Nile University

School of Communication and Information Technology (CIT)

Master of Science/Engineering MSCIT/ MCIT Program

Status Report 15

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version No.** | **Date** | **Description** | **Created By** | **e-mail** |
| 0.1 | 15-01-2019 | Status Report 15 | Ahmed Mohamed Abdel Rahman | [Robot209@gmail.com](mailto:Robot209@gmail.com) |
|  |  |  |  |  |

## Objective

* StudyLinear Discriminant Analysis (LDA) classifier
* Study how can LDA be distributed
* Implement LDA inside Spark MLlib

## Basic Concepts

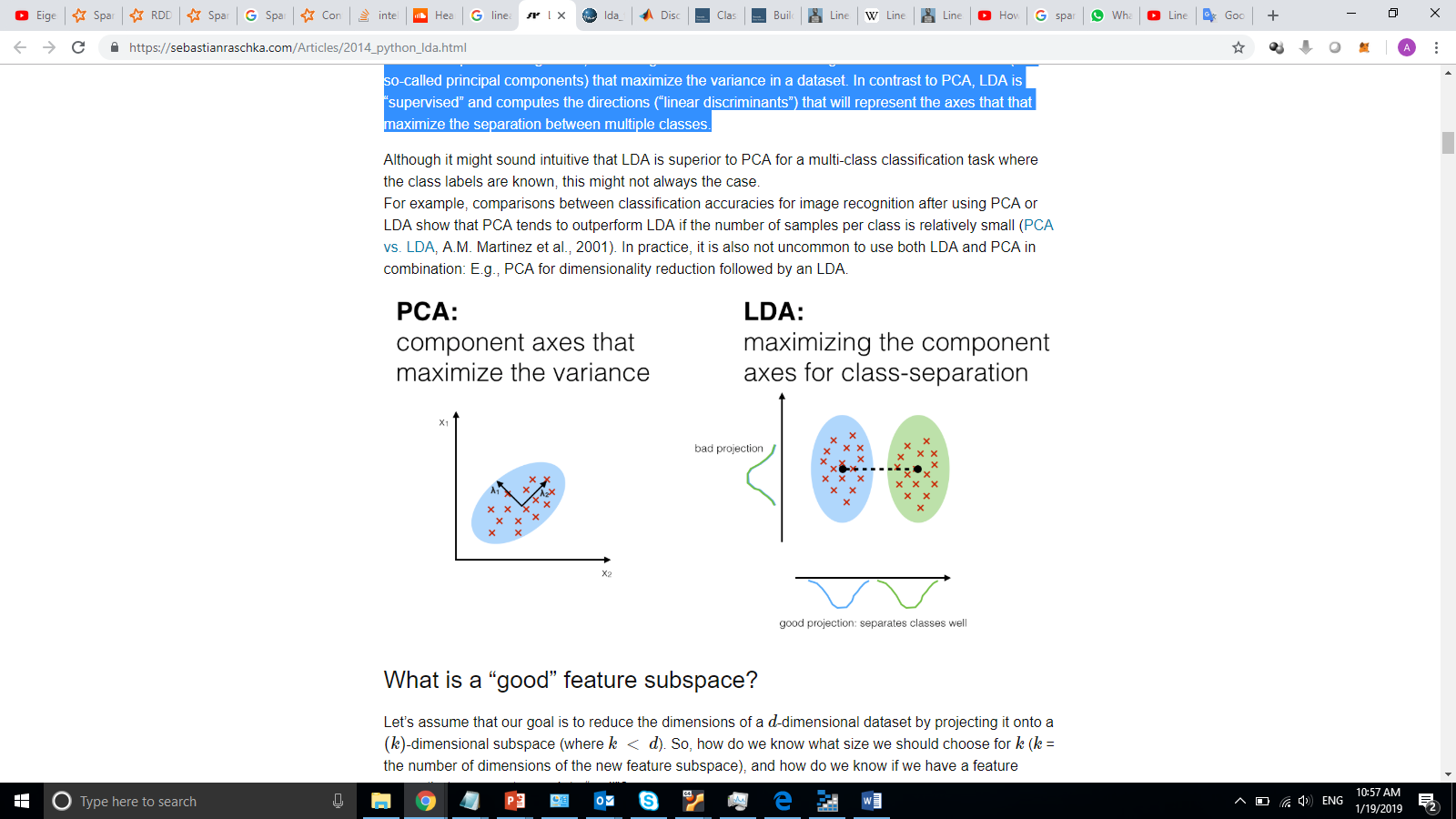
* **Linear Transformation**
* **Co-variance Matrix**
* **Eigen Vectors and Eigen Value**
* **Single value Decomposition**
* **Gaussian Distribution**
* **Multivariate Gaussian Distribution**
* **Bayes Rules**

## Linear Discriment Analysis

* LDA is a Classification method originally developed by A.R.Fisher @ 1936
* LDA is based upon the concept of searching for a linear combination of features that best separate classes.
* LDA usually used as a dimensionality reduction to process data before applying machine learning Algorithm.
* The goal is to project the dataset to a lower dimension space with good class reparability

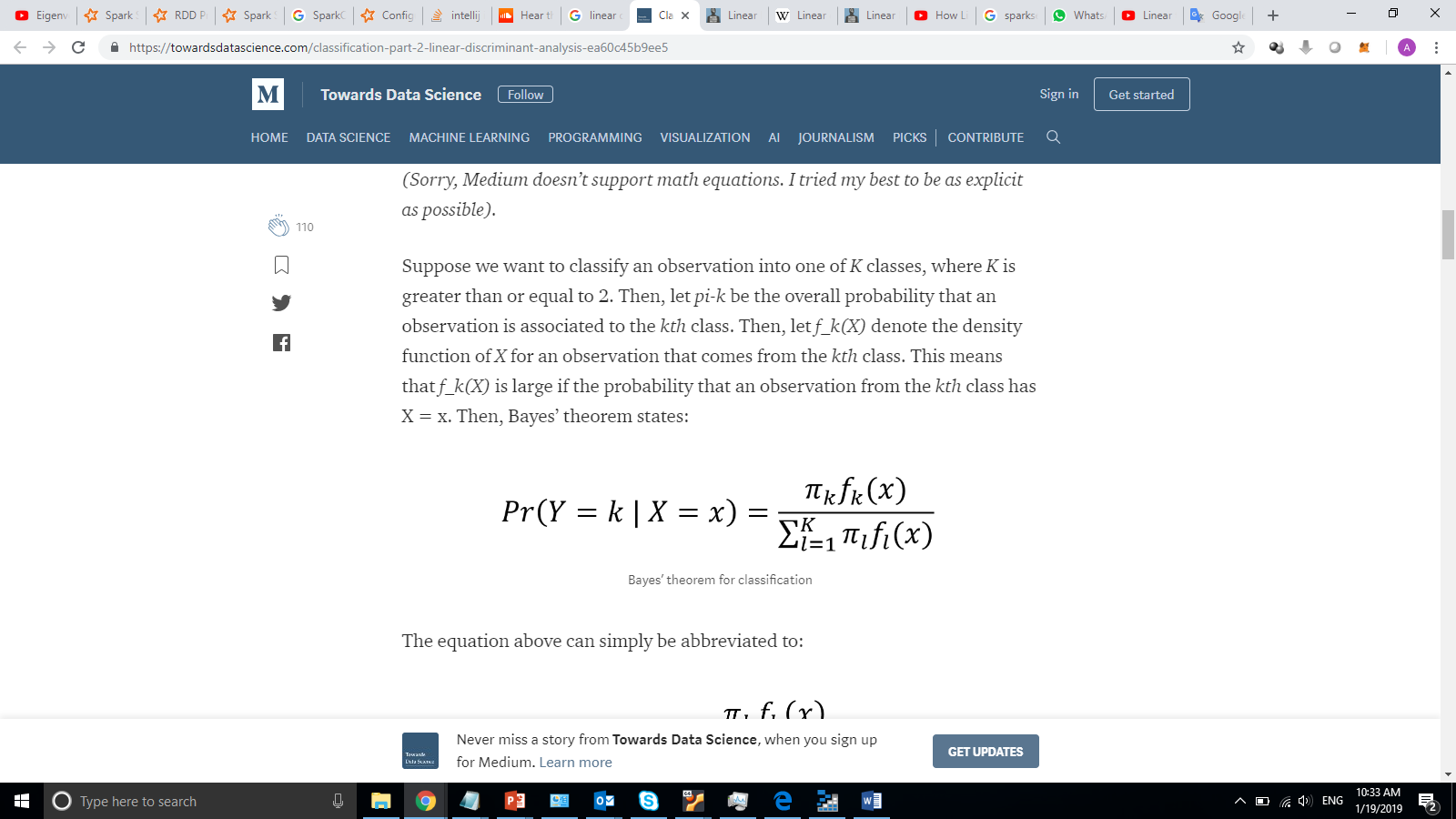
**LDA vs PCA**

* Both Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA) are linear transformation techniques that are commonly used for dimensionality reduction.
* PCA can be described as an “unsupervised” algorithm, since it “ignores” class labels and its goal is to find the directions (the so-called principal components) that maximize the variance in a dataset. In contrast to PCA.
* LDA is “supervised” and computes the directions (“linear discriminants”) that will represent the axes that that maximize the separation between multiple classes.

