

Amrita School of Engineering, Bengaluru
Amrita Vishwa Vidyapeetham



Course: 19AIE214 – Big Data Analytics

Course Instructor: Mr. S. Saravanan

Course Project Title: Flight Delay Prediction using PySpark

Team Members:

- 1.) BL.EN.U4AIE19013 - CHARAN TEJ K.**
- 2.) BL.EN.U4AIE19014 - CHAVALI SURYA TEJA**
- 3.) BL.EN.U4AIE19062 - SUGASH T.M.**

TABLE OF CONTENTS

- Abstract
- Chapter 1 – Introduction
- Chapter 2 – System Design
- Chapter 3 – Implementation
- Chapter 4 – Results
- Chapter 5 – Conclusion

ABSTRACT

In this project, we are going to predict the delay in arrival for several flights across different airports. Considering our data size (70 MB), using normal machine learning libraries won't be an efficient method here, therefore, we have to go with PySpark for dealing with such huge amounts of data. We work with PySpark SQL and ML libraries, to deal with our data.

We have used Logistic Regression machine learning algorithm to train our model. The model predicts a binary output consisting as 0 (meaning, the flight has not delayed) and 1 (meaning the flight has delayed). Later, we evaluate our model with several performance metrics such as Accuracy and Precision.

INTRODUCTION

Spark:

Apache Spark is a cluster computing platform designed to be fast and general purpose. On the speed side, Spark extends the popular MapReduce model to efficiently support more types of computations, including interactive queries and stream processing. Speed is important in processing large datasets, as it means the difference between exploring data interactively and waiting minutes or hours. One of the main features Spark offers for speed is the ability to run computations in memory, but the system is also more efficient than MapReduce for complex applications running on disk.

On the generality side, Spark is designed to cover a wide range of workloads that previously required separate distributed systems, including batch applications, iterative algorithms, interactive queries, and streaming. By supporting these workloads in the same engine, Spark makes it easy and inexpensive to combine different processing types, which is often necessary in production data analysis pipelines. In addition, it reduces the management burden of maintaining separate tools. Spark is designed to be highly accessible, offering simple APIs in Python, Java, Scala, and SQL, and rich built-in libraries. It also integrates closely with other Big Data tools. In particular, Spark can run in Hadoop clusters and access any Hadoop data source, including Cassandra.

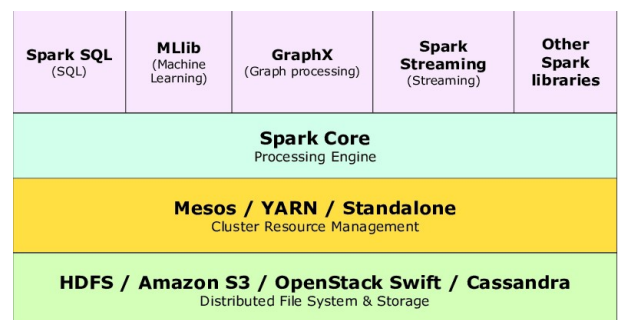


The Spark project contains multiple closely integrated components. At its core, Spark is a “computational engine” that is responsible for scheduling, distributing, and monitoring applications consisting of many computational tasks across many worker machines, or a computing

cluster. Because the core engine of Spark is both fast and general-purpose, it powers multiple higher-level components specialized for various workloads, such as SQL or machine learning. These components are designed to interoperate closely, letting you combine them like libraries in a software project. A philosophy of tight integration has several benefits. First, all libraries and higher level components in the stack benefit from improvements at the lower layers. For example, when Spark's core engine adds an optimization, SQL and machine learning libraries automatically speed up as well. Second, the costs associated with running the stack are minimized, because instead of running 5–10 independent software systems, an organization needs to run only one. These costs include deployment, maintenance, testing, support, and others. This also means that each time a new component is added to the Spark stack, every organization that uses Spark will immediately be able to try this new component. This changes the cost of trying out a new type of data analysis from downloading, deploying, and learning a new software project to upgrading Spark. Finally, one of the largest advantages of tight integration is the ability to build applications that seamlessly combine different processing models.

Spark Core:

Spark Core contains the basic functionality of Spark, including components for task scheduling, memory management, fault recovery, interacting with storage systems, and more. Spark Core is also home to the API that defines resilient distributed datasets (RDDs), which are Spark's main programming abstraction. RDDs represent a collection of items distributed across many compute nodes



that can be manipulated in parallel. Spark Core provides many APIs for building and manipulating these collections.

Spark SQL:

Spark SQL is Spark's package for working with structured data. It allows querying data via SQL as well as the Apache Hive variant of SQL—called the Hive Query Language (HQL)—and it supports many sources of data, including Hive tables, Parquet, and JSON. Beyond providing a SQL interface to Spark, Spark SQL allows developers to intermix SQL queries with the programmatic data manipulations supported by RDDs in Python, Java, and Scala, all within a single application, thus combining SQL with complex analytics. This tight integration with the rich computing environment provided by Spark makes Spark SQL unlike any other open-source data warehouse tool.

