Improving the Machine Learning Pipeline at Duke

Dima Fayyad Sean Holt David Rein
Project Leads: AJ Overton Ricardo Henao

Background

The large amounts of data that hospitals collect can make health data science projects computationally expensive. These projects at Duke currently do not take advantage of recent developments in distributed computing systems. Apache Spark is an open-source cluster-computing framework which supports implicit data parallelism, and provides a user-friendly interface for large-scale data processing.

Goals

We compared conventional (Oracle Exadata) and distributed (Apache Spark) systems in an effort to operationalize the application of distributed computing methodologies in the analysis of electronic medical records (EMR) at Duke. This involved developing project-agnostic tools for natural language processing (NLP) tasks. We applied these systems to an NLP project on clinical narratives and were able to predict growth failure in premature babies, a condition which can cause severe developmental issues later in life.

Why Spark?

Although data scientists are familiar with Apache Hadoop, we utilize Spark as it optimizes Hadoop. Spark improves memory allocation, is implementable in more environments, and generalizes well with SQL and Machine Learning processes. The improved memory allocation aids in this open-source software's speed and high performance, which motivated our project to compare this new software to the software that Duke Forge uses currently.

Tools

Functions developed for Health Data Science at Duke

1. Load Table - Pulls data from Oracle Exadata and stores it in parquet format (optimized for Spark)
2. Word Count - Counts the number of instances of each unique word in a document
3. Summarize Vitals - Summarizes vital signs (e.g. heart rate, blood pressure, etc.) for each patient
4. Regex Search - Searches documents for any regex expression
5. One Hot Encoding - Creates a one hot encoding for words in a document
6. Sum Vectors - Converts documents to word embedding representations and aggregates them accordingly. (See Aggregate Vectors)

To compare the traditional method vs. Spark, we developed and benchmarked these functions in both systems. These benchmarks allow us to make informed decisions when making pipeline recommendations.

Traditional vs. Spark

Distributed Computing: Apache Spark
For large tasks, Spark consistently outperforms conventional methods because it distributes data and tasks efficiently across multiple machines.

Linear Computing: Duke VM
A traditional Duke Virtual Machine (VM) is faster than Spark when analyzing small datasets because Spark has a computational overhead necessary to partition the data.