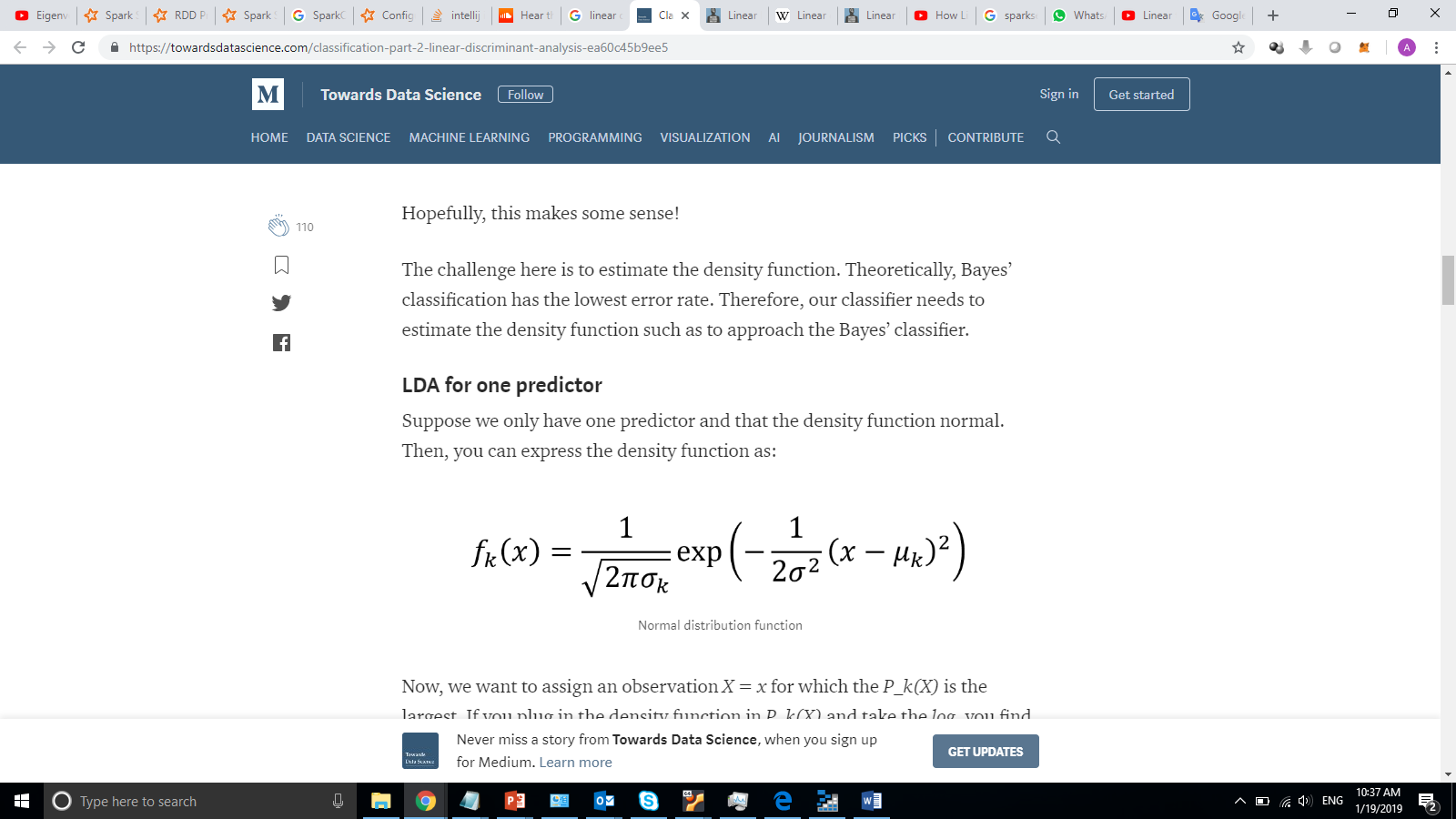


**How LDA Work?**

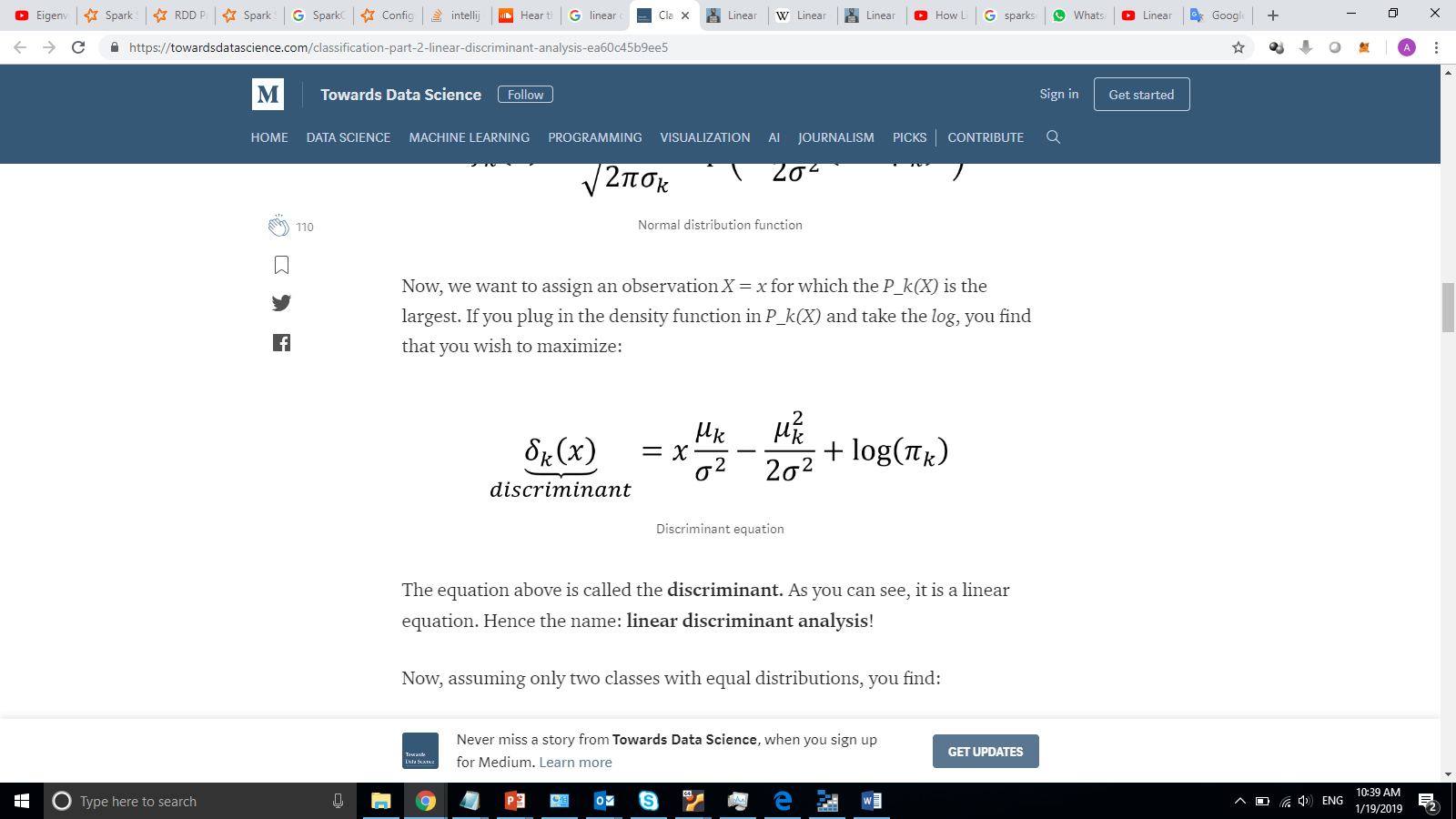
* LDA makes predictions by estimating the probability that a new set of inputs belongs to each class. The class that gets the highest probability is the output class and a prediction is made
* The model uses Bayes Theorem to estimate the probabilities, it models the distribution of predictors separately in each of the response classes, and then it uses Bayes’ theorem to estimate the probability.
* Suppose we want to classify an observation into one of *K* classes, where *K* is greater than or equal to 2. Then, let *pi-k* be the overall probability that an observation is associated to the *kth* class. Then, let *f\_k(X)* denote the density function of *X* for an observation that comes from the *kth* class. This means that *f\_k(X)* is large if the probability that an observation from the *kth*class has X = x. Then, Bayes’ theorem states:



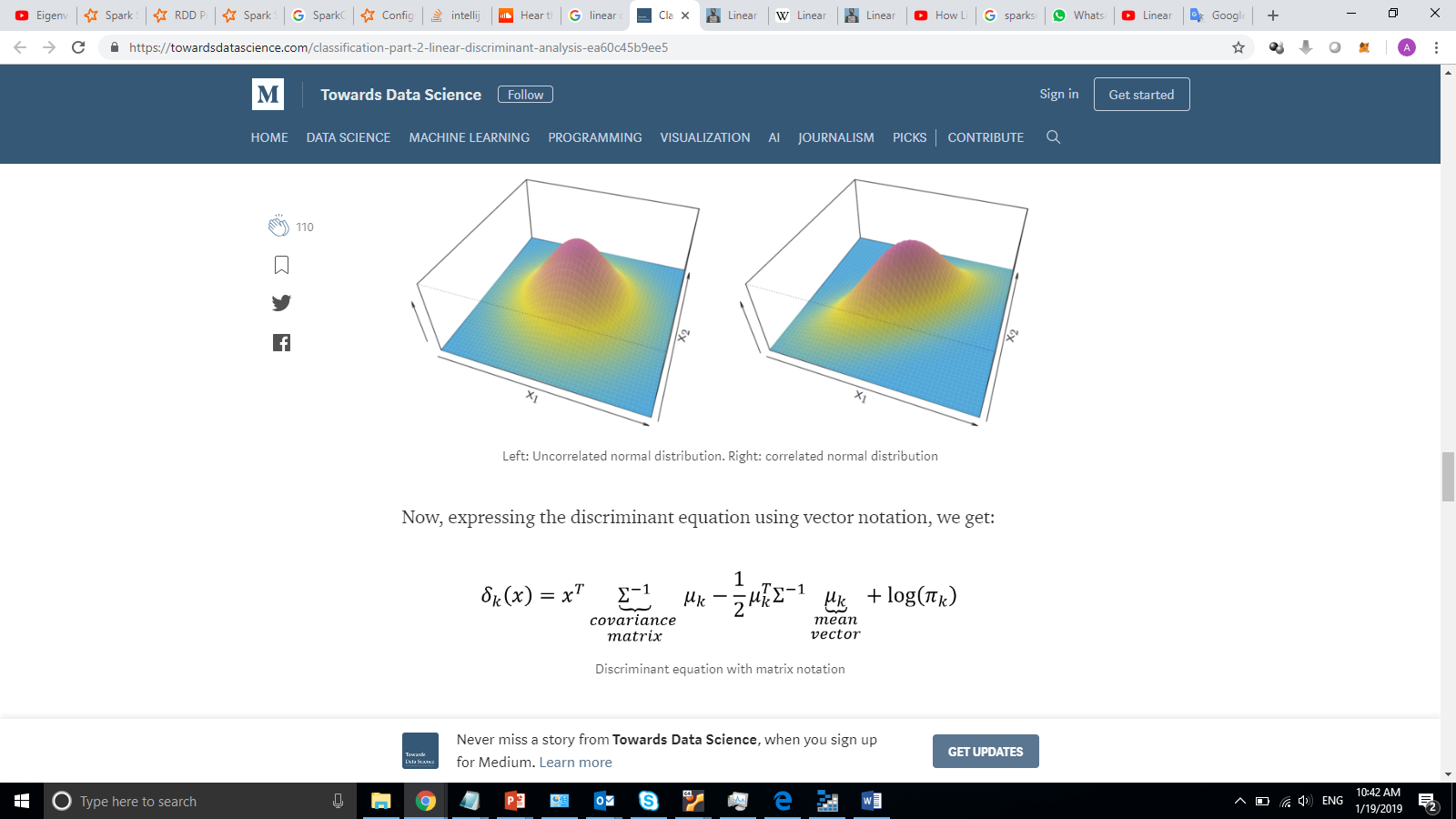
* The challenge here is to estimate the density function.
* Suppose we only have one predictor and that the density function normal. Then, you can express the density function as:



* Now, we want to assign an observation *X = x* for which the *P\_k(X)*is the largest. If you plug in the density function in *P\_k(X)* and take the *log*, you find that you wish to maximize:

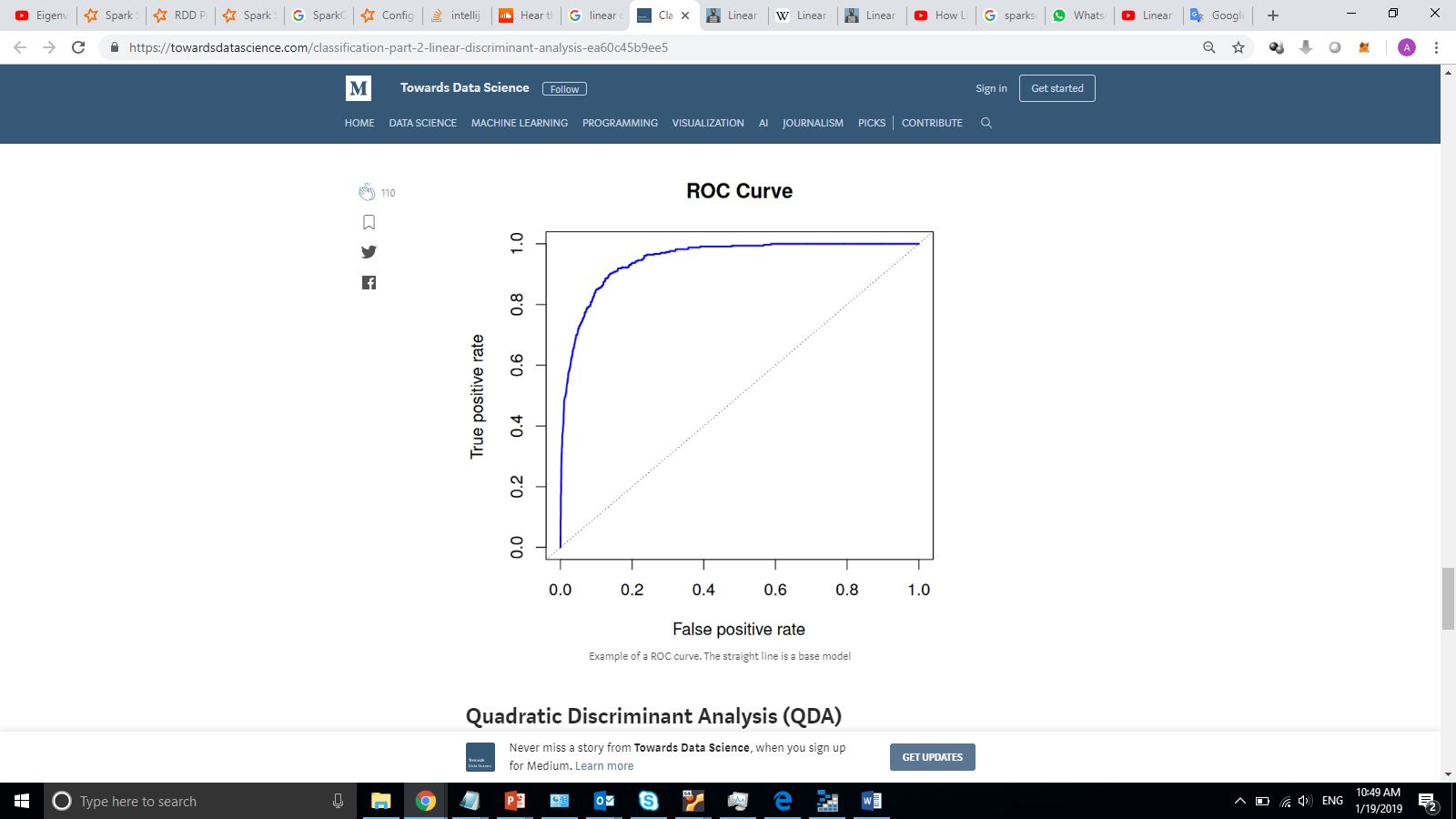


* Extending now for multiple predictors, we must assume that *X* is drawn from a **multivariate Gaussian distribution**, with a class-specific mean vector, and a common covariance matrix.



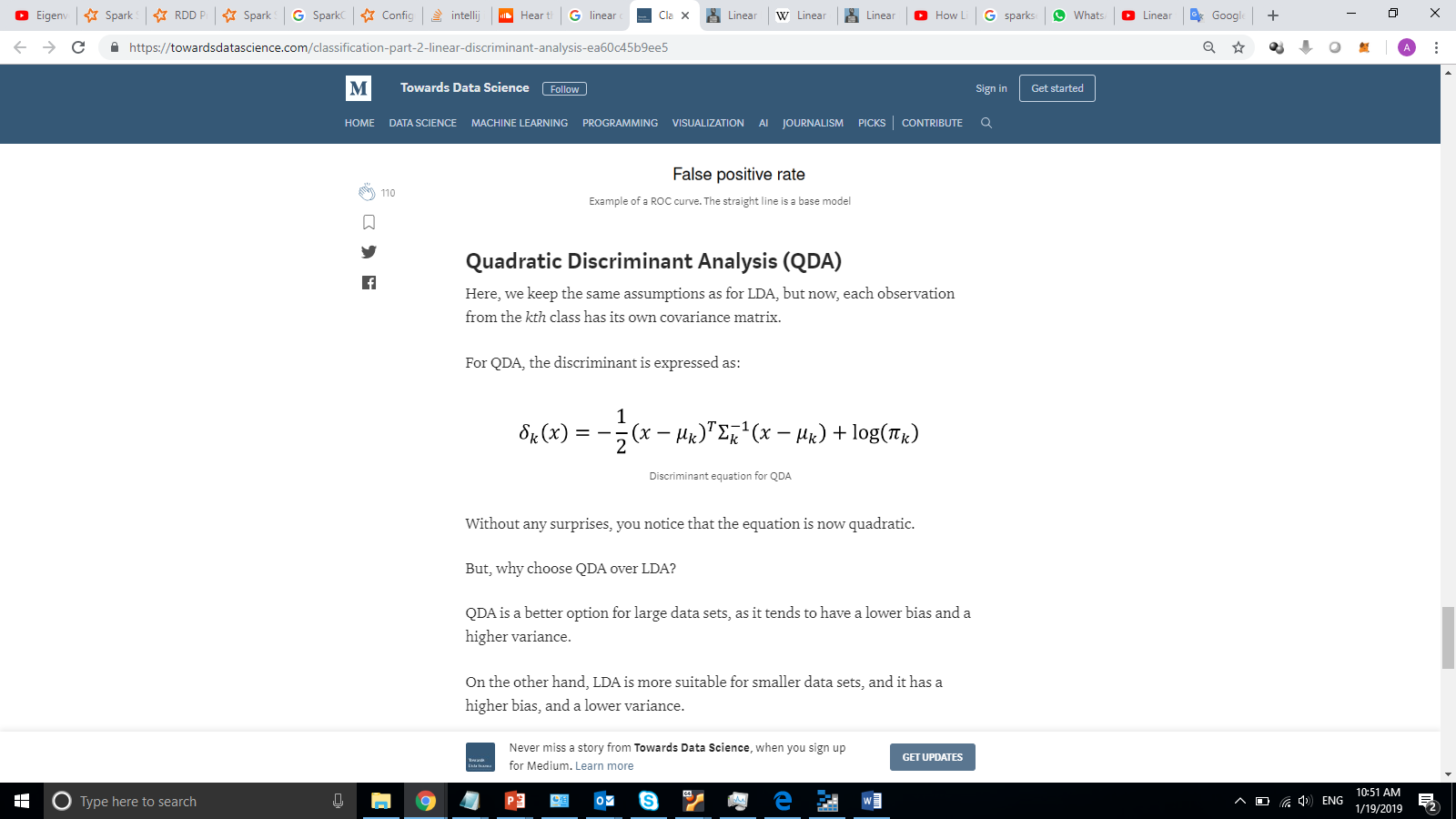
**Measure LDA Performance**

* With classification, it is sometimes irrelevant to use accuracy to assess the performance of a model.
* Usually, we use **sensitivity**and **specificity**.
  + **Sensitivity** is the true positive rate: the proportions of actual positives correctly identified.
  + **Specificity** is the true negative rate: the proportion of actual negatives correctly identified.
* Therefore, in an ideal situation, we want both a high sensitivity and specificity
* The **ROC curve** (receiver operating characteristic) is good to display the two types of error metrics described above. The overall performance of a classifier is given by the area under the ROC curve (**AUC**). Ideally, it should hug the upper left corner of the graph, and have an area close to 1.