Spark Streaming:

Spark Streaming is a Spark component that enables processing of live streams of data. Spark Streaming provides an API for manipulating data streams that closely matches the Spark Core's RDD API, making it easy for programmers to learn the project and move between applications that manipulate data stored in memory, on disk, or arriving in real time. Underneath its API, Spark Streaming was designed to provide the same degree of fault tolerance, throughput, and scalability as Spark Core.

MLlib:

Spark comes with a library containing common machine learning (ML) functionality, called MLlib. MLlib provides multiple types of machine learning algorithms, including classification, regression, clustering, and

collaborative filtering, as well as supporting functionality such as model evaluation and data import. It also provides some lower-level ML primitives, including a generic gradient descent optimization algorithm. All of these methods are designed to scale out across a cluster.

GraphX:

GraphX is a library for manipulating graphs (e.g., a social network's friend graph) and performing graph-parallel computations. Like Spark Streaming and Spark SQL, GraphX extends the Spark RDD API, allowing us to create a directed graph with arbitrary properties attached to each vertex and edge. GraphX also provides various operators for manipulating graphs

Cluster Managers:

Under the hood, Spark is designed to efficiently scale up from one-to-many thousands of compute nodes. To achieve this while maximizing flexibility, Spark can run over a variety of cluster managers, including Hadoop YARN, Apache Mesos, and a simple cluster manager included in Spark itself called the Standalone Scheduler. If you are just installing Spark on an empty set of machines, the Standalone Scheduler provides an easy way to get started; if you already have a Hadoop YARN or Mesos cluster, however, Spark's support for these cluster managers allows your applications to also run on them.

Uses of Spark:

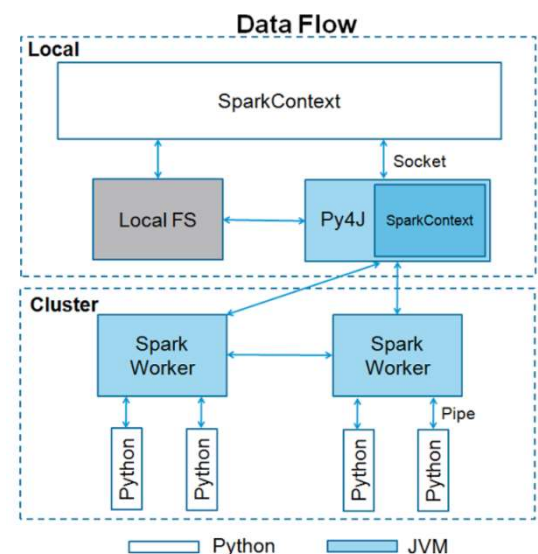
Because Spark is a general-purpose framework for cluster computing, it is used for a diverse range of applications. In the Preface we outlined two groups of readers that this book targets: data scientists and

engineers. Let's take a closer look at each group and how it uses Spark. Unsurprisingly, the typical use cases differ between the two, but we can roughly classify them into two categories, data science and data applications.

PySpark:

Apache Spark is written in Scala programming language. To support Python with Spark, Apache Spark Community released a tool, PySpark. Using PySpark, you can work with RDDs in Python programming language also. It is because of a library called Py4j that they are able to achieve this. PySpark offers PySpark Shell which links the Python API to the spark core and initializes the Spark context. Majority of data scientists and analytics experts today use Python because of its rich library set. Integrating Python with Spark is a boon to them.

SparkContext is the entry point to any spark functionality. When we run any Spark application, a driver program starts, which has the main function and your SparkContext gets initiated here. The driver program then runs the operations inside the executors on worker nodes. SparkContext uses Py4J to launch a JVM and creates a JavaSparkContext. By default, PySpark has SparkContext available as 'sc', so creating a new SparkContext won't work.



Following are the parameters of a SparkContext.

- **Master** – It is the URL of the cluster it connects to.
- **appName** – Name of your job.

- **sparkHome** – Spark installation directory.
- **pyFiles** – The .zip or .py files to send to the cluster and add to the PYTHONPATH.
- **Environment** – Worker nodes environment variables.
- **batchSize** – The number of Python objects represented as a single Java object. Set 1 to disable batching, 0 to automatically choose the batch size based on object sizes, or -1 to use an unlimited batch size.
- **Serializer** – RDD serializer.
- **Conf** – An object of L{SparkConf} to set all the Spark properties.
- **Gateway** – Use an existing gateway and JVM, otherwise initializing a new JVM.
- **JSC** – The JavaSparkContext instance.

profiler_cls – A class of custom Profiler used to do profiling.

Pyspark ML Libraries:

Apache Spark offers a Machine Learning API called **MLlib**. PySpark has this machine learning API in Python as well. It supports different kind of algorithms, which are mentioned below –

- **mllib.classification** – The **spark.mllib** package supports various methods for binary classification, multiclass classification and regression analysis. Some of the most popular algorithms in classification are **Random Forest**, **Naive Bayes**, **Decision Tree**, etc.