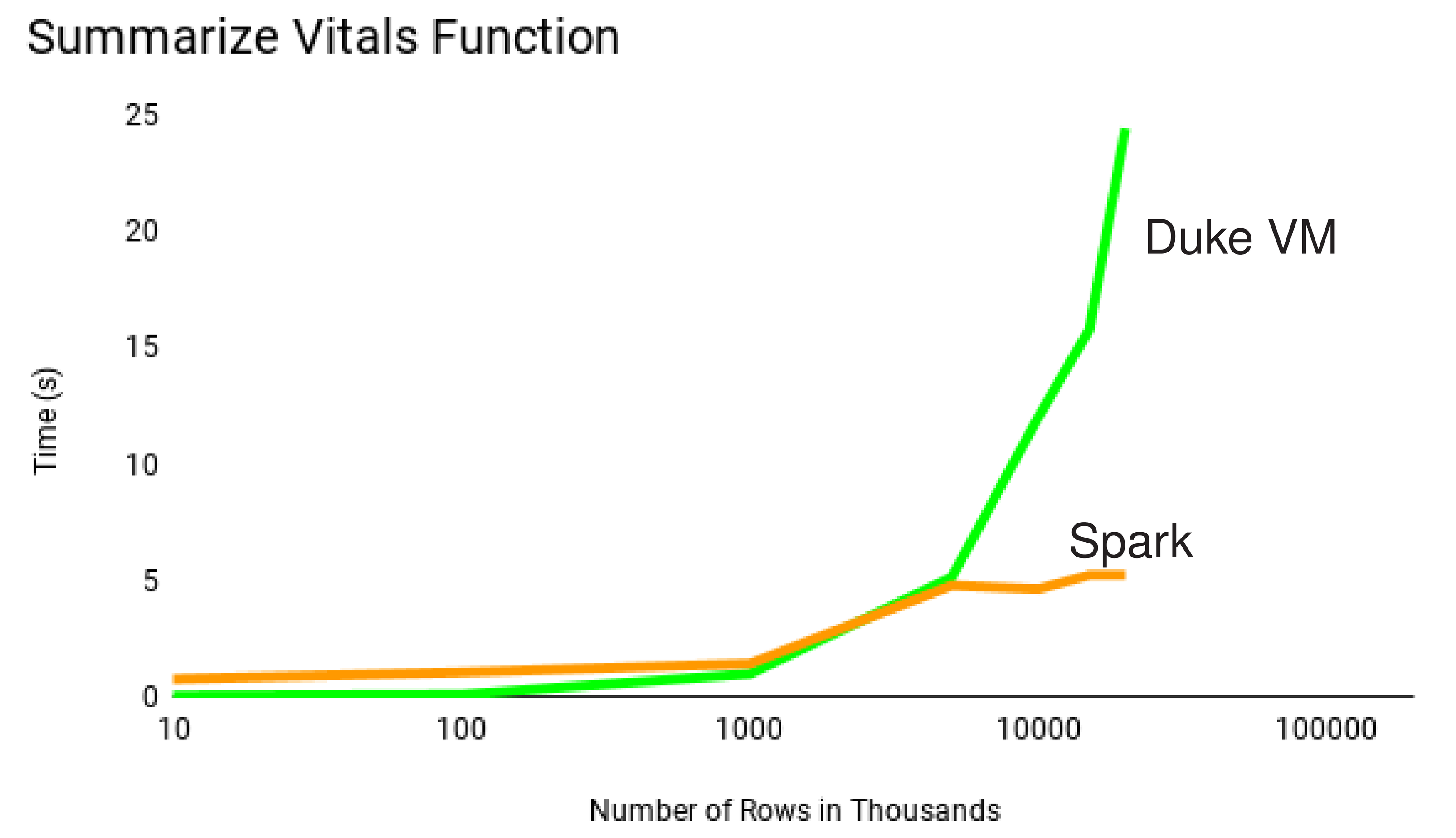

Figure 1: For this function, the run-times for the two computing methods diverge around 10,000,000 observations. This difference will increase as more observations are used.

