined into a single vector using `Vector Assembler`. `MinMaxScaler` was applied to scale all features to the $[0, 1]$ range for model stability.

Linear Regression with StandardScaler(Andrea):

The first model implemented on the ICU LOS pipeline is a Linear Regression (LR) model. This acts as a baseline model for regression tasks. Using the scaled feature single vector (features), the model was initialized to predict the original, untransformed LOS. The LR model was configured to run with a maximum of 10 iterations (`maxIter = 10`). The preprocessed data was then split into 80/20 (`training_ratio = 0.8`) using a fixed random seed (`seed = 42`) for reproducibility. The model was then trained using the training data. Next, the trained LR model was used to generate the predictions on the testing set. Using `RegressionEvaluator`, the performance was assessed by calculating the root mean square error (RMSE) and R-squared (R^2). The model achieved an RMSE of 9.778 days and an R^2 of 0.017, which indicates it does not have much predictive power.

Random Forest Regression with MinMaxScaler (Olivia):

The second model implemented in the pipeline was a Random Forest Regressor (RF). This ensemble method was used to determine whether a more complex, non-linear statistical analysis could improve prediction accuracy relative to the baseline model.

This model used the same preprocessed data as the LR model. Using the scaled feature single vector (features), the model was initialized to predict the original, untransformed LOS. RF was configured to run with 50 decision trees (`numTrees = 50`).

RF used the same 80/20 test split (`training_ratio = 0.8`). The model was trained (fit) with the training data. Next, the trained model generated predictions. `RegressionEvaluator` was used to evaluate the model using RMSE and R^2 . The model achieved an RMSE of 9.567 days and an R^2 of 0.059, which is a minor improvement over the LR baseline.

Compare Models:

Both models showed limited predictive power (R^2 max of 0.059). Currently, the RF model predicts the original, heavily skewed LOS data. To improve accuracy, the model could be retrained using the log-transformed LOS (los_log). This approach would normalize LOS distribution, allowing the model to learn the underlying relationship between the features and the target variable more effectively. In addition, hyperparameter tuning (such as CrossValidator) could be used to optimize performance.

Model	Root Mean Square Error	R-Squared
Linear Regression	9.778 Days	0.017
Random Forest Regressor	9.567 Days	0.059

VM Configuration

To meet the project's distributed computing requirements, we configure four VMs: one master node and three worker nodes. We accessed the Michigan Tech vSphere client using the BIG-IP Edge VPN and used the VMware Remote Console (VMRC) to launch and manage the VMs.

The IP addresses assigned were:

- Andrea:
 - Master: 192.168.13.200
 - Worker1: 192.168.13.201
- Olivia:
 - Worker2: 192.168.13.122
 - Worker3: 192.168.13.123

Network & Hostname Configuration:

We assigned hostnames to each IP address by editing /etc/hosts on the master VM with `sudo vi /etc/hosts`. This allows us to use easier hostnames rather than IP addresses.

We verified connectivity between nodes using ping and ensured that all necessary ports were open for Spark communication.

```
--- master ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.031/0.031/0.031/0.000 ms
PING worker1 (192.168.13.201) 56(84) bytes of data.
64 bytes from worker1 (192.168.13.201): icmp_seq=1 ttl=64 time=0.428 ms

--- worker1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.428/0.428/0.428/0.000 ms
PING worker2 (192.168.13.122) 56(84) bytes of data.
64 bytes from worker2 (192.168.13.122): icmp_seq=1 ttl=64 time=0.431 ms

--- worker2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.431/0.431/0.431/0.000 ms
PING worker3 (192.168.13.123) 56(84) bytes of data.
64 bytes from worker3 (192.168.13.123): icmp_seq=1 ttl=64 time=0.464 ms

--- worker3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.464/0.464/0.464/0.000 ms
```

Apache Spark Configuration

To enable distributed processing, we configured Apache Spark in standalone cluster mode across our four VMs. One master node and three worker nodes.

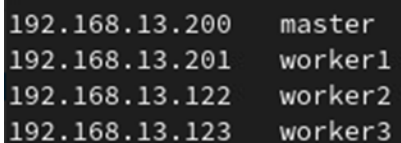
We set up the SSH key by generating SSH key pairs on the master VM and shared them with the corresponding worker VMs to enable passwordless SSH communication. For this, we used these commands:

```
ssh-keygen -t rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
scp ~/.ssh/id_rsa.pub worker1:~/.ssh/authorized_keys
scp ~/.ssh/id_rsa.pub worker2:~/.ssh/authorized_keys
scp ~/.ssh/id_rsa.pub worker3:~/.ssh/authorized_keys
```