- **mllib.clustering** – Clustering is an unsupervised learning problem, whereby you aim to group subsets of entities with one another based on some notion of similarity.
- **mllib.fpm** – Frequent pattern matching is mining frequent items, itemsets, subsequences or other substructures that are usually among the first steps to analyze a large-scale dataset. This has been an active research topic in data mining for years.
- **mllib.linalg** – MLib utilities for linear algebra.
- **mllib.recommendation** – Collaborative filtering is commonly used for recommender systems. These techniques aim to fill in the missing entries of a user item association matrix.
- **spark.mllib** – It currently supports model-based collaborative filtering, in which users and products are described by a small set of latent factors that can be used to predict missing entries. spark.mllib uses the Alternating Least Squares (ALS) algorithm to learn these latent factors.
- **mllib.regression** – Linear regression belongs to the family of regression algorithms. The goal of regression is to find relationships and dependencies between variables. The interface for working with linear regression models and model summaries is similar to the logistic regression case.

There are other algorithms, classes and functions also as a part of the mllib package. As of now, let us understand a demonstration on **pyspark.mllib**.

Advantages of PySpark:

Following are the benefits of using PySpark. Let's talk about them in detail

In-Memory Computation in Spark: With in-memory processing, it helps you increase the speed of processing. And the best part is that the data is being cached, allowing you not to fetch data from the disk every time thus the time is saved. For those who don't know, PySpark has DAG execution engine that helps facilitate in-memory computation and acyclic data flow that would ultimately result in high speed.

Swift Processing: When you use PySpark, you will likely to get high data processing speed of about 10x faster on the disk and 100x faster in memory. By reducing the number of read-write to disk, this would be possible.

Dynamic in Nature: Being dynamic in nature, it helps you to develop a parallel application, as Spark provides 80 high-level operators.

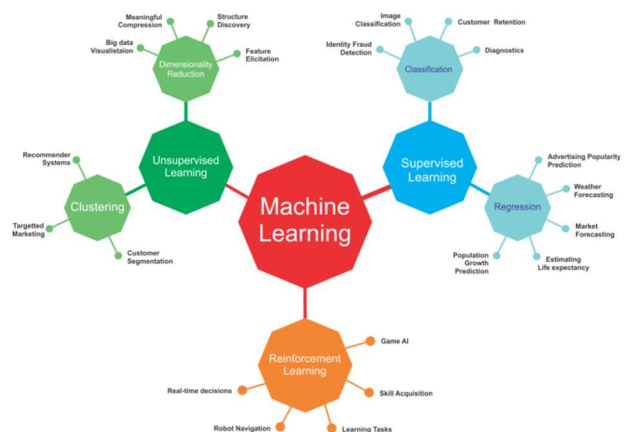
Fault Tolerance in Spark: Through Spark abstraction-RDD, PySpark provides fault tolerance. The programming language is specifically designed to handle the malfunction of any worker node in the cluster, ensuring that the loss of data is reduced to zero.

Real-Time Stream Processing: PySpark is renowned and much better than other languages when it comes to real-time stream processing. Earlier the problem with Hadoop MapReduce was that it can manage the data which is already present, but not the real-time data. However, with PySpark Streaming, this problem is reduced significantly.

Data scientists and other Data Analyst professionals will benefit from the distributed processing power of PySpark. And with PySpark, the best part is that the workflow for accomplishing this becomes incredibly simple like never before. By using PySpark, data scientists can build an analytical application in Python and can aggregate and transform the data, then bring the consolidated data back. There is no arguing with the fact that PySpark would be used for the creation and evaluation stages. However, things get tangled a bit when it comes to drawing a heat map to show how well the model predicted people's preferences. PySpark can significantly accelerate analysis by making it easy to combine local and distributed data transformation operations while keeping control of computing costs. In addition, the language helps data scientists to avoid always having to down sample large sets of data.

Machine Learning:

Machine learning (ML) is the study of computer algorithms that improve automatically through experience and by the use of data. It is seen as a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks. A subset of machine learning is closely related to computational statistics,



which focuses on making predictions using computers; but not all machine learning is statistical learning. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a related field of study, focusing on exploratory data analysis through unsupervised learning. In its application across business problems, machine learning is also referred to as predictive analytics.

Supervised Learning:

Supervised learning algorithms build a mathematical model of a set of data that contains both the inputs and the desired outputs. The data is known as training data, and consists of a set of training examples. Each training example has one or more inputs and the desired output, also known as a supervisory signal. In the mathematical model, each training example is represented by an array or vector, sometimes called a feature vector, and the training data is represented by a matrix.

Through iterative optimization of an objective function, supervised learning algorithms learn a function that can be used to predict the output associated with new inputs. An optimal function will allow the algorithm to correctly determine the output for inputs that were not a part of the training data. An algorithm that improves the accuracy of its outputs or predictions over time is said to have learned to perform that task.