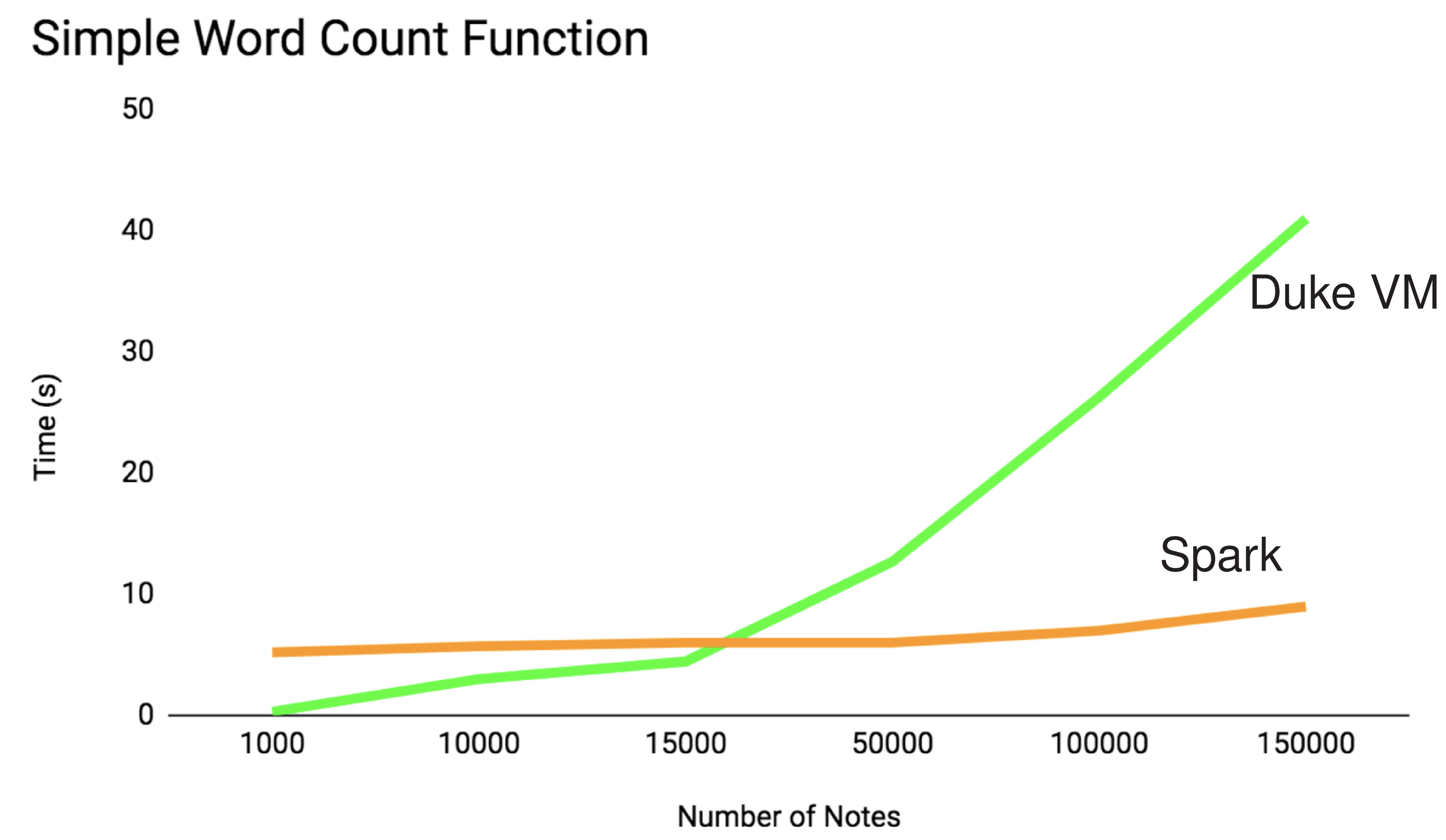

Figure 2: For the Word Count Function, the run-times for the two computing methods diverge around 15,000 labels. Word count tasks require much less data to significantly affect run-time performance.

Pipeline and Preprocessing

Pipeline

After benchmarking different processes, we constructed the pipeline shown below.

