

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE:

Enero - Junio 2020

CARRERA:

Ing. Tecnologías de la Información y Comunicaciones

MATERIA:

Datos Masivos

UNIDAD A EVALUAR:

Unidad 2

NOMBRE Y NÚMERO DE CONTROL DEL ALUMNO:

Diaz Martinez Ruben Emilio # 15210791

Flores Acosta Alfredo # 15210331

NOMBRE DEL MAESTRO (A):

Dr. Christian Romero Hernandez.

MLlib standardizes APIs for machine learning algorithms to make it easier to combine multiple algorithms into a single pipeline, or workflow. This section covers the key concepts introduced by the Pipelines API, where the pipeline concept is mostly inspired by the scikit-learn project.

DataFrame: This ML API uses DataFrame from Spark SQL as an ML dataset, which can hold a variety of data types. E.g., a DataFrame could have different columns storing text, feature vectors, true labels, and predictions.

Transformer: A Transformer is an algorithm which can transform one DataFrame into another DataFrame. E.g., an ML model is a Transformer which transforms a DataFrame with features into a DataFrame with predictions.

Estimator: An Estimator is an algorithm which can be fit on a DataFrame to produce a Transformer. E.g., a learning algorithm is an Estimator which trains on a DataFrame and produces a model.

Pipeline: A Pipeline chains multiple Transformers and Estimators together to specify an ML workflow.

Parameter: All Transformers and Estimators now share a common API for specifying parameters.

DataFrame

Machine learning can be applied to a wide variety of data types, such as vectors, text, images, and structured data. This API adopts the DataFrame from Spark SQL in order to support a variety of data types.

DataFrame supports many basic and structured types; see the Spark SQL datatype reference for a list of supported types. In addition to the types listed in the Spark SQL guide, DataFrame can use ML Vector types.

A `DataFrame` can be created either implicitly or explicitly from a regular `RDD`. See the code examples below and the Spark SQL programming guide for examples.

Columns in a `DataFrame` are named. The code examples below use names such as “text,” “features,” and “label.” Pipeline components Transformers

A `Transformer` is an abstraction that includes feature transformers and learned models. Technically, a `Transformer` implements a method `transform()`, which converts one `DataFrame` into another, generally by appending one or more columns. For example:

A feature transformer might take a `DataFrame`, read a column (e.g., text), map it into a new column (e.g., feature vectors), and output a new `DataFrame` with the mapped column appended.

A learning model might take a `DataFrame`, read the column containing feature vectors, predict the label for each feature vector, and output a new `DataFrame` with predicted labels appended as a column.

Estimators

An `Estimator` abstracts the concept of a learning algorithm or any algorithm that fits or trains on data. Technically, an `Estimator` implements a method `fit()`, which accepts a `DataFrame` and produces a `Model`, which is a `Transformer`. For example, a learning algorithm such as `LogisticRegression` is an `Estimator`, and calling `fit()` trains a `LogisticRegressionModel`, which is a `Model` and hence a `Transformer`. Properties of pipeline components

`Transformer.transform()`s and `Estimator.fit()`s are both stateless. In the future, stateful algorithms may be supported via alternative concepts.

Each instance of a `Transformer` or `Estimator` has a unique ID, which is useful in specifying parameters (discussed below). Pipeline

In machine learning, it is common to run a sequence of algorithms to process and learn from data. E.g., a simple text document processing workflow might include several stages:

Split each document’s text into words.

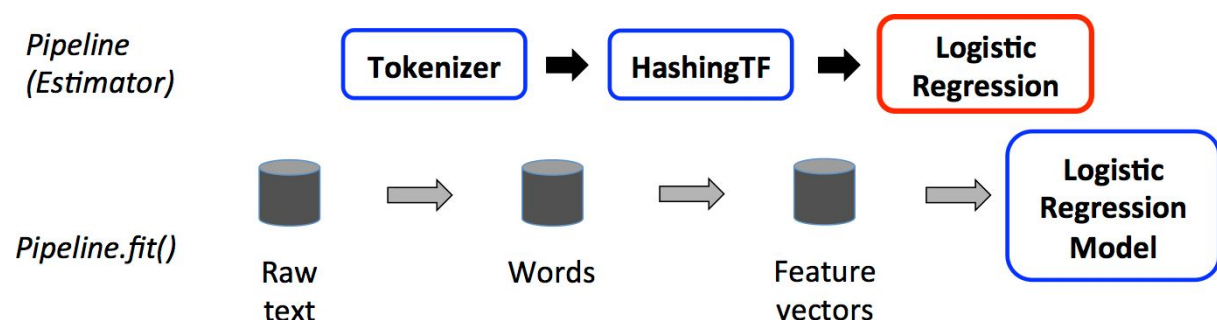
Convert each document’s words into a numerical feature vector.