Logistic Regression:

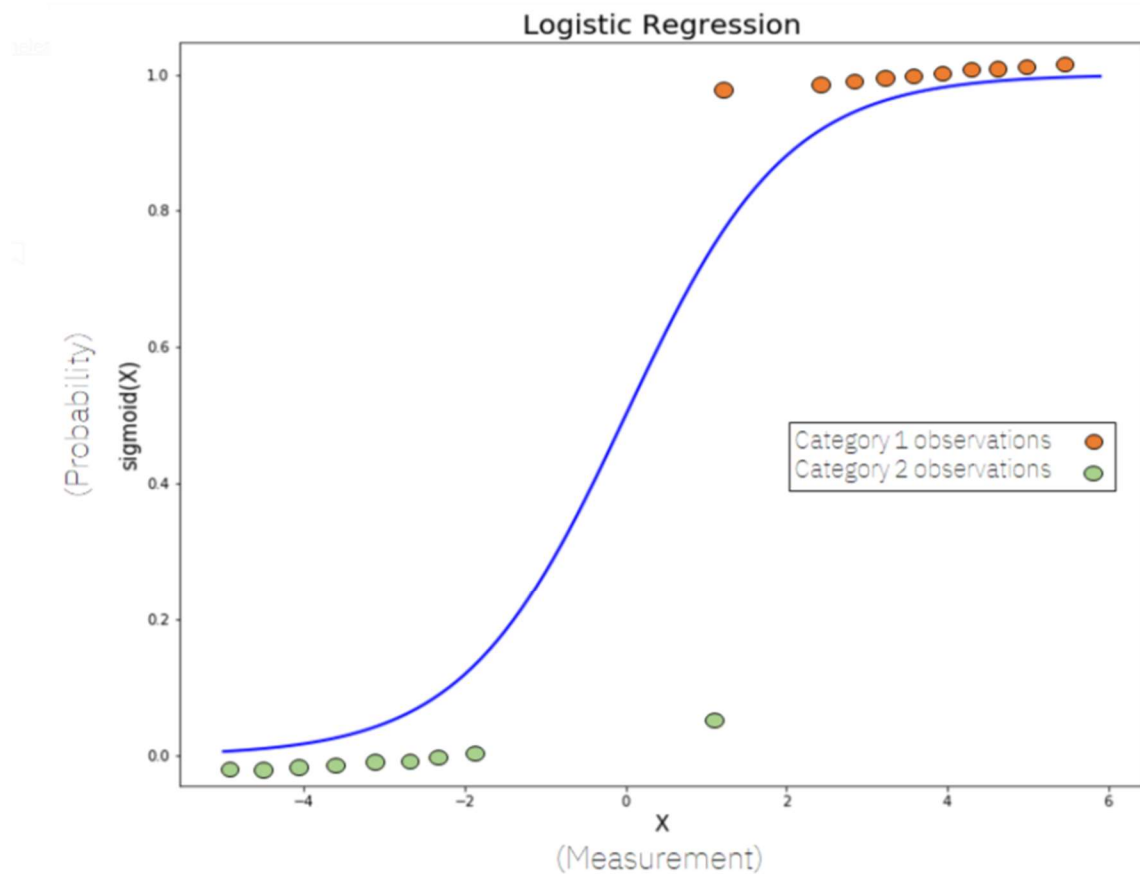
In the Machine Learning world, Logistic Regression is a kind of parametric classification model, despite having the word 'regression' in its name.

This means that logistic regression models are models that have a certain fixed number of parameters that depend on the number of input features, and they output categorical prediction, like for example if a plant belongs to a certain species or not.

In reality, the theory behind Logistic Regression is very similar to the one from Linear Regression. In Logistic Regression, we don't directly fit a straight line to our data like in linear regression. Instead, we fit a S shaped curve, called Sigmoid, to our observations.

It is a class of regression where the independent variable is used to predict the dependent variable. When the dependent variable has two categories, then it is a binary logistic regression. When the dependent variable has more than two categories, then it is a multinomial logistic regression. When the dependent variable category is to be ranked, then it is an ordinal logistic regression (OLS). To obtain the maximum likelihood estimation, transform the dependent variable in the logit function. Logit is basically a natural log of the dependent variable and tells whether or not the event will occur. Ordinal logistic regression does not assume a linear relationship between the dependent and independent variable. It does not assume homoscedasticity. Wald statistics tests the significance of the individual independent variable.

In many ways, logistic regression and linear regression are similar. However, the difference lies in what they're used for. Linear regression algorithms are used for predicting or forecasting values but logistic regression is used in classification tasks.



Let's examine this figure closely.

First of all, like we said before, Logistic Regression models are classification models; specifically binary classification models (they can only be used to distinguish between 2 different categories — like if a person is obese or not given its weight, or if a house is big or small given its size). This means that our data has two kinds of observations (Category 1 and Category 2 observations) like we can observe in the figure.

Note: *This is a very simple example of Logistic Regression, in practice much harder problems can be solved using these models, using a wide range of features and not just a single one.*

Secondly, as we can see, the Y-axis goes from 0 to 1. This is because the sigmoid function always takes as maximum and minimum these two values, and this fits very well our goal of classifying samples in two different categories. By computing the sigmoid function of X (that is a weighted sum of the input features, just like in Linear Regression), we get a probability (between 0 and 1 obviously) of an observation belonging to one of the two categories.

The formula for the sigmoid function is the following:

$$\textit{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

There are multiple ways to train a Logistic Regression model (fit the S shaped line to our data). We can use several iterative optimisation algorithms like Gradient Descent to calculate the parameters of the model (the weights) or we can use probabilistic methods like Maximum likelihood (which are not required for our project).

Logistic regression is one of the simplest Machine Learning models. They are easy to understand, interpretable, and can give pretty good results.

