Spark Machine Learning

CMPT 732, Fall 2022

Recap: Machine Learning

We have input columns (***features***), x, and ***parameters*** θ.

Functions $y(x; \theta)$ produce predictions, y.

We want parameters θ that minimize (some measure of) error between the predictions y and target ***labels*** t.

We ***train*** the model on labeled data to estimate good values of θ. We can check performance on other labeled data to check how the model will (probably) behave on future predictions where we don't have "truth": ***validation*** and ***testing***.

If we have a discrete set of labels/predictions, then we have a ***classification*** problem: we are trying to predict the class or category.

If we have a continuous set of labels/predictions, then we have a ***regression*** problem: we are trying to predict a numeric value.

These are ***supervised learning***, where we know "right" answers and want to make predictions to match.

Maybe we don't have labels are are trying to partition our inputs into to-be-discovered categories: ***clustering***.

Or we want to find data point that *don't* fit in with the rest: ***outlier detection***.

These are ***unsupervised learning***.

Spark ML

Spark has some ML tools built in.

The collection of models/tools in Spark aren't as complete as Scikit-Learn, but they're pretty good. Roughly: the most common ML models that can be adapted to big data sets are in Spark.

As with other components: there's an older RDD-based API, and a newer DataFrame-based API. We'll talk only about the DataFrame ML tools: the `pyspark.ml` package.

See also the Spark ML Guide.

Complete code for the example to follow: ml_pipeline.py.

Pieces we'll have:

DataFrame
: As before, store data: features, feature vectors, labels, and predictions.

Transformers
: Feature extraction and transformation. Generally manipulating the data to get the features you need. Implemented with `Transformer` instances.

Estimator
: Implementations of ML algorithms that make predictions: regressors, classifiers, clustering. Implemented with `Estimator` instances.

Pipelines

[Concepts may be familiar to you from Scikit-Learn, but also maybe not…]

We are often going to want to take the data we have, and manipulate it before passing it into the model: feature engineering, but also just reformatting and tidying the data.

That's what the `Transformer`s are for. Common pattern: data → `Transformer` → ⋯ → `Transformer` → `Estimator` → predictions.

You *could* apply transformations manually, but that's going to be tedious and error-prone: you have to apply the same ones for training, validation, testing, predicing.

A ***pipeline*** describes a series of transformations to its input, finishing with an estimator to make predictions. Implemented with `Pipeline` instances.

A pipeline can be trained as a unit: some transforms need training (PCA, indexer, etc), and the estimator certainly does.

Estimators need a single column of all features put together into a vector, so minimal pipeline might be:

```
assemble_features = VectorAssembler(
    inputCols=['length', 'width', 'height'],
    outputCol='features')
regressor = GBTRegressor(
    featuresCol='features', labelCol='volume')
pipeline = Pipeline(stages=[assemble_features, regressor])
```

Models

When a Spark estimator is trained, it produces a ***model*** object: a trained estimator that can actually make predictions; a `Model` (or probably `PipelineModel`) instance.

If we have some training data:

```
model = pipeline.fit(training)
```

Once trained, we can predict, possibly on some validation data:

```
predictions = model.transform(validation)
predictions.show()
```

Evaluation

Once you have a trained model, you probably want to come up with a score to see how it's working. This is done with `Evaluator` instances.

Regression and classification are evaluated differently, but the API is the same.

```
r2_evaluator = RegressionEvaluator(
    predictionCol='prediction', labelCol='volume',
    metricName='r2')
r2 = r2_evaluator.evaluate(predictions)
print(r2)
```

ML Algorithms

There are lots of ML algorithms to choose from. All of them implement the `Estimator` interface.

There is a nice collection of hyperparameter tuning tools in Spark (as there are in Scikit-Learn).

More Topics

Outside of the core Spark ML tools, there are other ML things where Spark might help.

Spark Deep Learning: deep learning tools implemented in Spark. Particularly nice API for transfer learning where you adapt a pre-trained model to your specific problem.

Joblib Apache Spark Backend: use Scikit-Learn for the machine learning implementation, but parallelize the hyperparameter search (or other Joblib tasks? Demo.) across a cluster.

Demo

The toy problem from last week: generate random numbers for length, width, height. Calculate volume as their product. Use ML to predict the volume.

Let's see if we can do better. And let's compare-and-contrast with Scikit-Learn.

Spark code: ml_demo_spark.py.

Some notes on the Spark regression:

- The predictions were very questionable, despite the good r^2 score.
- Some different hyperparameters on the GBTRegressor help: `maxIter=100, maxDepth=5`.
- The results from the predicions (`model.transform(validation)`) are the full DataFrame: original data, transformer output, targets, predictions.

Let's do the same task with Scikit-Learn to see how they are different.

- Approach is basically the same, but different API.
- Predicting only returns the predictions, not the other data.

Scikit-learn code: ml_demo_sklearn.py.

We could try different models or hyperparameters, but instead let's change the problem.

Maybe we don't care about the numeric value of these values, only their sign: are they positive or negative? That's a classification problem, much easier, and an excuse for some more transformations.

In Spark…

- A `SQLTransformer` can do arbitrary things to the input (both features and target).
- The target has to be numeric, so let's use `Binarizer`. (But both transformations could have been done with either `SQLTransformer` or `Binarizer`.)

In Scikit-Learn…

- Transformers can only work on the features, not the target: target must be transformed before the pipeline.
- Features must be numeric, not categorical.

For some more advanced ML in Spark, see the Databricks blog post Fine-Grained Time Series Forecasting.

Or more Databricks ML posts.