**Quadratic Discriminant Analysis (QDA)**

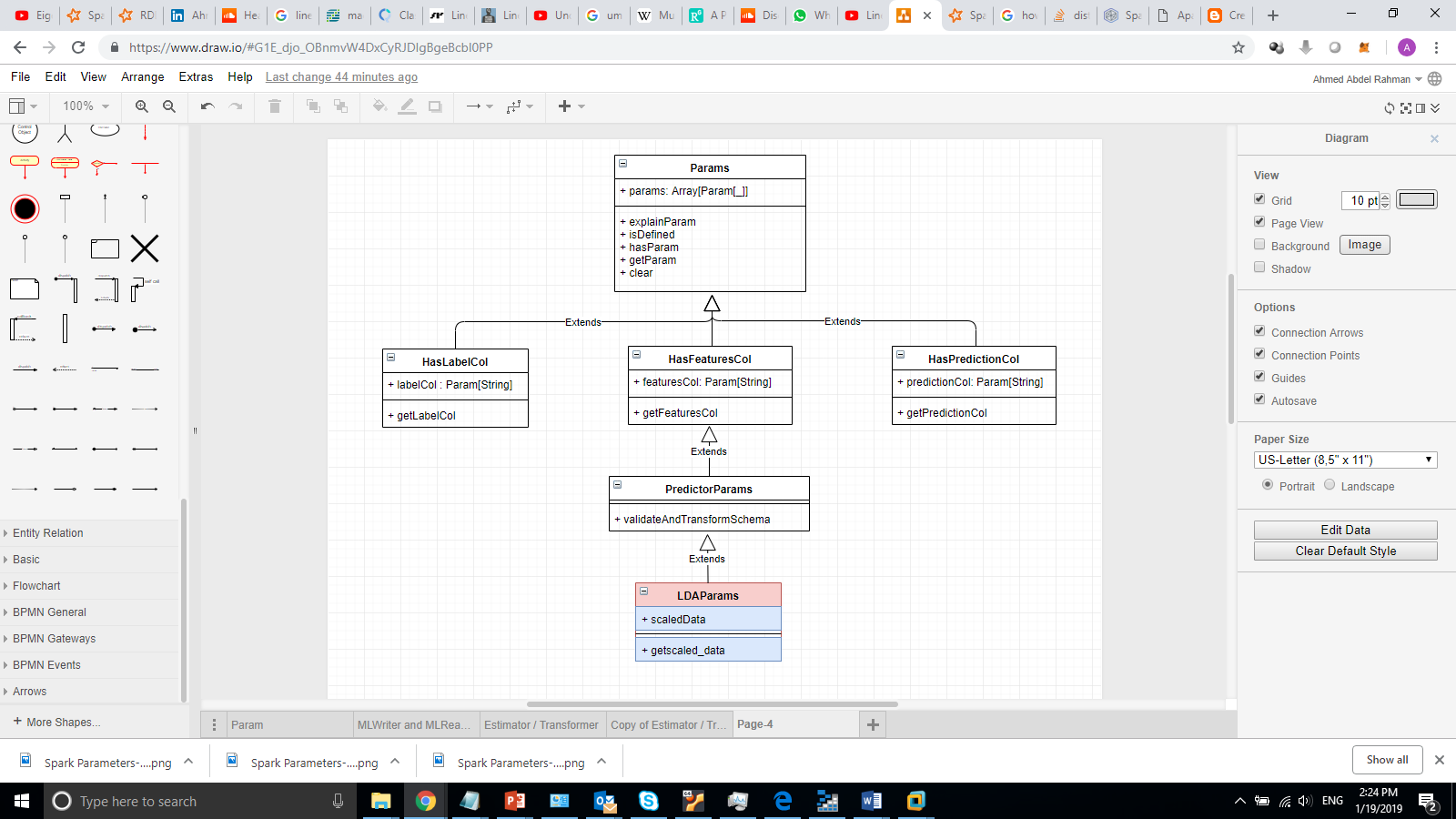
* Here, we keep the same assumptions as for LDA, but now, each observation from the *kth* class has its own covariance matrix.
* For QDA, the discriminant is expressed as:



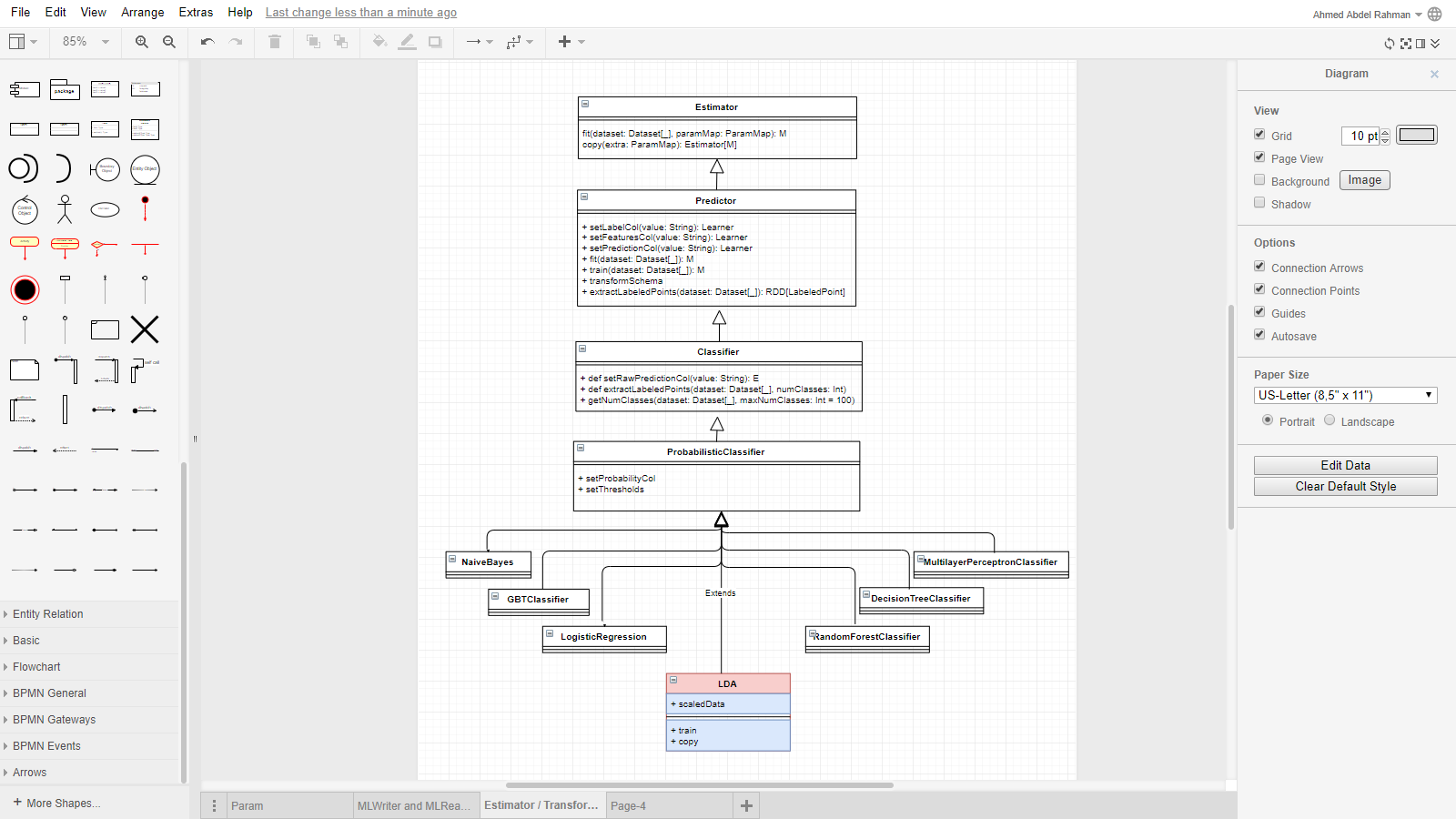
* QDA is a better option for large data sets, as it tends to have a lower bias and a higher variance.
* On the other hand, LDA is more suitable for smaller data sets, and it has a higher bias, and a lower variance.

## Adding LDA to Spark MLlib

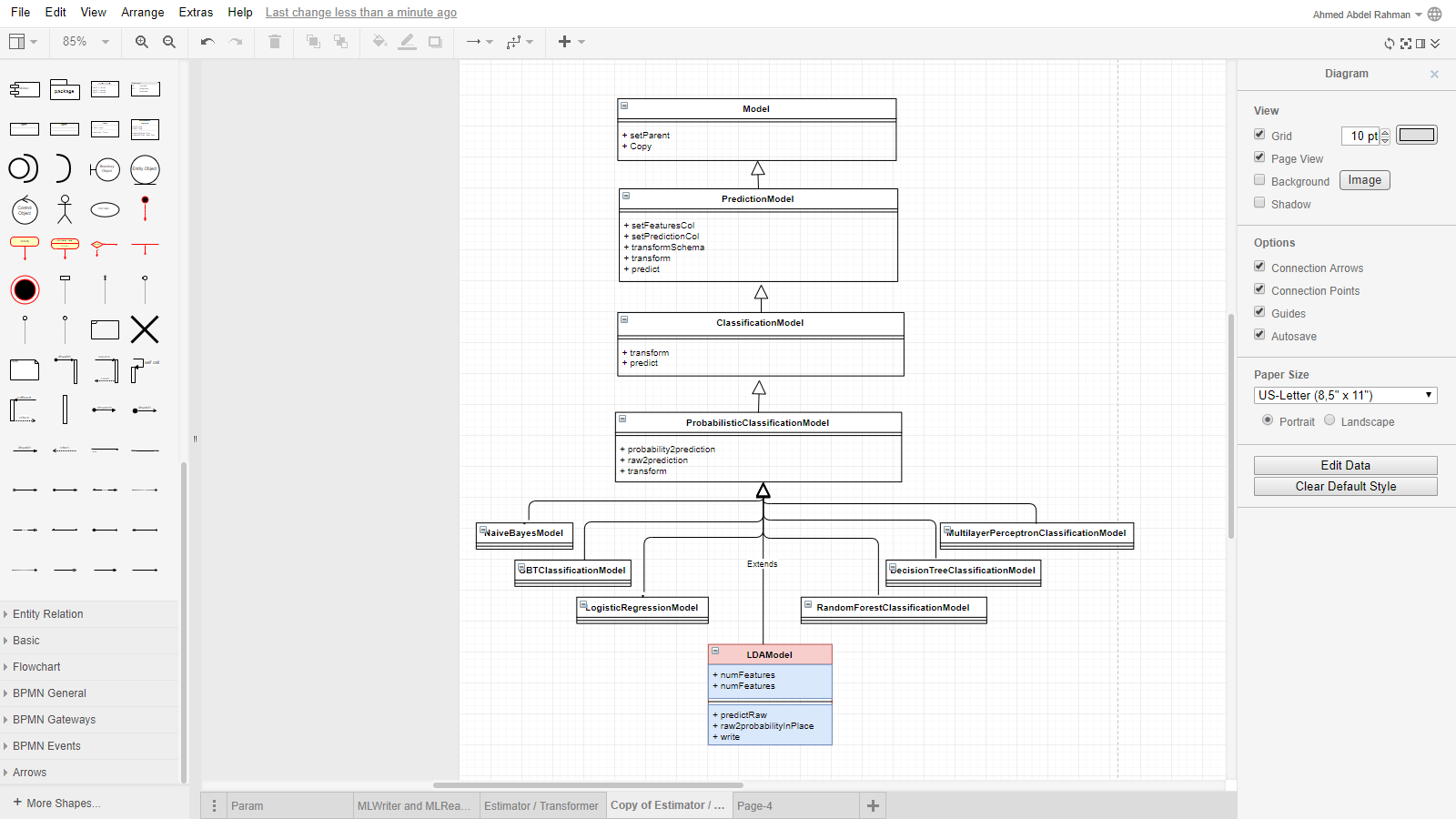
* In order to implement LDA inside Spark MLlib, I have developed four components:
  + Class for Algorithm Parameter **(LDAParams)**