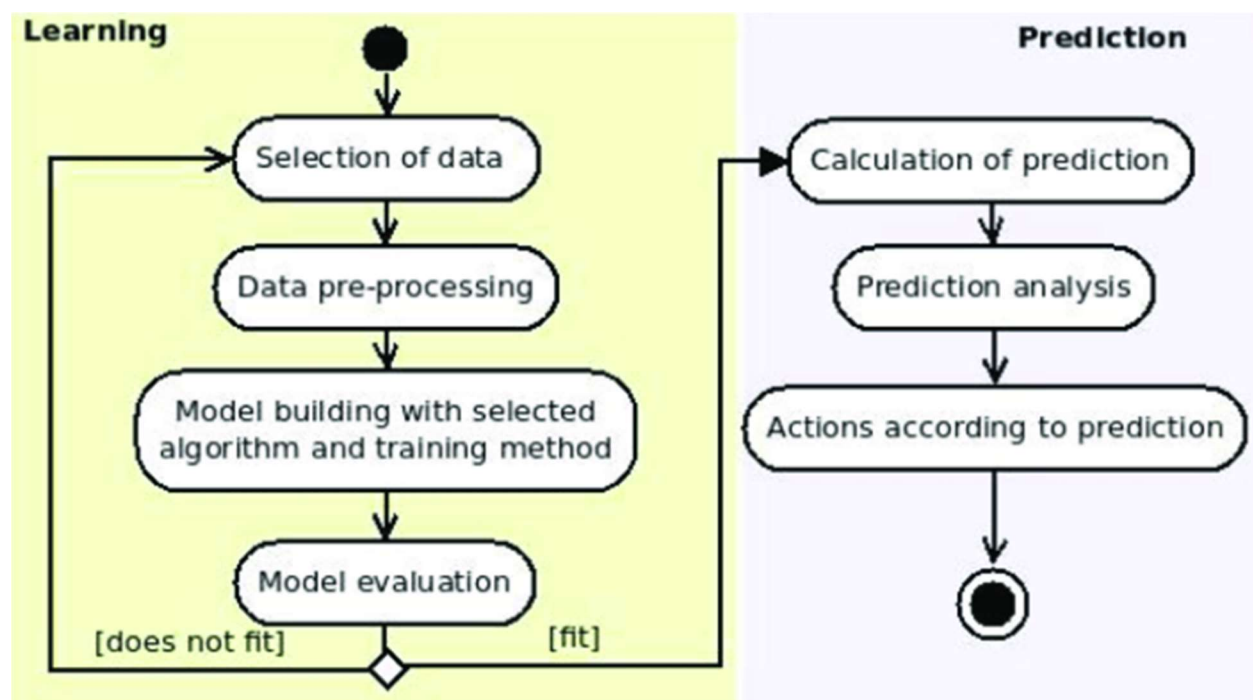
Applications and Advantages of Logistic Regression:

- Logistic regression is used in various fields, including machine learning, most medical fields, and social sciences. For example, the Trauma and Injury Severity Score (TRISS), which is widely used to predict mortality in injured patients, was originally developed by Boyd et al. using logistic regression.
- Logistic regression may be used to predict the risk of developing a given disease (e.g., diabetes; coronary heart disease), based on observed characteristics of the patient
- The technique can also be used in engineering, especially for predicting the probability of failure of a given process, system or product. It is also used in marketing applications such as prediction of a customer's propensity to purchase a product or halt a subscription, etc.
- Logistic regression is easier to implement, interpret, and very efficient to train. It makes no assumptions about distributions of classes in feature space.
- It can easily extend to multiple classes (multinomial regression) and a natural probabilistic view of class predictions. It not only provides a measure of how appropriate a predictor (coefficient size) is, but also its direction of association (positive or negative).
- It can interpret model coefficients as indicators of feature importance.

Prediction System:

“Prediction” refers to the output of an algorithm after it has been trained on a historical dataset and applied to new data when forecasting the likelihood of a particular outcome, such as whether or not a customer will churn in 30 days. The algorithm will generate probable values for an unknown variable for each record in the new data, allowing the model builder to identify what that value will most likely be.

Predictions are important because, Machine learning model predictions allow businesses to make highly accurate guesses as to the likely outcomes of a question based on historical data which can be about all kinds of things – customer churn likelihood, possible fraudulent activity, and more. These provide the business with insights that result in tangible business value. For example, if a model predicts a customer is likely to churn, the business can target them with specific communications and outreach that will prevent the loss of that customer.



SYSTEM DESIGN

In the previous section, we have gone through Spark and PySpark in detail. Now we have a look at our project design.

We have used Pipeline structure for preparing the data for training. A pipeline consists of a series of transformer and estimator stages that typically prepare a Data Frame for modelling and then train a predictive model. In this case, we will create a pipeline with six stages namely:

- A StringIndexer estimator that converts string values to indexes for categorical features
- A VectorAssembler that combines categorical features into a single vector
- A VectorIndexer that creates indexes for a vector of categorical features
- A VectorAssembler that creates a vector of continuous numeric features
- A MinMaxScaler that normalizes continuous numeric features
- A VectorAssembler that creates a vector of categorical and continuous features

Before moving on with these, we must first import some necessary dependencies such as **pyspark.sql.types**, **pyspark.sql.functions**, **pyspark.ml**.