Learn a prediction model using the feature vectors and labels.

MLlib represents such a workflow as a Pipeline, which consists of a sequence of PipelineStages (Transformers and Estimators) to be run in a specific order. We will use this simple workflow as a running example in this section. How it works

A Pipeline is specified as a sequence of stages, and each stage is either a Transformer or an Estimator. These stages are run in order, and the input DataFrame is transformed as it passes through each stage. For Transformer stages, the transform() method is called on the DataFrame. For Estimator stages, the fit() method is called to produce a Transformer (which becomes part of the PipelineModel, or fitted Pipeline), and that Transformer's transform() method is called on the DataFrame.

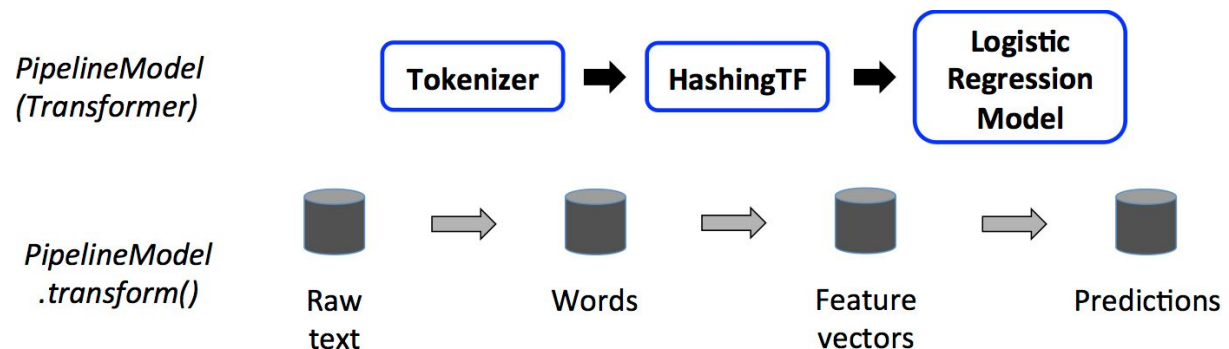
We illustrate this for the simple text document workflow. The figure below is for the training time usage of a Pipeline.



ML Pipeline Example

Above, the top row represents a Pipeline with three stages. The first two (Tokenizer and HashingTF) are Transformers (blue), and the third (LogisticRegression) is an Estimator (red). The bottom row represents data flowing through the pipeline, where cylinders indicate DataFrames. The Pipeline.fit() method is called on the original DataFrame, which has raw text documents and labels. The Tokenizer.transform() method splits the raw text documents into words, adding a new column with words to the DataFrame. The HashingTF.transform() method converts the words column into feature vectors, adding a new column with those vectors to the DataFrame. Now, since LogisticRegression is an Estimator, the Pipeline first calls LogisticRegression.fit() to produce a LogisticRegressionModel. If the Pipeline had more Estimators, it would call the LogisticRegressionModel's transform() method on the DataFrame before passing the DataFrame to the next stage.

A Pipeline is an Estimator. Thus, after a Pipeline's `fit()` method runs, it produces a `PipelineModel`, which is a Transformer. This `PipelineModel` is used at test time; the figure below illustrates this usage.