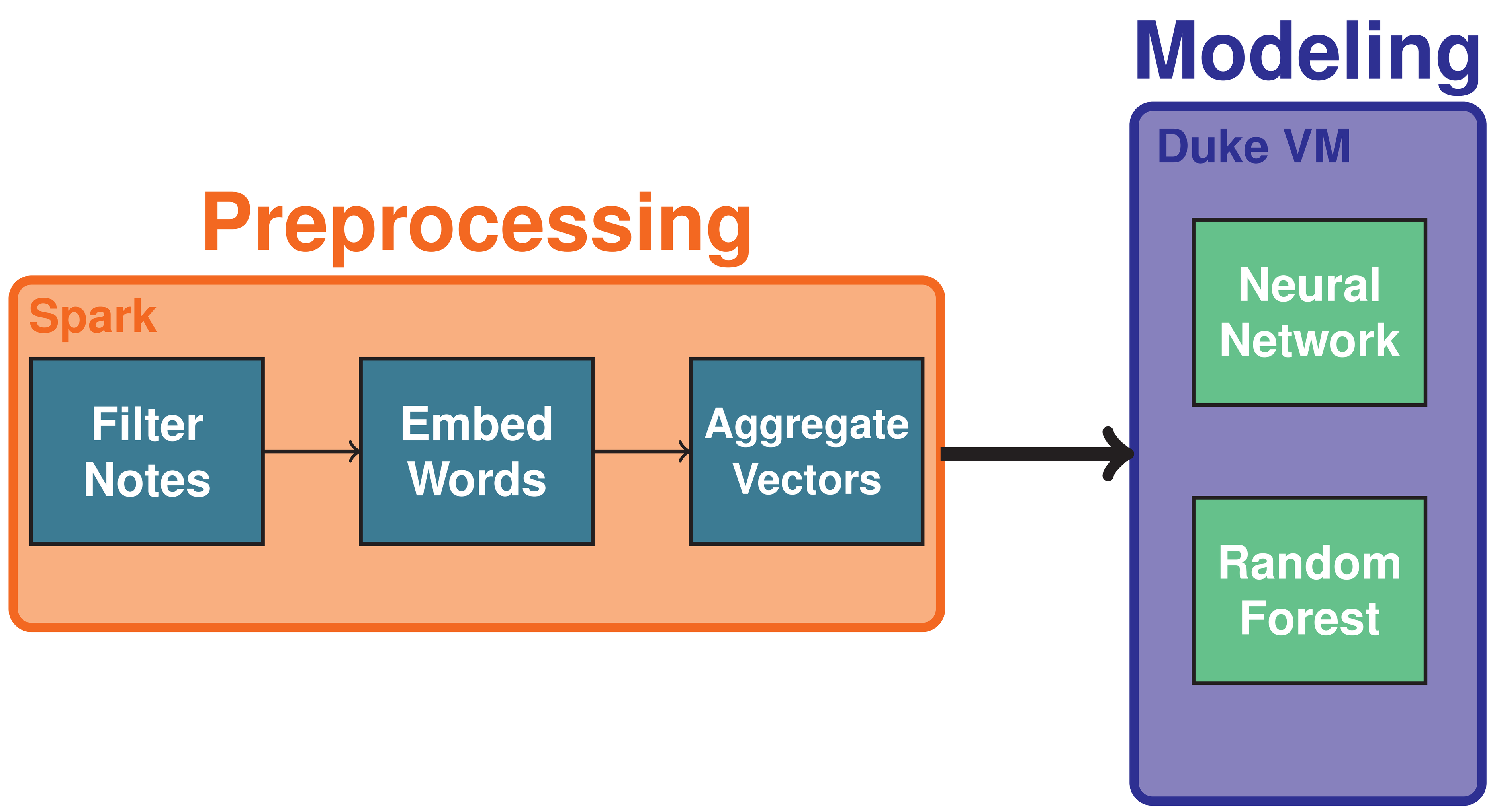

Figure 3: We preprocess the data in Spark, and train our models on the Duke VM. This is because after aggregation we have a small training set, and model training for this small dataset is faster on a Duke VM.

Filter Notes: The Population of Interest

Our population of interest is infants who: *

(a) Were born before 34 weeks gestational age.
(b) Have a weight measurement between 34 and 38 weeks gestational age.
(c) Have a doctor note before 34 weeks.

These criteria yielded **a population of 1,042 infants** from a total population of around 17,000 infants admitted to the NICU.

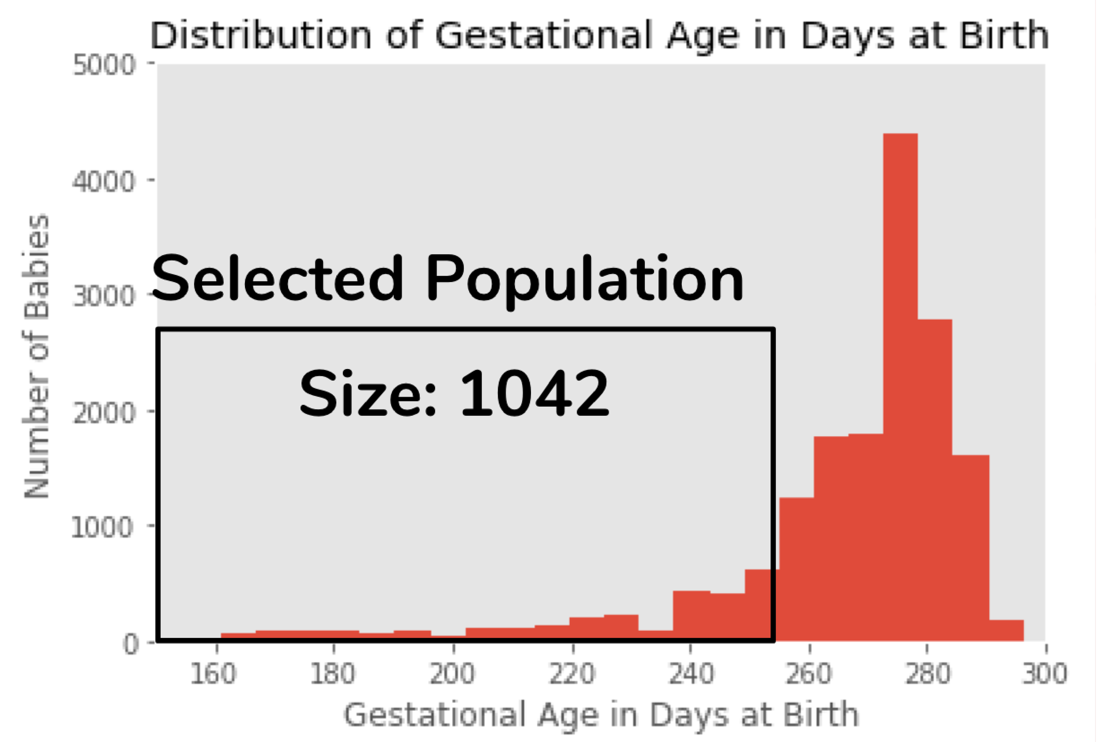

Figure 4

*These criteria were suggested to us by Dr. Noelle Younge.