* + - The parameter class provide the flowing parameters:
      * predictionCol: prediction column name
      * labelCol: label column name
      * featuresCol: features column name
      * scaledData: is the data need to be scaled or not?
  + Class for the Algorithm (LDA)
    - This class responsible for the training, it learns from input dataset and produce Model as an output.
    - This class inherit from “**ProbabilisticClassifier”**



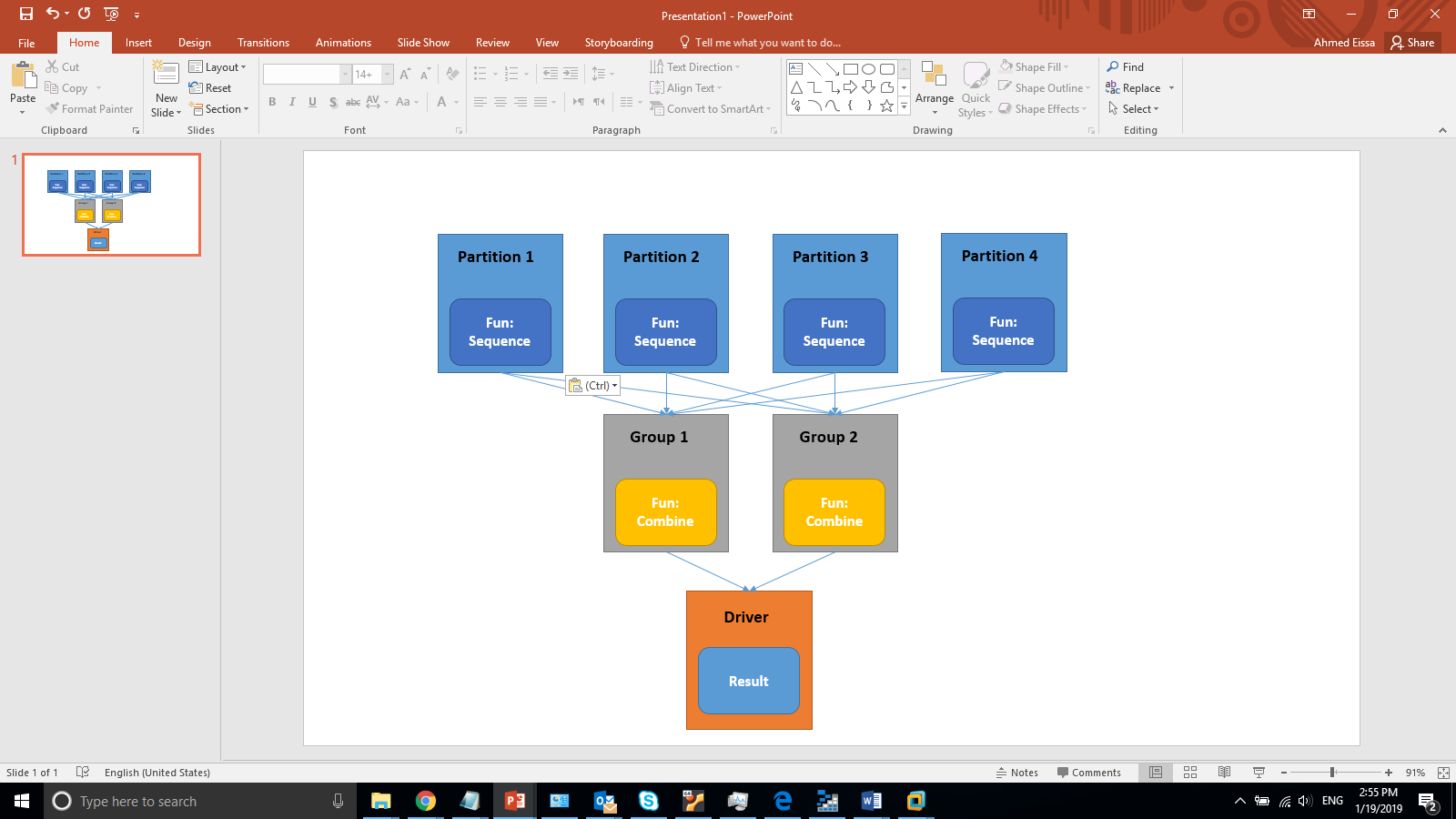
* + Class for the Model (LDAModel)
    - Represent the trained Model, this model able to predict label for input data
    - It is inheriting from “**ProbabilisticClassificationModel”**
    - LDAModel class is a Transformer class (at the end it inherits from Transformer Class) it receives dataset and transform it (add new column for the predicted label)



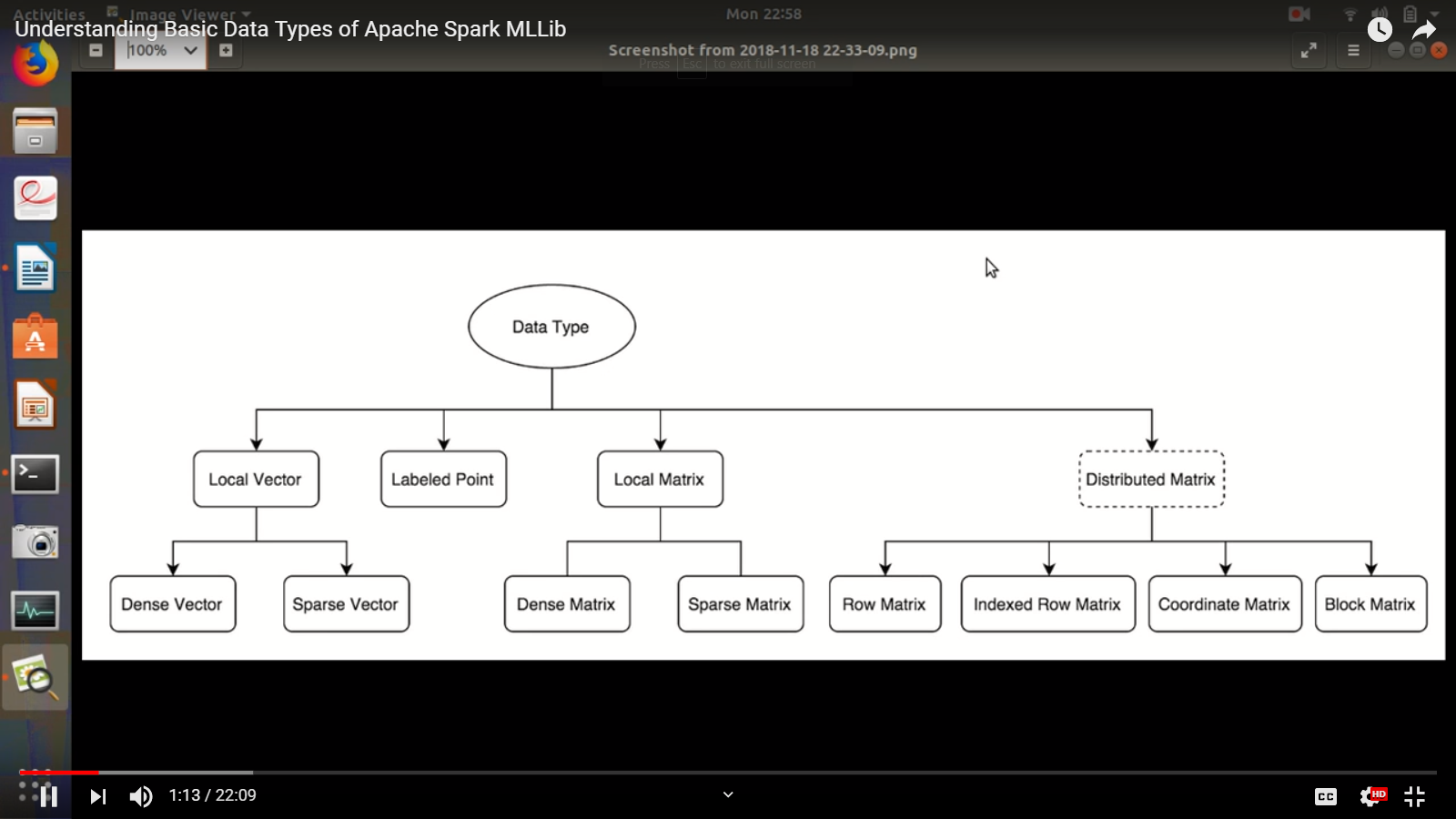
* + Utility Class
    - It is a helper class
    - Main function is to calculate matrix inverse using singular value decomposition

## How LDA will be Distributed

* Calculate statistics
  + In order to calculate (mean vector and count of instances) per each class I am using AggregateByKey function