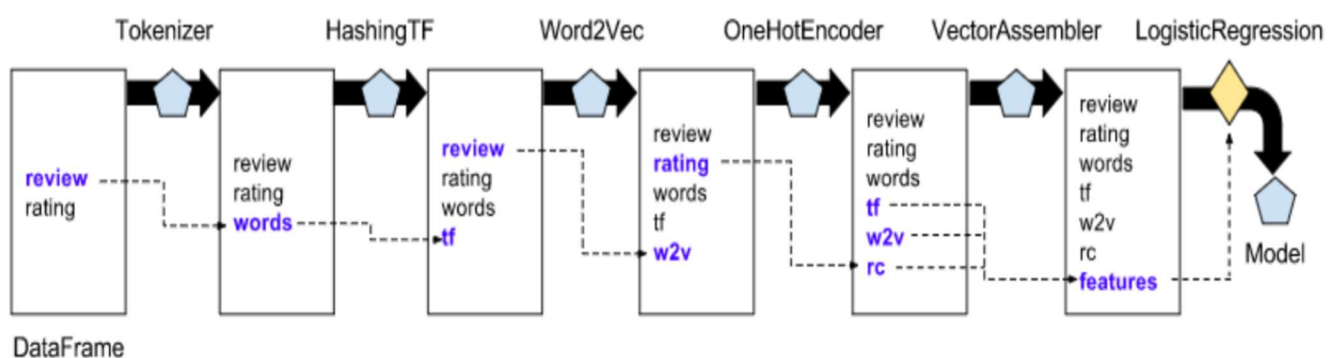
PySpark SQL is a Spark module for structured data processing. It provides a programming abstraction called Data Frames and can also act as a distributed SQL query engine. It enables unmodified Hadoop Hive queries to run up to 100x faster on existing deployments and data.

PySpark SQL Types class is a base class of all data types in PySpark which defined in a package “pyspark.sql.types.” Data Type and they are used to create Data Frame with a specific type.

PySpark MLlib is a machine-learning library. It is a wrapper over PySpark Core to do data analysis using machine-learning algorithms. It works on distributed systems and is scalable. We can find implementations of classification, clustering, linear regression, and other machine-learning algorithms in PySpark MLlib.

ML Pipelines provide a uniform set of high-level APIs built on top of Data Frames that help users create and tune practical machine learning pipelines. We can find the **LogisticRegression** module under the library named **pyspark.ml.classification**.

Below is clear picture of how Spark Pipelines work:



In this, we are not going to use Tokenizer, Hashing and One hot encoder.

IMPLEMENTATION

We basically divide our implementation part into 5 steps as below:

- Importing Dependencies
- Data Visualization
- Encoding and Processing
- Training and Testing the data using Logistic Regression
- Evaluation of the model

1.) As discussed previously, we import the required libraries such as **pyspark.sql.types**, **pyspark.sql.functions**, **pyspark.ml** to perform the necessary actions. We even have to initialize a Spark session so that, it sets the Spark master URL to connect to, such as "local" to run locally, "local[4]" to run locally with 4 cores, or "spark://master:7077" to run on a Spark standalone cluster.

```
from pyspark.sql.types import *
from pyspark.sql.functions import *
from pyspark.sql import SparkSession

from pyspark.ml import Pipeline
#from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.feature import VectorAssembler, StringIndexer, VectorIndexer, MinMaxScaler
from pyspark.ml.classification import LogisticRegression

spark = SparkSession.builder.master("local[*]").getOrCreate()
```


We load our data set “**flights.csv**” and store it under a variable called data. Below is the picture of the Spark Data frame representation of our data set:

DayofMonth	DayOfWeek	Carrier	OriginAirportID	DestAirportID	DepDelay	ArrDelay
19	5	DL	11433	13303	-3	1
19	5	DL	14869	12478	0	-8
19	5	DL	14057	14869	-4	-15
19	5	DL	15016	11433	28	24
19	5	DL	11193	12892	-6	-11
19	5	DL	10397	15016	-1	-19
19	5	DL	15016	10397	0	-1
19	5	DL	10397	14869	15	24
19	5	DL	10397	10423	33	34
19	5	DL	11278	10397	323	322

only showing top 10 rows

2.) We perform data visualization on our data set to get various insights of the categorical features in our columns:

```
from matplotlib import cm
fig = plt.figure(figsize=(25,15)) ## Plot Size
st = fig.suptitle("Distribution of Features", fontsize=50,
                  verticalalignment='center') # Plot Main Title
```



```

for col,num in zip(data.toPandas().describe().columns, range(1,7)):
    ax = fig.add_subplot(3,4,num)
    ax.hist(data.toPandas()[col])
    plt.style.use('dark_background')
    plt.grid(False)
    plt.xticks(rotation=45,fontsize=20)
    plt.yticks(fontsize=15)
    plt.title(col.upper(),fontsize=20)
plt.tight_layout()
st.set_y(0.95)
fig.subplots_adjust(top=0.85,hspace = 0.4)
plt.show()

```

3.) Encoding and Pre – processing section includes, converting the text data into numerical format. Since machine learning or deep learning models require input to be feature matrices or numerical values and currently, we still have our data in character or string format. So, the next step is to encode these characters. Moreover, we even convert our Arrival delay column into 0's and 1's – if the flight delay is over 15 minutes, we take it as 1 and if it is minimal (i.e., lesser than 15), we take the number to be 0. As we know that in Logistic Regression, the output would be binary in nature, i.e., either 0 or 1. Later we then convert this format into our desired string format.