Embed Words

Word embeddings are a technique for quantifying semantic meanings of words. We trained our own word embeddings on ~1,000,000 notes using Word2Vec, and tried pre-trained GloVe embeddings. We explored different text munging processes, including stemming and the removal of stop words. Generally these processes did not make a meaningful difference in the model's ability to classify patients. We settled on the pre-trained GloVe embeddings, as this eliminated some concern of bias.

Examples of how Word Embeddings capture semantic meaning

Word embeddings are used in natural language processing to quantify the semantic meanings of specific words. They assign an n-dimensional vector to each unique word. Interesting examples include:

1. Closest vectors to "sore throat": "scratchy", "cough" (embedding from EMR data).
2. King - Man + Woman = Queen (embedding from GloVe).

Aggregate Vectors

We begin by concatenating each patient's set of notes to get a "patient note". Then, after mapping each word to its embedding (f_0), we aggregate the embedded vectors for each patient to get a "patient embedding".

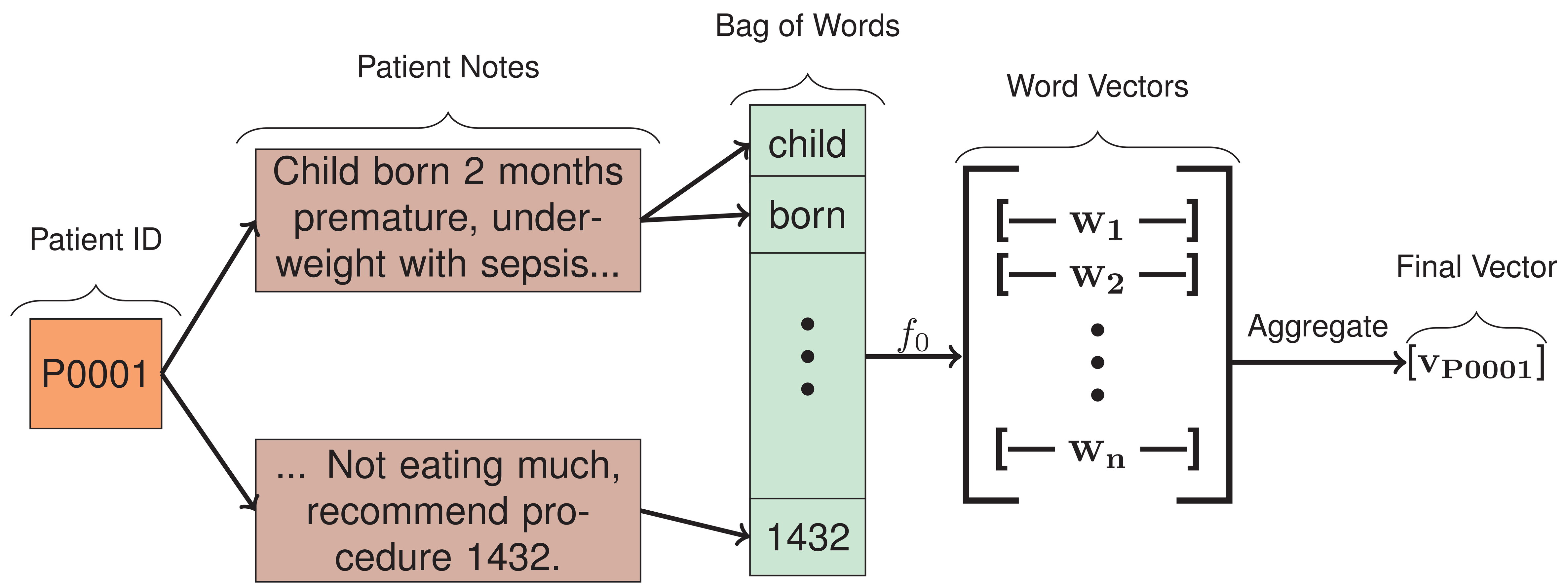

Figure 5

We tested and evaluated three different aggregation functions:

1. Averaging: taking the average value from each embedding dimension
2. Max-pooling: taking the maximum value of each embedding dimension
3. Hierarchical-pooling: averaging local windows of word vectors across the "patient note" and max-pooling the averages. Ideally this preserves some spatial information..