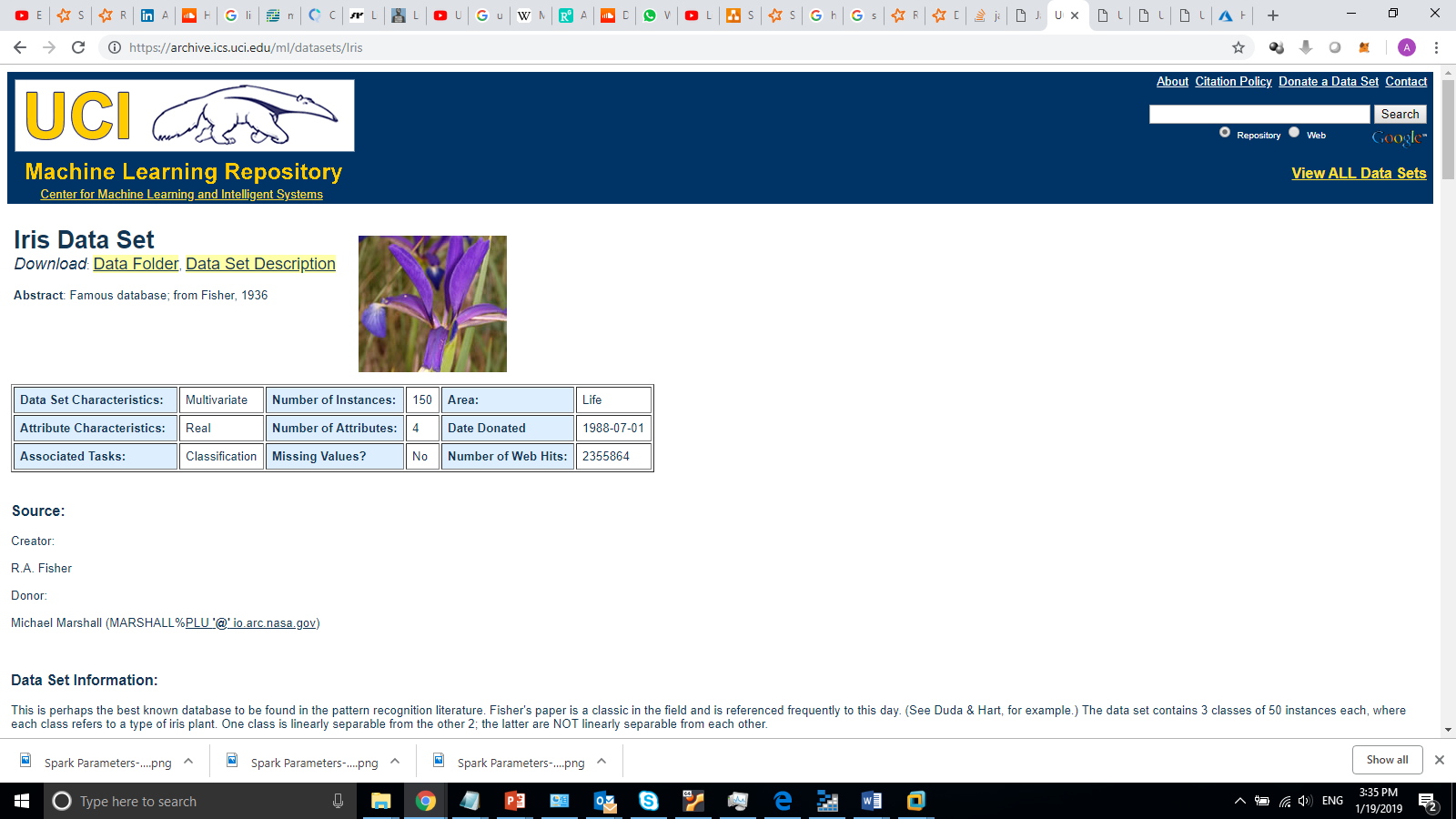


* + Sequence function work per partition to calculate the count and mean per class
  + Combine function work across partitions.
* Matrices operation
  + MLlib supports local vectors and matrices stored on a single machine, as well as distributed matrices backed by one or more RDDs.
  + I have used distributed matrix which has long-typed row and column indices and double-typed values, stored distributively in one or more RDDs
  + I have used “RowMatrix” to distributed the matrix operation (mainly to calculate the covariance matrix.



## MLlib LDA Examples

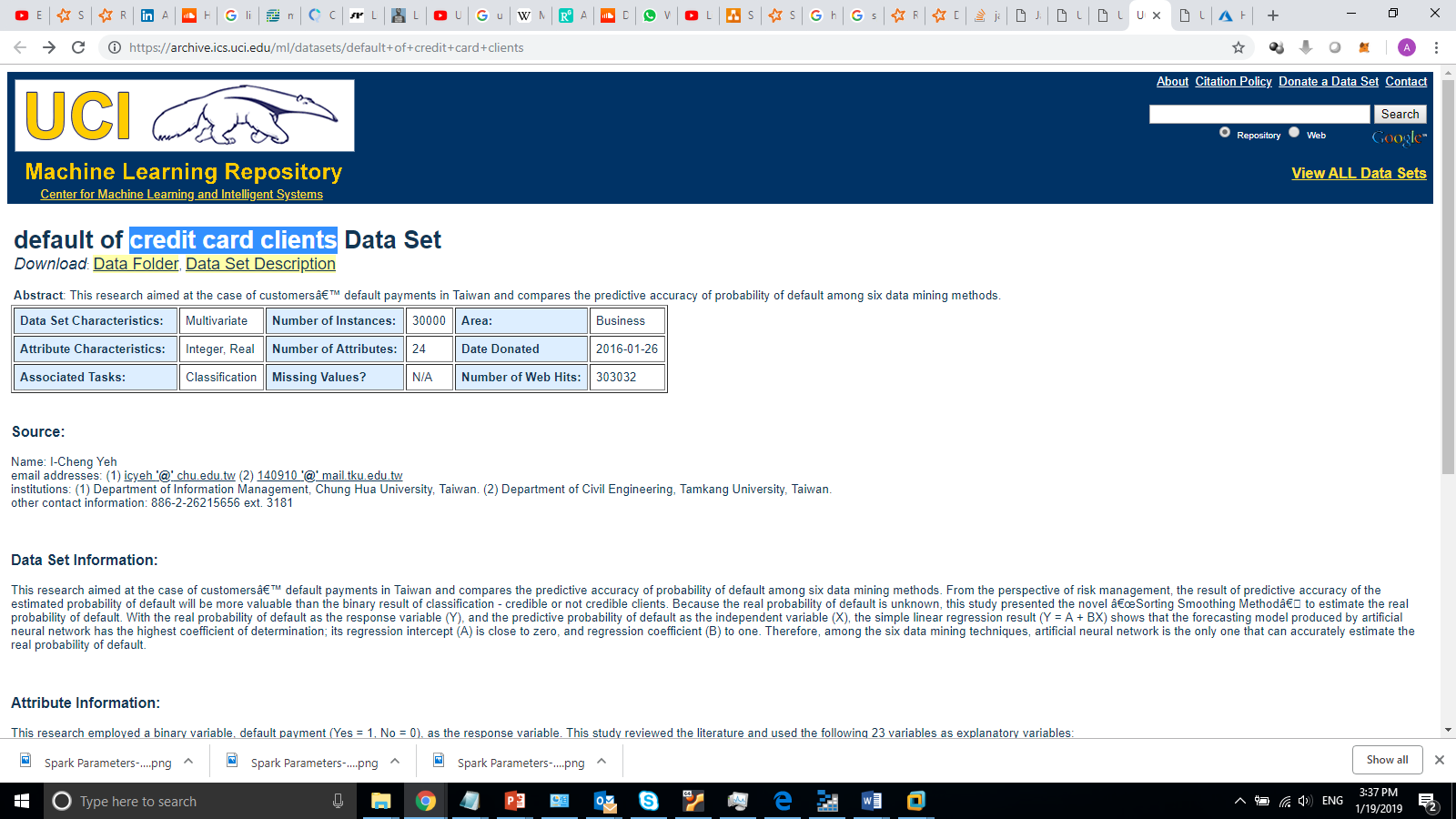
* Iris Data Set



<https://archive.ics.uci.edu/ml/datasets/Iris>

**Summary Statistics**

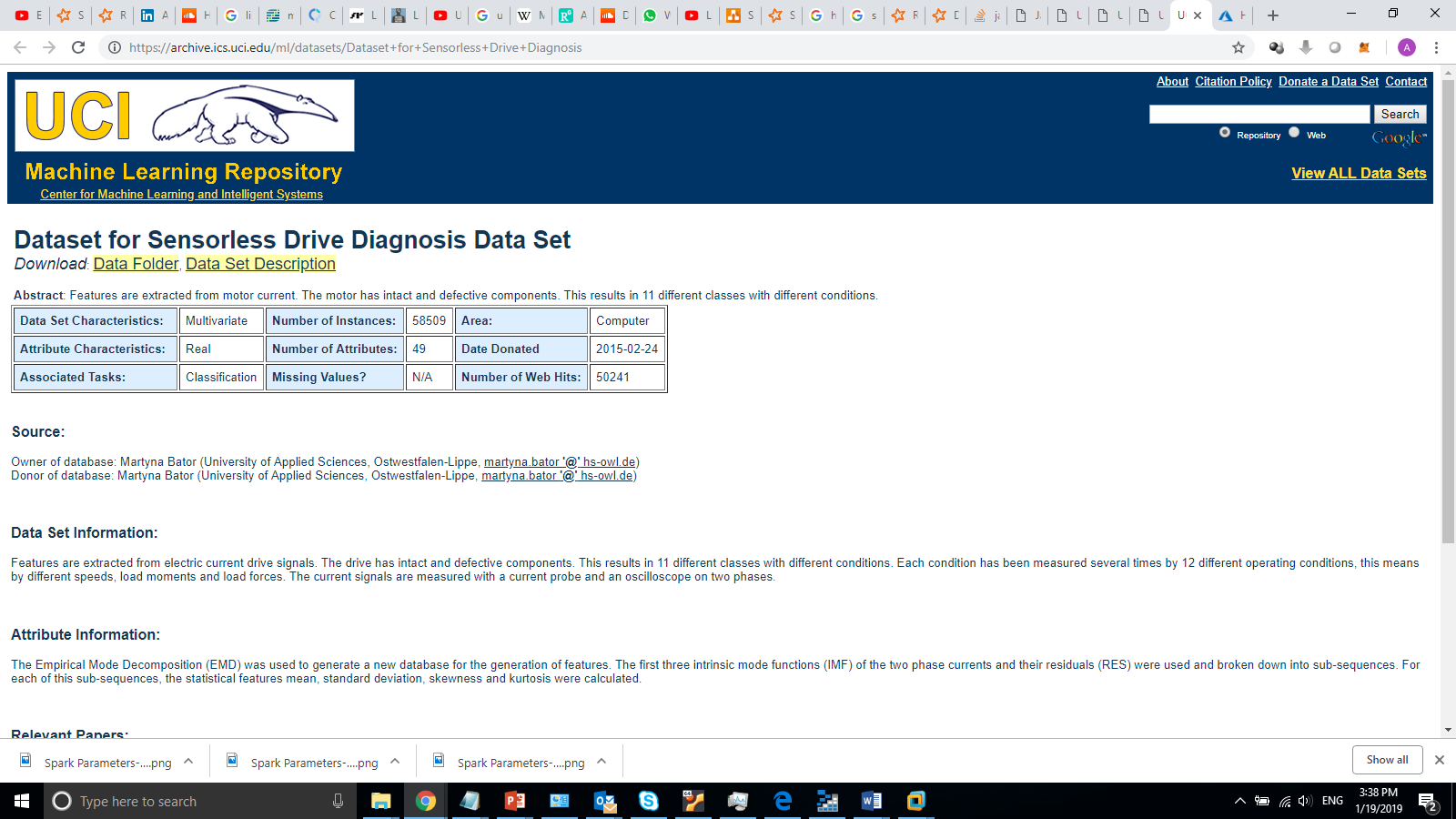
* + Accuracy = 0.984375
  + Precision (0.0) = 1.0
  + Precision (1.0) = 0.9545454545454546
  + Precision (2.0) = 1.0
  + Time: 45 Second
* credit card clients