Spark was already installed on the VMs. We packaged the entire Spark directory (/opt/spark) and transferred it to the worker nodes using scp:

```
cd /opt
tar czf spark.tar.gz spark
scp spark.tar.gz worker1:/opt
scp spark.tar.gz worker2:/opt
scp spark.tar.gz worker3:/opt
```

After transferring spark to each worker we verified that /opt/spark directory was the same on all VMS:



```
192.168.13.200    master
192.168.13.201    worker1
192.168.13.122    worker2
192.168.13.123    worker3
```

Next we created and distributed the Spark files, updating /opt/spark/conf/spark-env.sh with:

```
export JAVA_HOME=/opt/jdk
export SPARK_MASTER_HOST=master
export SPARK_WORKER_CORES=8
export SPARK_WORKER_MEMORY=4g
export SPARK_LOCAL_DIRS=/tmp/spark
```

On master nodes we listed all the worker host names using /opt/spark/conf/workers:

```
worker1
worker2
worker3
```

We copied these files to every node using scp and changed the file permissions so Spark could save and update its temporary data:

```
sudo chown -R $(whoami):$(id -gn) /opt/spark /tmp/spark
```

Next we started spark and confirmed the cluster was running:

```
/opt/spark/sbin/start-all.sh
```

```
for n in master worker1 worker2 worker3; do echo "== $n =="; ssh $n jps; done
```

```
== worker1 ==
20791 Jps
19519 Worker
== worker2 ==
24355 Jps
23059 Worker
== worker3 ==
20706 Jps
19800 Worker
```

Performance Comparison:

In this project, we compared the performance of running the `icu_los_pipeline.py` on 1, 2, 3, and 4 virtual machines using Apache Spark in distributed mode.

Number of VMs	Cores Used (8 per VM)	Duration (min)	Duration (seconds)
1	8	1.7	102.0
2	16	1.4	84.0
3	24	1.4	84.0
4	32	1.5	90

Performance Summary:

Runtime decreased from one VM to multiple VMs. The best performance was achieved with two to three VMs, which reduced runtime from 1.7 minutes to 1.4 minutes, roughly a 20% improvement in processing time. The four VM was more efficient than the single VM, but had slightly higher runtime than two and three VMs. This could be due to communication overhead across an additional node.

This demonstrates that Spark processing improves computation efficiency for larger datasets. However, for smaller data sets, network overhead can offset parallelization gains.

Spark Master at spark://m x

master:8080

http://hadoop1:9870 http://hadoop1:9864 spark:4040 Spark Master 8080

Workers (4)

Worker Id	Address	State	Cores	Memory	Resources
worker-20251026202758-192.168.13.200-45047	192.168.13.200:45047	ALIVE	8 (0 Used)	4.0 GiB (0.0 B Used)	
worker-20251026210815-192.168.13.122-36057	192.168.13.122:36057	ALIVE	8 (0 Used)	4.0 GiB (0.0 B Used)	
worker-20251026211124-192.168.13.201-37955	192.168.13.201:37955	ALIVE	8 (0 Used)	4.0 GiB (0.0 B Used)	
worker-20251026211338-192.168.13.123-45629	192.168.13.123:45629	ALIVE	8 (0 Used)	4.0 GiB (0.0 B Used)	

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

Completed Applications (5)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20251026211557-0004	ICU_los_pipeline	32	1024.0 MiB		2025/10/26 21:15:57	sat3812	FINISHED	1.6 min
app-20251026211349-0003	ICU_los_pipeline	32	1024.0 MiB		2025/10/26 21:13:49	sat3812	FINISHED	1.5 min
app-20251026211136-0002	ICU_los_pipeline	24	1024.0 MiB		2025/10/26 21:11:36	sat3812	FINISHED	1.4 min
app-20251026210835-0001	ICU_los_pipeline	16	1024.0 MiB		2025/10/26 21:08:35	sat3812	FINISHED	1.4 min
app-20251026210522-0000	ICU_los_pipeline	8	1024.0 MiB		2025/10/26 21:05:22	sat3812	FINISHED	1.7 min

Discussion and Conclusion

Both Linear Regression and Random Forest models were successfully implemented using PySpark MLlib on a distributed Spark cluster. While predictive accuracy was low ($R^2 \approx 0.06$), the project achieved its goal of demonstrating how distributed computing improves processing time and scalability for clinical data analysis.

The 4-VM cluster reduced runtime from about 1.7 minutes (on one VM) to 1.5 minutes, showing clear parallel processing benefits. Future work could explore hyperparameter tuning and log-transformed LOS to enhance prediction accuracy and extend this pipeline to larger datasets such as MIMIC-IV.

Dataset Used:

Kaggle. (2024). *HOSP & ICU Datasets (100,000 med-data from 2001–2019)* [Dataset].

Retrieved from <https://www.kaggle.com/datasets/luciadam/icu-datasets>