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

NOMBRE DEL MAESTRO (A):
Dr. Christian Romero Hernandez.

VectorAssembler Library

VectorAssembler is a transformer that combines a given list of columns into a single vector column. It is useful for combining raw features and features generated by different feature transformers into a single feature vector, in order to train ML models like logistic regression and decision trees. VectorAssembler accepts the following input column types: all numeric types, boolean type, and vector type. In each row, the values of the input columns will be concatenated into a vector in the specified order. Examples

Assume that we have a DataFrame with the columns id, hour, mobile, userFeatures, and clicked:

id	hour	mobile	userFeatures	clicked
0	18	1.0	[0.0, 10.0, 0.5]	1.0

userFeatures is a vector column that contains three user features. We want to combine hour, mobile, and userFeatures into a single feature vector called features and use it to predict clicked or not. If we set VectorAssembler's input columns to hour, mobile, and userFeatures and output column to features, after transformation we should get the following DataFrame:

id	hour	mobile	userFeatures	clicked	features
0	18	1.0	[0.0, 10.0, 0.5]	1.0	[18.0, 1.0, 0.0, 10.0, 0.5]

Vectors Library

Factory methods for org.apache.spark.ml.linalg.Vector. We don't use the name Vector because Scala imports scala.collection.immutable.Vector by default.

What does RMSE really mean?

These errors, thought of as random variables, might have Gaussian distribution with mean μ and standard deviation σ, but any other distribution with a square-integrable PDF (probability density function) would also work. We want to think of $\hat{y}_i$ as an underlying physical quantity, such as the exact distance from Mars to the Sun at a particular point in time. Our observed quantity y_i would then be the distance from

Mars to the Sun as we measure it, with some errors coming from mis-calibration of our telescopes and measurement noise from atmospheric interference.

RMSE in Data Science: Subtleties of Using RMSE

In data science, RMSE has a double purpose:

To serve as a heuristic for training models

To evaluate trained models for usefulness / accuracy

This raises an important question: What does it mean for RMSE to be “small”?

We should note first and foremost that “small” will depend on our choice of units, and on the specific application we are hoping for. 100 inches is a big error in a building design, but 100 nanometers is not. On the other hand, 100 nanometers is a small error in fabricating an ice cube tray, but perhaps a big error in fabricating an integrated circuit.

For training models, it doesn’t really matter what units we are using, since all we care about during training is having a heuristic to help us decrease the error with each iteration. We care only about relative size of the error from one step to the next, not the absolute size of the error.

But in evaluating trained models in data science for usefulness / accuracy , we do care about units, because we aren’t just trying to see if we’re doing better than last time: we want to know if our model can actually help us solve a practical problem. The subtlety here is that evaluating whether RMSE is sufficiently small or not will depend on how accurate we need our model to be for our given application. There is never going to be a mathematical formula for this, because it depends on things like human intentions (“What are you intending to do with this model?”), risk aversion (“How much harm would be caused be if this model made a bad prediction?”), etc.

Besides units, there is another consideration too: “small” also needs to be measured relative to the type of model being used, the number of data points, and the history of training the model went through before you evaluated it for accuracy. At first this may sound counter-intuitive, but not when you remember the problem of over-fitting.

There is a risk of over-fitting whenever the number of parameters in your model is large relative to the number of data points you have. For example, if we are trying to

predict one real quantity y as a function of another real quantity x, and our observations are (x_i, y_i) with $x_1 < x_2 < x_3 \ldots$, a general interpolation theorem tells us there is some polynomial f(x) of degree at most n+1 with $f(x_i) = y_i$ for i = 1, … , n. This means if we chose our model to be a degree n+1 polynomial, by tweaking the parameters of our model (the coefficients of the polynomial), we would be able to bring RMSE all the way down to 0. This is true regardless of what our y values are. In this case RMSE isn't really telling us anything about the accuracy of our underlying model: we were guaranteed to be able to tweak parameters to get RMSE = 0 as measured measured on our existing data points regardless of whether there is any relationship between the two real quantities at all.