<https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients>

**Summary Statistics**

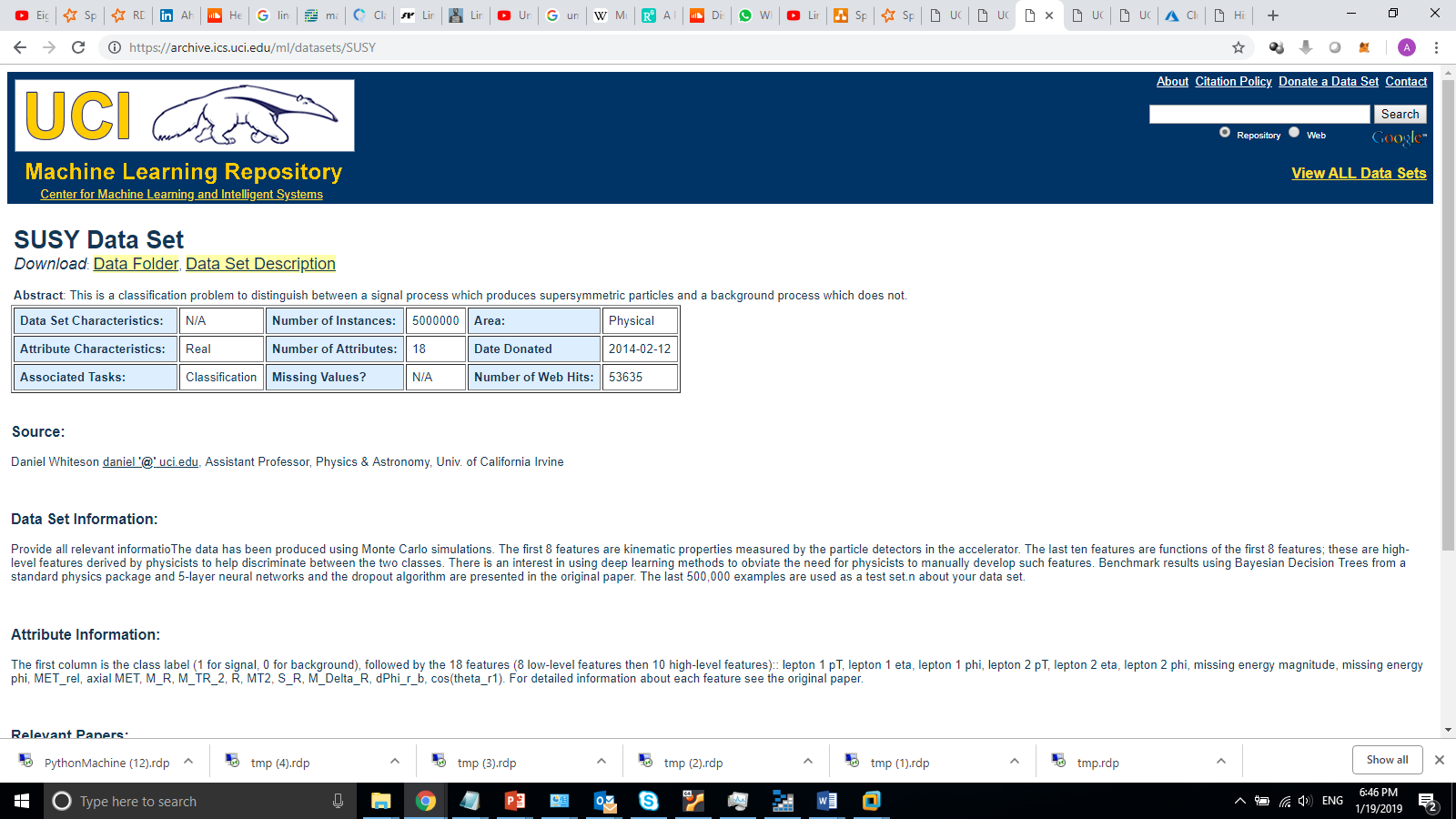
* + Accuracy = 0.8146944083224967
  + Precision (0.0) = 0.9207547169811321
  + Precision (1.0) = 0.44862518089725034
  + Time: 40 Second
* Sensor less Drive Diagnosis



<https://archive.ics.uci.edu/ml/datasets/Dataset+for+Sensorless+Drive+Diagnosis>

**Summary Statistics**

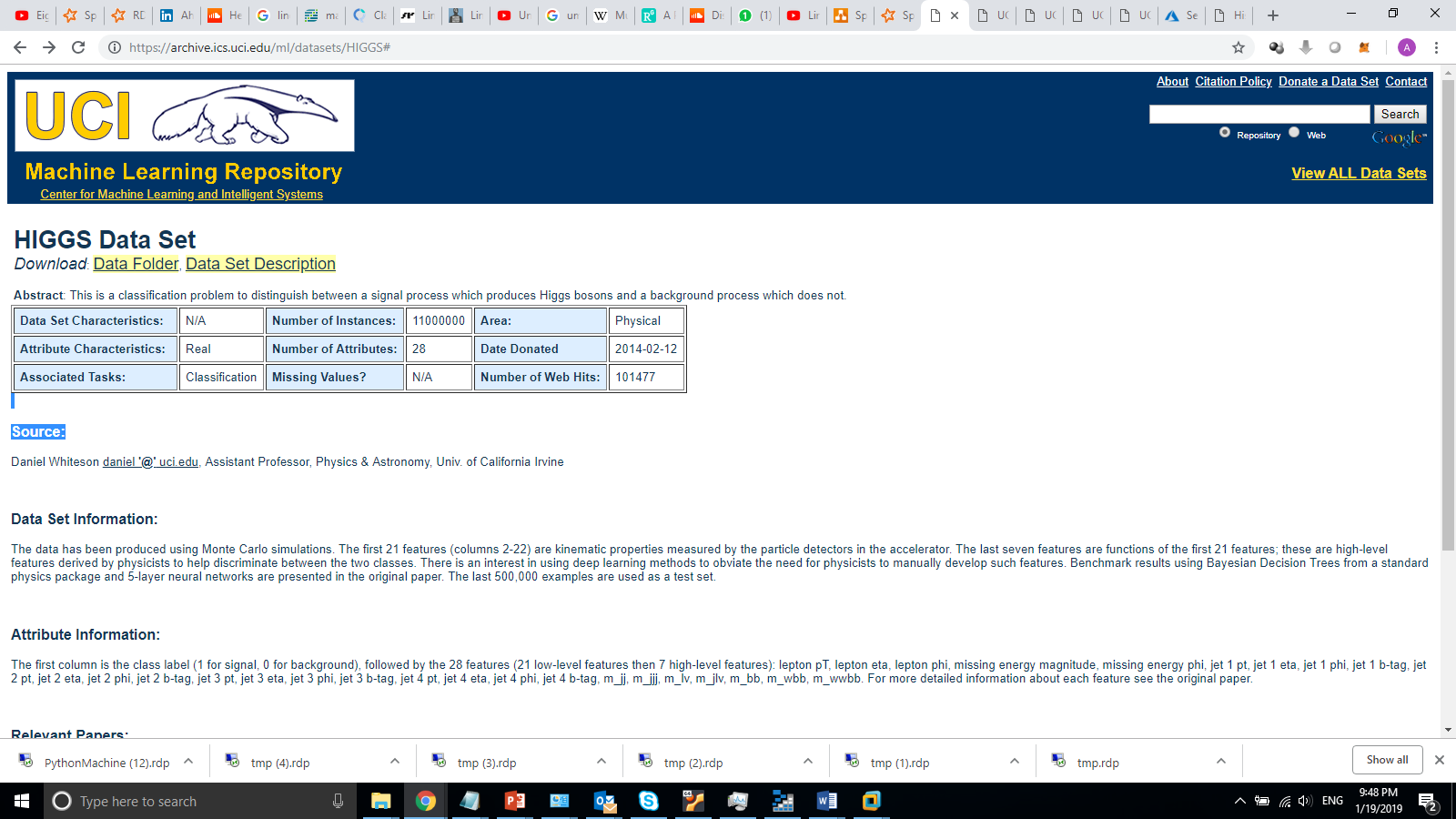
* + Accuracy = 0.7707168276563294
  + Precision (0.0) = 1.0
  + Precision (1.0) = 0.7151277013752456
  + Precision (2.0) = 0.4822429906542056
  + Precision (3.0) = 0.9105058365758755
  + Precision (4.0) = 0.7419962335216572
  + Precision (5.0) = 0.7704626334519573
  + Precision (6.0) = 0.47761194029850745
  + Precision (7.0) = 0.9942418426103646
  + Precision (8.0) = 0.6835664335664335
  + Precision (9.0) = 0.7652495378927912
  + Precision (10.0) = 0.9447731755424064
  + Precision (11.0) = 1.0
  + Time: 1.3 Minutes
* SUSY



<https://archive.ics.uci.edu/ml/datasets/SUSY>

**Summary Statistics**

* + Accuracy = 0.7621754610553266
  + Precision (0.0) = 0.8955602490159733
  + Precision (1.0) = 0.6045069266827867
  + Time: 5 minutes
* HIGGS