Therefore, this step plays a crucial role in generating the output.

In pre – processing, we change our target variable's column to another simple and understandable name – “True Label”.

This will be very helpful in comparing both the predicted quantities and the actual delay in arrival.


```
data = csv.select("DayofMonth", "DayOfWeek", "Carrier", "OriginAirportID", "DestAirportID", "DepDelay",
((col("ArrDelay") > 15).cast("Int").alias("label")))
data.show(10)
```

4.) Next step is to train and test our model using Logistic Regression ML algorithm. Before we fit our model for training, we have to split our data set for training and testing the model. We have divided the data as 70 % training and 30 % testing:

```
splits = data.randomSplit([0.7, 0.3])
print(splits)
train = splits[0]
test = splits[1].withColumnRenamed("label", "trueLabel")
train_rows = train.count()
test_rows = test.count()
print("Training Rows:", train_rows, " Testing Rows:", test_rows)
```

We have to define a pipeline for our model now:

```
strIdx = StringIndexer(inputCol = "Carrier", outputCol = "CarrierIdx")
catVect = VectorAssembler(inputCols = ["CarrierIdx", "DayofMonth", "DayOfWeek", "OriginAirportID",
"DestAirportID"], outputCol="catFeatures")
catIdx = VectorIndexer(inputCol = catVect.getOutputCol(), outputCol = "idxCatFeatures")
numVect = VectorAssembler(inputCols = ["DepDelay"], outputCol="numFeatures")
minMax = MinMaxScaler(inputCol = numVect.getOutputCol(), outputCol="normFeatures")
featVect = VectorAssembler(inputCols=["idxCatFeatures", "normFeatures"], outputCol="features")
lr = LogisticRegression(labelCol="label", featuresCol="features", maxIter=10, regParam=0.3)
pipeline = Pipeline(stages=[strIdx, catVect, catIdx, numVect, minMax, featVect, lr])
```

We then run the pipeline model and fit our training data into it, by creating an instance from it and storing it in a variable called **pipelineModel**:

```
pipelineModel = pipeline.fit(train)
prediction = pipelineModel.transform(test)
predicted = prediction.select("features", "prediction", "trueLabel")
predicted.show(100, truncate=False)
```

5.) Last but not the least is the model evaluation step. This is an important step as to understand the performance of our trained model on testing data. We two performance metrics such as **Accuracy** and **Precision**. We will also have a look at the Confusion Matrix to get a better understanding of the True positive and True Negative values.

```
from sklearn.metrics import accuracy_score, precision_score
print("Confusion matrix for predictions on flight delay\n")
print(pd.crosstab(pd.Series(y_true, name='Actual'), pd.Series(y_pred, name='Predicted')))
```

After generating the confusion matrix, we then proceed on to finding the Accuracy and Precision for understanding our good our model was in predicting the output, as compared to the original one.

Accuracy in Machine Learning is very important because, industrial companies use machine learning models to make practical business decisions, and more accurate model outcomes result in better decisions. ... The benefits of improving model accuracy help avoid considerable time, money, and undue stress.


```
tp = float(predicted.filter("prediction == 1.0 AND truelabel == 1").count())
fp = float(predicted.filter("prediction == 1.0 AND truelabel == 0").count())
tn = float(predicted.filter("prediction == 0.0 AND truelabel == 0").count())
fn = float(predicted.filter("prediction == 0.0 AND truelabel == 1").count())
accuracy = (tp + tn) / (tp + tn + fp + fn)
pr = tp / (tp + fp)
metrics = spark.createDataFrame([
    ("TP", tp),
    ("FP", fp),
    ("TN", tn),
    ("FN", fn),
    ("Accuracy", accuracy),
    ("Precision", pr),
], ["metric", "value"])
metrics.show()
```

True Positives (TP):

Cases in which we predicted yes (the flight has been delayed), and the actual also says the flight has been delayed.

True Negatives (TN):

Cases where, we predicted no, and the actual output also says that the flight has not been delayed.

False Positives (FP):