For classification, averaging performed the best. After this process, each patient has a single feature vector "patient embedding". These are used as the inputs for our models.

Results and Application

Classification Results: Using Only Notes

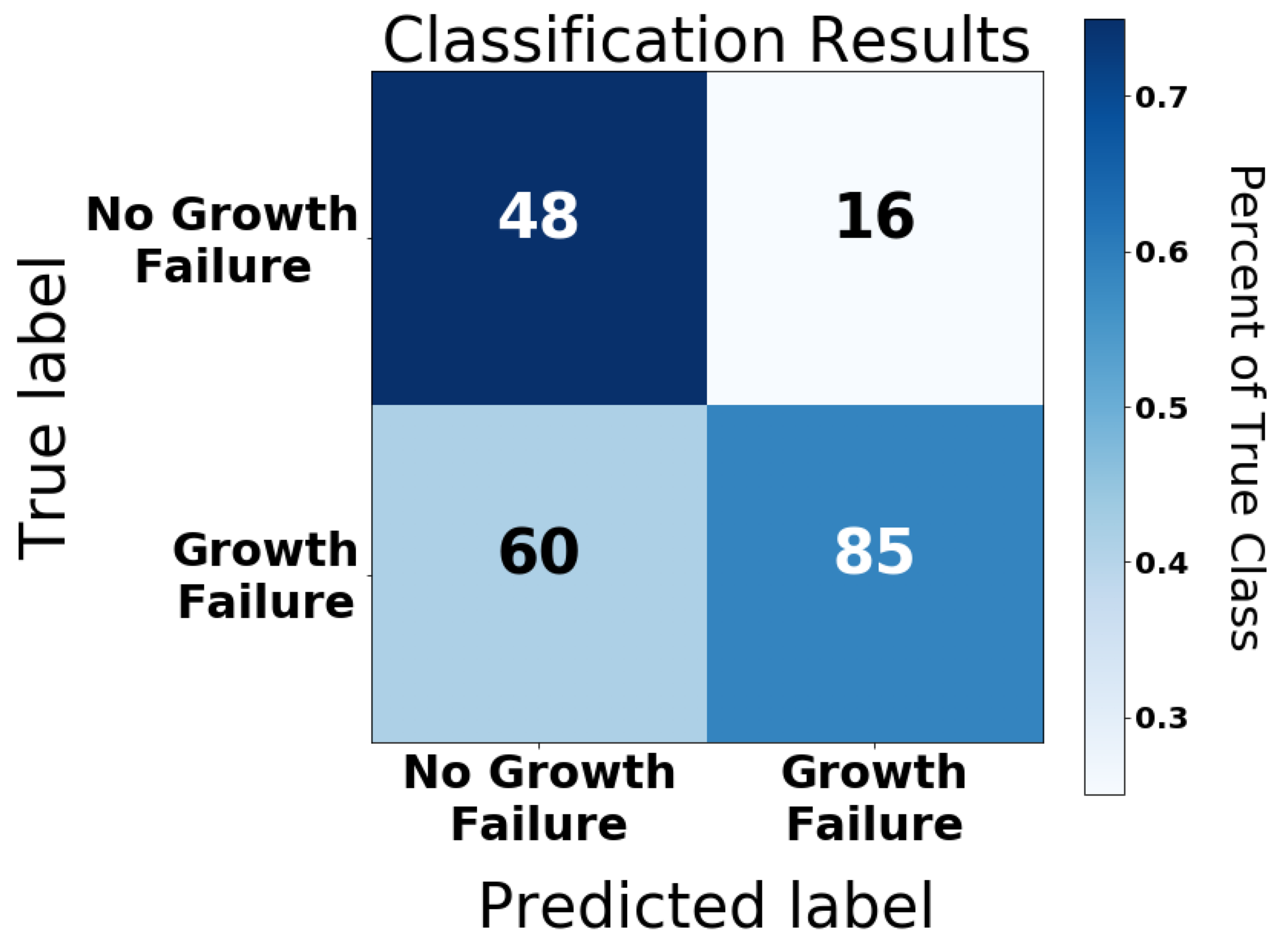

Figure 6: This confusion matrix was created by a Multi-Layer Perceptron (MLP) with an operating point of .65. This threshold is chosen with equal value for sensitivity and specificity, and can be changed by clinicians based on the costs associated with each type of misclassification. Specificity: 0.75; Sensitivity: 0.59; PPV: 0.84; NPV: 0.44