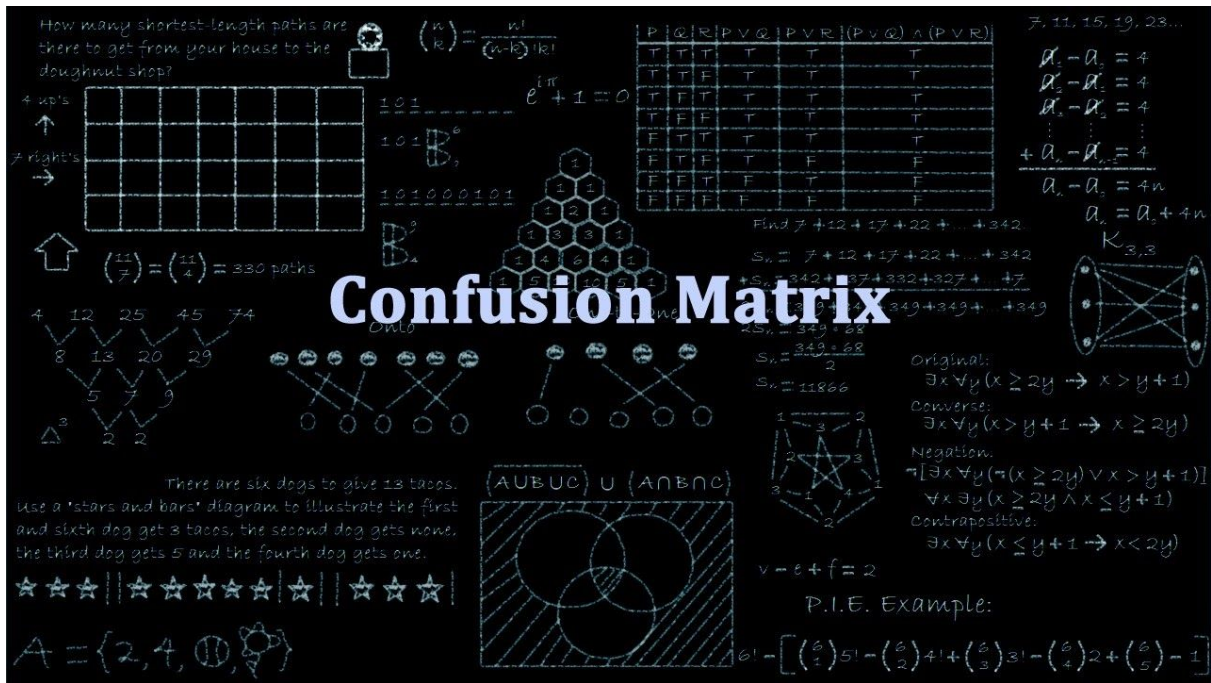
ML PipelineModel Example

In the figure above, the `PipelineModel` has the same number of stages as the original Pipeline, but all Estimators in the original Pipeline have become Transformers. When the `PipelineModel`'s `transform()` method is called on a test dataset, the data are passed through the fitted pipeline in order. Each stage's `transform()` method updates the dataset and passes it to the next stage.

Pipelines and `PipelineModels` help to ensure that training and test data go through identical feature processing steps.

Understanding Confusion Matrix





When we get the data, after data cleaning, pre-processing and wrangling, the first step we do is to feed it to an outstanding model and of course, get output in probabilities. But hold on! How in the hell can we measure the effectiveness of our model. Better the effectiveness, better the performance and that's exactly what we want. And it is where the Confusion matrix comes into the limelight. Confusion Matrix is a performance measurement for machine learning classification.



		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

It is extremely useful for measuring Recall, Precision, Specificity, Accuracy and most importantly AUC-ROC Curve.

Let's understand TP, FP, FN, TN in terms of pregnancy analogy.

		Actual Values	
		1	0
Predicted Values	1	TRUE POSITIVE 	FALSE POSITIVE  TYPE 1 ERROR
	0	FALSE NEGATIVE  TYPE 2 ERROR	TRUE NEGATIVE 

True Positive:

Interpretation: You predicted positive and it's true.

You predicted that a woman is pregnant and she actually is.

True Negative:

Interpretation: You predicted negative and it's true.

You predicted that a man is not pregnant and he actually is not.

False Positive: (Type 1 Error)

Interpretation: You predicted positive and it's false.

You predicted that a man is pregnant but he actually is not.

False Negative: (Type 2 Error)

Interpretation: You predicted negative and it's false.

You predicted that a woman is not pregnant but she actually is.

Just Remember, We describe predicted values as Positive and Negative and actual values as True and False.