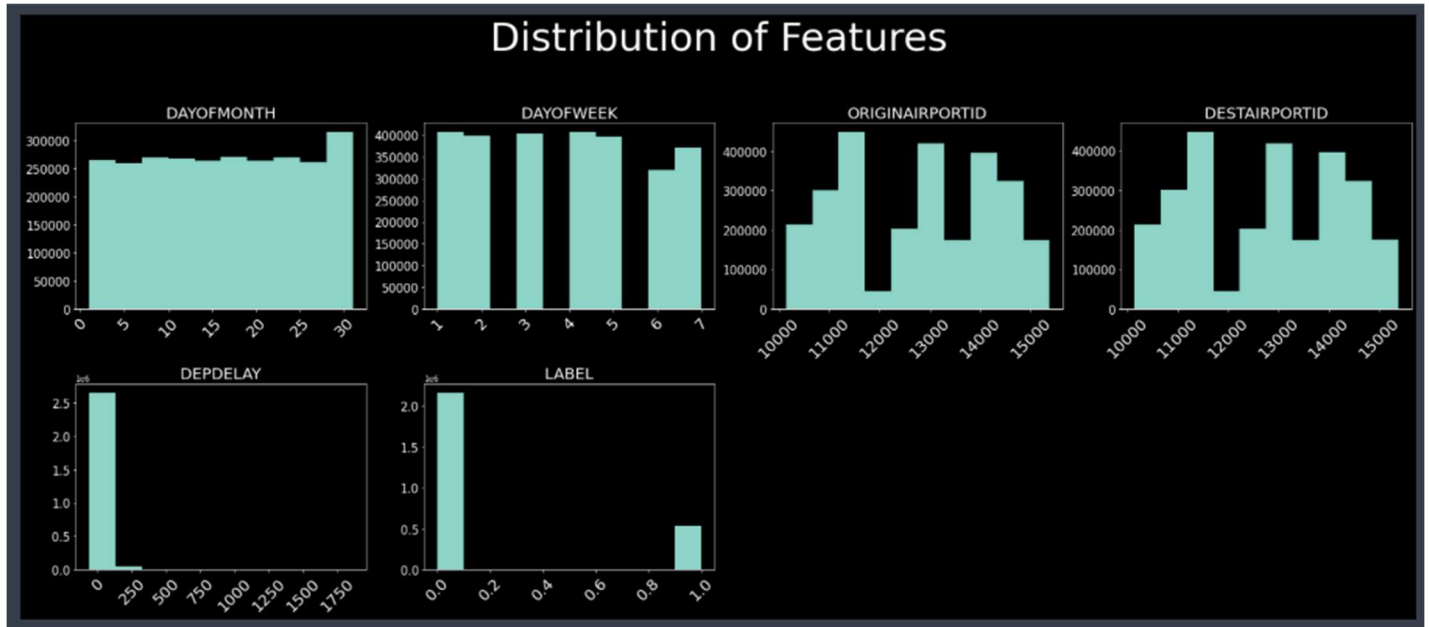
Cases where, we predicted yes, but there hasn't been a delay in the flight's arrival.

False Negatives (FN):

Cases in which we predicted no, but there has actually been a delay in the flight arrival.

RESULTS

1.) Let's have a look at the data visualization part. Below is the detailed distribution of features (all the columns of the data set):



2.) Predicted Output – Our model predicted a binary output for the testing data that we have passed in:

features	prediction	trueLabel
[10.0,1.0,0.0,10397.0,10693.0,0.030161206448257934]	0.0	0
[10.0,1.0,0.0,10397.0,12264.0,0.0405616224648986]	0.0	0
[10.0,1.0,0.0,10397.0,13851.0,0.030161206448257934]	0.0	0
[10.0,1.0,0.0,10423.0,11433.0,0.028601144045761834]	0.0	0
[10.0,1.0,0.0,10423.0,13244.0,0.0280811232449298]	0.0	0
[10.0,1.0,0.0,10423.0,13487.0,0.026001040041601666]	0.0	0
[10.0,1.0,0.0,10423.0,13487.0,0.027561102444097766]	0.0	0
[10.0,1.0,0.0,10423.0,14869.0,0.04888195527821113]	0.0	1
[10.0,1.0,0.0,10529.0,11193.0,0.028601144045761834]	0.0	0
[10.0,1.0,0.0,10529.0,11193.0,0.0358814352574103]	0.0	0
[10.0,1.0,0.0,10529.0,11433.0,0.028601144045761834]	0.0	0
[10.0,1.0,0.0,10693.0,10397.0,0.028601144045761834]	0.0	0
[10.0,1.0,0.0,10693.0,11193.0,0.0296411856474259]	0.0	0

3.) As discussed previously, we are going to evaluate our model based on its performance, using some of the performance metrics such as **Accuracy** and **Precision**. Below is the output result for that.

Confusion matrix for predictions on flight delay

Predicted	(0.0,)	(1.0,)
Actual		
(0,)	649545	86
(1,)	142508	18723

- Accuracy is calculated as $(TP + TN) / (TP + TN + FP + FN)$
- Precision is calculated as $(TP) / (TP + FP)$

+-----+-----+	
metric	value
+-----+-----+	
TP	18723.0
FP	86.0
TN	649545.0
FN	142508.0
Accuracy	82.41451689683325
Precision	99.54277207719709
+-----+-----+	

CONCLUSION

We have successfully implemented and executed the flight delay prediction model using PySpark. We have explained about Pipeline structure in Spark and have used in building our model. With this, we can clearly say that using PySpark for bigger data sets is very much efficient rather than using normal scikit-learn modules for data processing and predictions as, it would consume less time and generates the result in a much-organized manner. Our model has showcased excellent precision value compared to accuracy (yet a decent one).

The future enhancements for our project would be tune the hyperparameters for a better performance of the model, mainly to improve the accuracy by using **BinaryClassifierEvaluator()** function.

To the best of our knowledge, we have explained about the applications of PySpark and its use in building Machine Learning Models. Finally, we conclude by saying that we as a team understood the importance of PySpark and its applications in dealing large datasets, and the various machine learning models building through it.