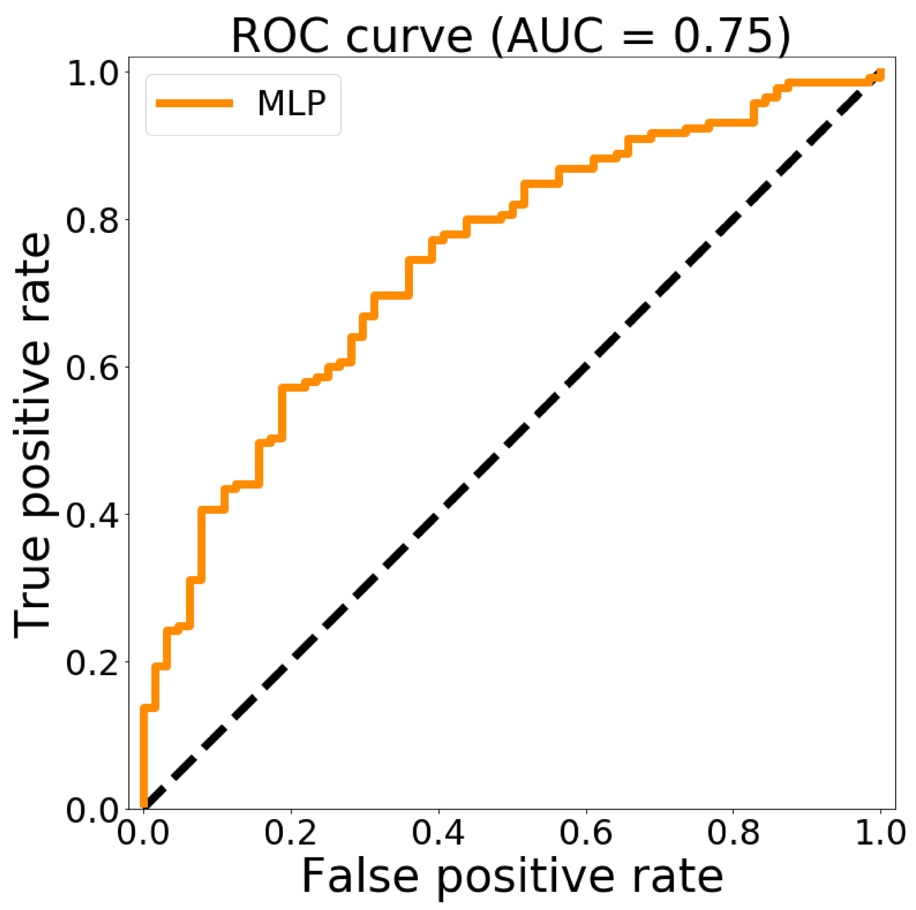

Figure 7: The ROC curve for our MLP. The AUC: 0.75. Models previously implemented by Duke Hospital had an AUC of ∼0.75.

Exploring Other Features

Although the initial goal of this project was to improve the pipeline and to pursue a proof of concept that established notes as a feature with predictive potential, we we briefly explored other features and their predictive potentials.

Additional Features

1. Notes, birth weight and difference between weights at birth and 34 weeks: 0.92 AUC
2. Notes and birth weight: 0.84 AUC
3. Notes and weight at 34 weeks: 0.94 AUC

The improvements seen to the MLP with the addition of new features are an example of the data exploration and analyses that our proposed pipeline makes possible by using Apache Spark.

In Depth Analysis

To better understand the performance of our model, we examined the patients misclassified by our model. We plot the true weight at 36 weeks against our model's predicted probability of growth failure to get a sense of "how wrong" our model is, and how changing our operating point and growth failure thresholds affects our predictions.