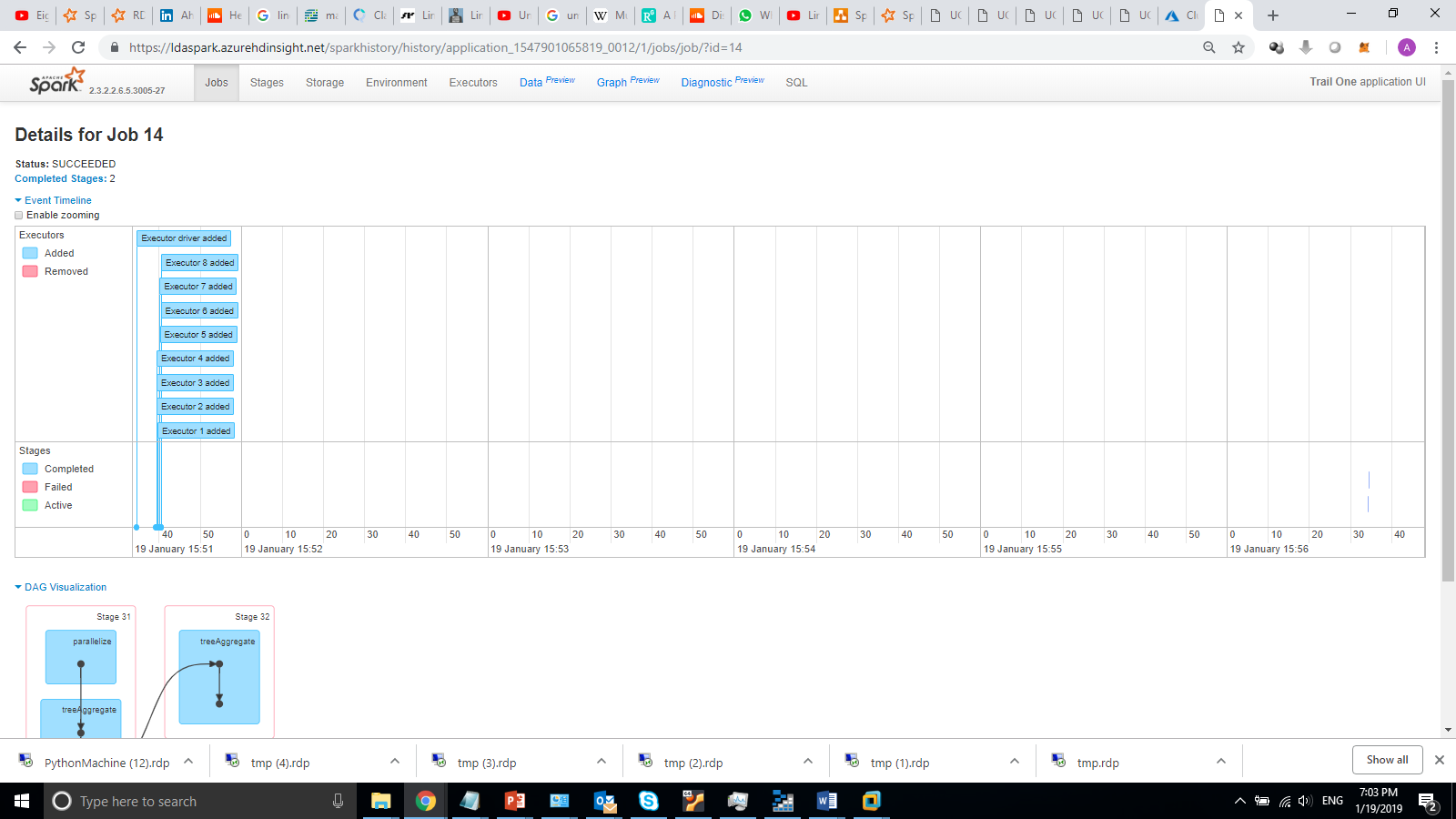
<https://archive.ics.uci.edu/ml/datasets/HIGGS#>

**Summary Statistics**

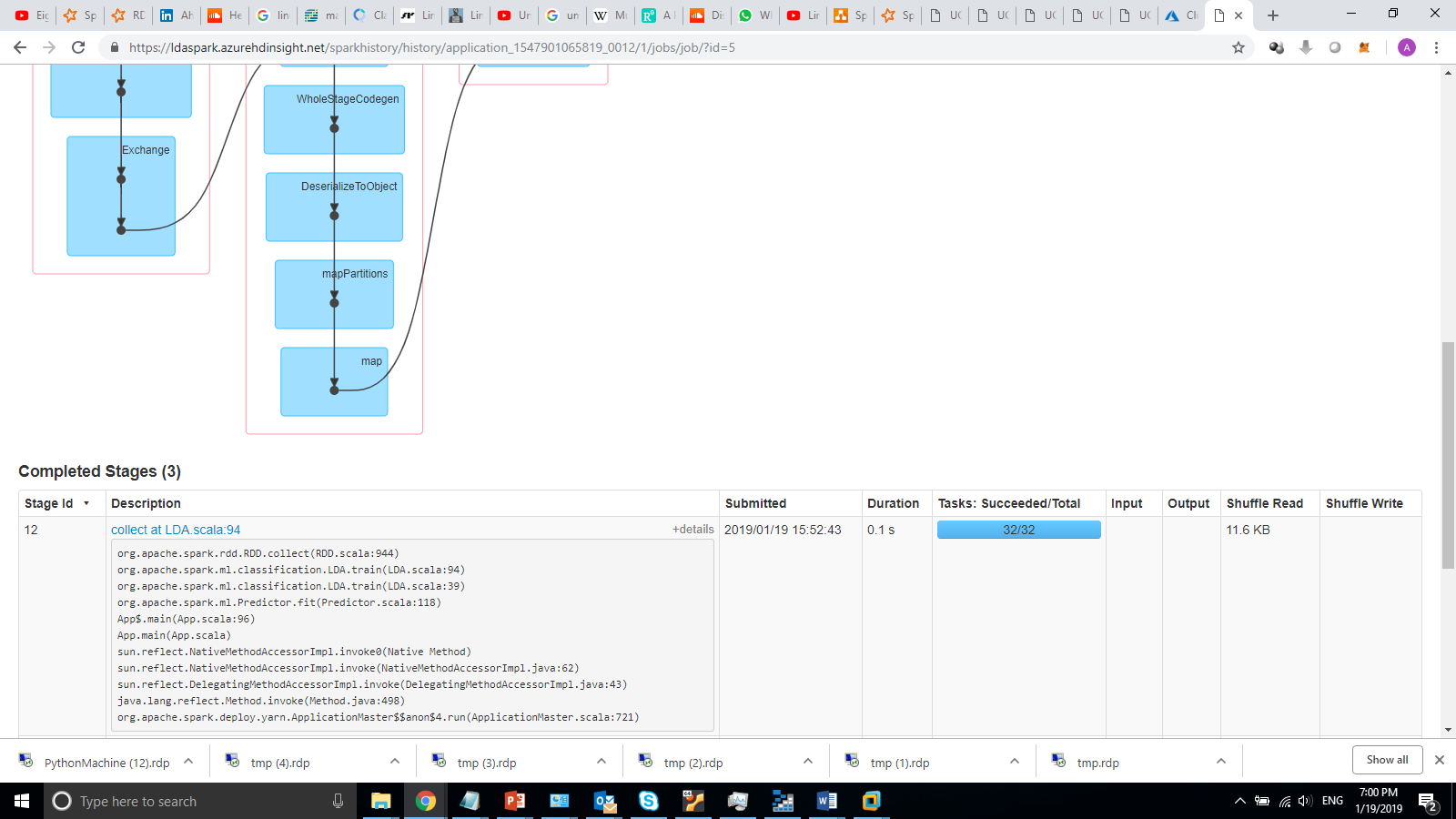
* + Accuracy = 0.6389403524884008
  + Precision (0.0) = 0.5706670536448508
  + Precision (1.0) = 0.6994716324071175
  + Time: 4.4 Minutes

**Distribution of a job from Train Function**

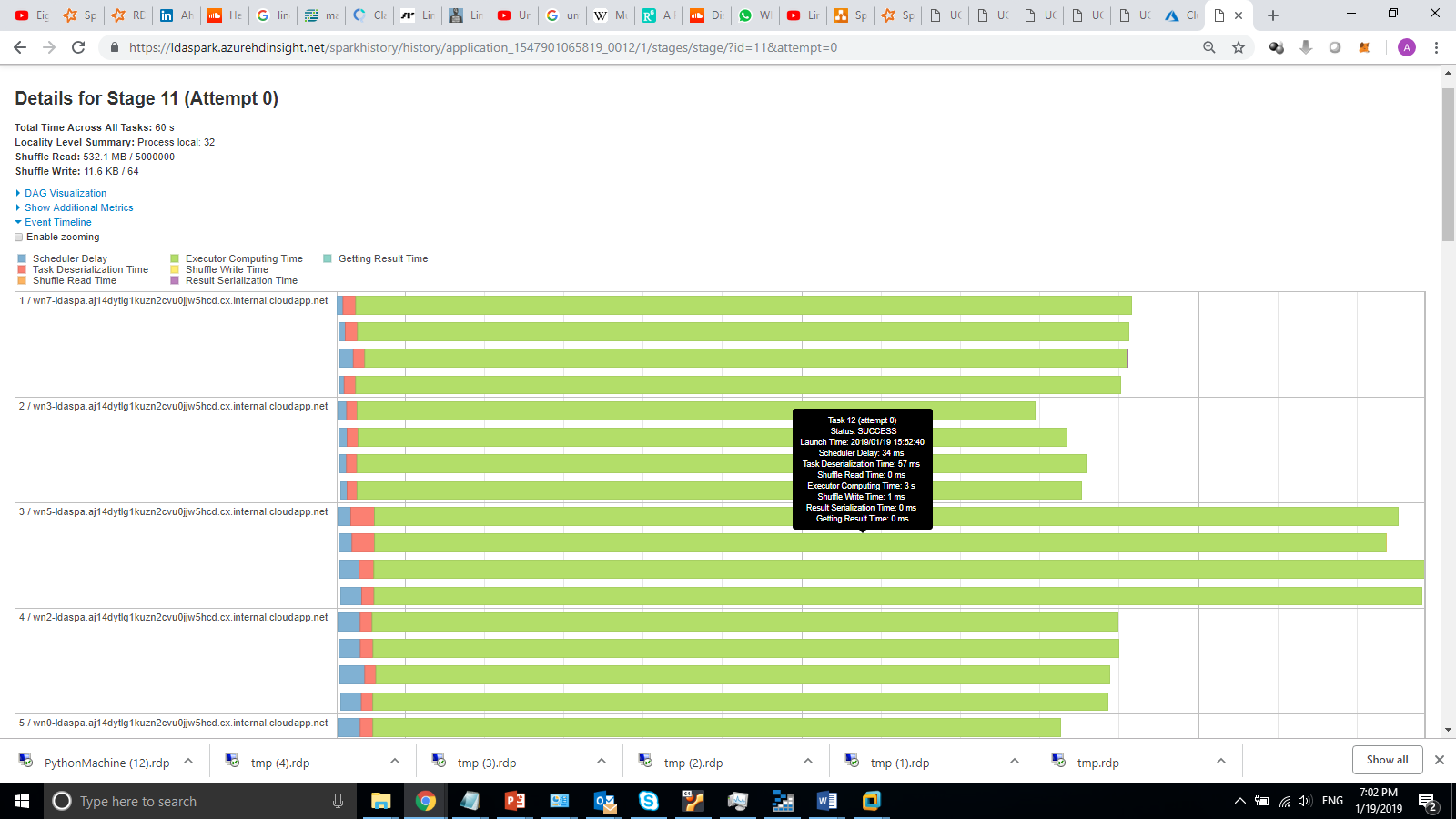
* + 8 executers added and working in parallel



* + 32 tasks (4 cores X 8 Executers) – 32 parathions processed in parallel



* + A sample stage of one job, it shows that computation is distributed between nodes



## Resources

* LDA
* <https://towardsdatascience.com/classification-part-2-linear-discriminant-analysis-ea60c45b9ee5>
* <https://sebastianraschka.com/Articles/2014_python_lda.html>
* <https://www.isip.piconepress.com/publications/reports/1998/isip/lda/lda_theory.pdf>
* Spark Matrices and vectors
* <https://spark.apache.org/docs/2.2.0/mllib-data-types.html>