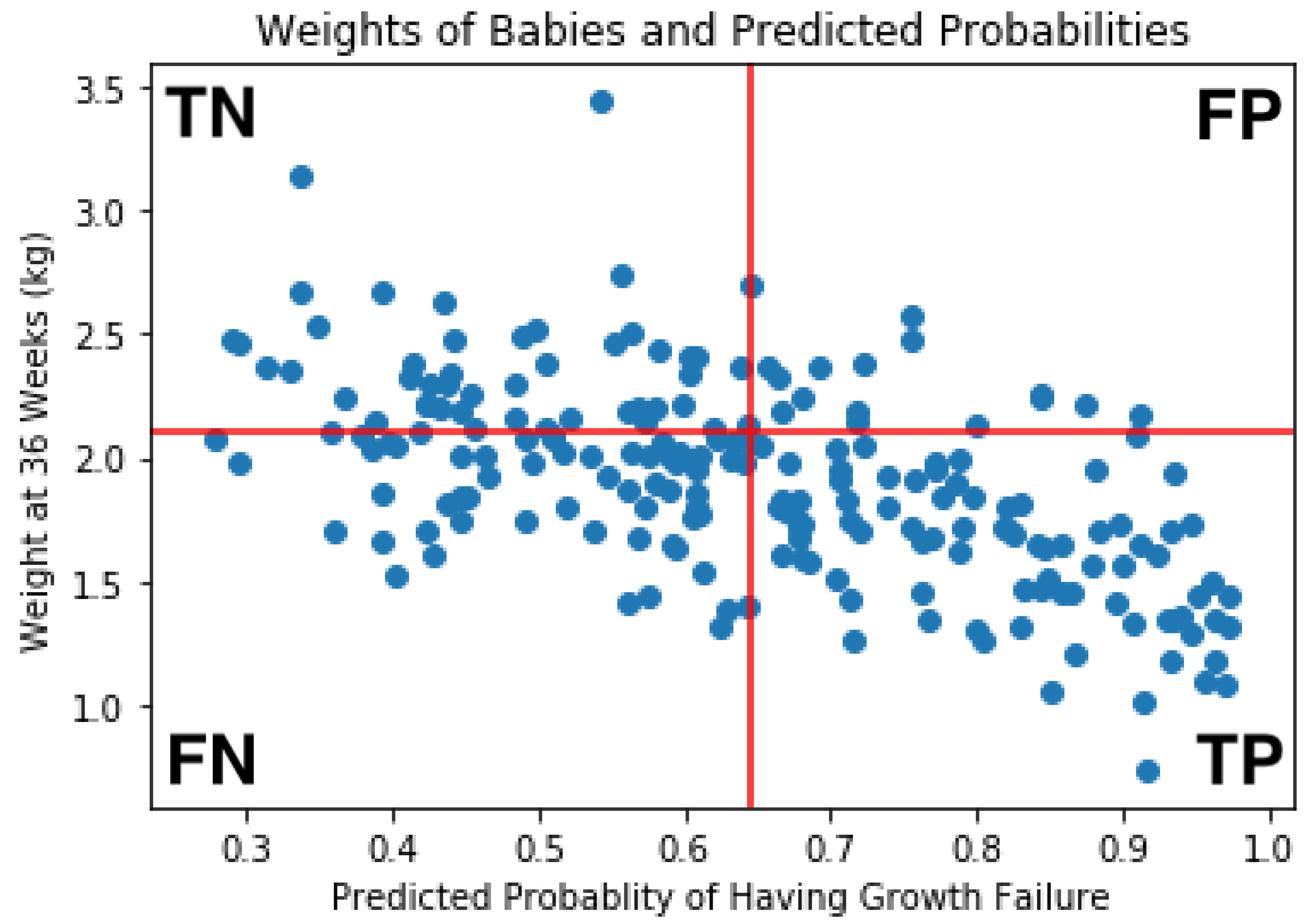

Figure 8

As figure 8 shows, most misclassified patients have weights close to the growth failure weight threshold of 2.1 kg. We suggest that a risk of growth failure be predicted instead of a binary label.

Conclusions

- The use of Spark improves the speed and computational capabilities of our machines, allowing for analyses not previously possible.
- Project-agnostic functions were developed and benchmarked for optimal performance which will aid future projects.
- We provide proof that notes is a feature with predictive potential, justifying inclusion of notes as a feature with other variables for modeling growth failure.

References

[1] Sehn, Dinghan et al. 2018, *Baseline Needs More Love: On Simple Word-Embedding-Based Models and Associated Pooling Mechanisms*, **arXiv:1805.09843v1**.

[2] Thomas Mikolov, Wen-tau Yih and Geoffrey Zweig. 2013 *Linguistic Regularities in Continuous Space Word Representations* **Proceedings of NAACL-HLT 2013, 746–751**.

Acknowledgments

We would like to acknowledge the following for their contributions to this project: Project Leads AJ Overton and Dr. Ricardo Henao, Mark McCahill, Dr. Noelle Younge, Dr. Paul Bendich, Ariel Dawn and Ursula Rogers.