Assignment 10

Spark Streaming

In this question, we will work with Spark Structured Streaming to process a constant stream of data in real(-ish) time. (Note that we are **not** using the older DStream-style streaming where you work with RDDs.)

We will get our streaming data from Kafka on our cluster. A process is already running that produces randomly-generated (x, y) points as plain text messages encode as text, separated by spaces like this: `"-740.844 -10829.375"`.

The code that produces the messages is simple: it generates random points near a line. This will simulate a stream of sensor data. [It's also possible it will die for reasons I don't foresee: if you aren't getting messages, bug Greg.]

We will imagine a situation where storing all of the incoming data isn't practical or necessary. The sensible thing to do in this case it to do some filtering/aggregating/summarization on the data as it comes in and stores only those results.

We will do a simple linear regression on the data (not using machine learning tools: just calculating the simple linear regression formula) and look at only the slope (which the Wikipedia page calls β) and intercept (α) of the values. (That's maybe not a particularly sensible summary of the data to store, but it's a not-totally-trivial calculation, and will give at least some idea of the data we're getting.) A reasonable formula to calculate the slope and intercept:

$$\beta = \frac{\sum xy - \frac{1}{n}\sum x \sum y}{\sum x^2 - \frac{1}{n}(\sum x)^2}$$
$$\alpha = \frac{\sum y}{n} - \beta\frac{\sum x}{n}$$

Technical Notes

There are messages being produced on three topics: `xy-1`, `xy-5`, and `xy-10`, which get one, five, and ten messages per second, respectively. If you want to see what's going on, you can try a simple consumer just to see the messages and confirm that they're coming through.

Call your Spark program `read_stream.py`. Take a command-line argument for the topic to listen on. You'll also have to load the Spark Kafka package on the command line:

```
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.1 read_stream.py xy-1
```

See our Kafka instructions for more information.

Working with Streams

All of the cluster nodes are in the Kafka cluster, so you can start by connecting to any of them. In Spark Structured Streaming, that will be like this:

```
messages = spark.readStream.format('kafka') \
        .option('kafka.bootstrap.servers', 'node1.local:9092,node2.local:9092') \
        .option('subscribe', topic).load()
values = messages.select(messages['value'].cast('string'))
```

Once you create a Streaming DataFrame (like `messages` and `values` above), you can specify calculations on it almost exactly like any other DataFrame.

The goal for this question is to aggregate to a DataFrame with a single row containing a slope and intercept.

Usually, data like this would be sent to a database, but output options seem limited in Structured Streaming. For this question, we will just output to the console (`.format('console')`), so we can read-off the slope and intercept. [❓]

Please write your streaming job with a modest **timeout** so you don't actually create an infinitely-running job:

```
stream = streaming_df.…start()
stream.awaitTermination(600)
```

Spark ML: Colour Prediction

We have collected some data on RGB colours. When creating the experiment, I showed the user an RGB colour on-screen and gave options for the 11 English basic colour terms. The result is >5000 data points mapping RGB colours (each component 0–255) to colour words. You can find it in the usual places as `colour-words-1`.

Let's actually use it for its intended purpose: training a classifier. This data set is likely small enough that you can do this question entirely locally: the cluster probably won't actually speed things up.

Create a Spark program `colour_predict.py` that takes the input path on the command line. Our output paths will be hard-coded:

```
spark-submit colour_predict.py colour-words-1
```

For this question, you **must create your classifiers as machine learning pipelines**. That is, each model will be a Pipeline instance containing each step in the workflow to classifying the colours.

Provided Pieces

You can download a ZIP file with some pieces for this assignment.

The `colour_tools.py` module contains code to help with this question. Save it in the same directory as your `colour_predict.py`.

The provided `colour_predict_hint.py` provides a starting structure for your `colour_predict.py`. The first thing it does is read the data and split into training and validation sets.

The other stuff in `colour_tools` will be described below…

The Pipeline

Here are the things needed in your Pipeline to get predictions out of this data:

- The classifiers all need a single column containing a vector of values. i.e. three columns (`'R'`,`'G'`,`'B'`) will have to become a vector of length three. The VectorAssembler transformer can do this for us.
- The targets in the data are strings: `'red'`, `'black'` and so on. The classifiers in Spark insist on predicting numeric values. The StringIndexer transformer can convert these values to numbers.
- You need a classifier that can actually made predictions. You're free to experiment with Spark's classifiers.

The classifier I used was a MultilayerPerceptronClassifier. I think you're welcome to use whichever classifier you like.

```
MultilayerPerceptronClassifier(layers=[3, 30, 11])
```

Training and Validating

Once you have a pipeline, you need to train it on the training data. This produces what Spark calls a "model" object: a trained version of the classifier pipeline.

To evaluate it (on the validation data), use a MulticlassClassificationEvaluator to produce an accuracy score, which we can use as a first approximation of "goodness" of a classifier. Print the score like this:

```
print('Validation score for RGB model:', score)
```

The provided `plot_predictions` function will produce a spectrum of colours, use your model to predict their colour word, and plot the results. Add this and have a look at the image `predictions-RGB.png` produced (on the local filesystem, not HDFS if you're on the cluster):

```
plot_predictions(rgb_model, 'RGB', labelCol='word')
```

Feature Engineering

Mapping RGB ↦ word is fundamentally one of human perception, but RGB colour space isn't about perception: it's about computer displays. The LAB colour space is designed to encode something about human vision: maybe we can create better input features.

The provided function `rgb2lab_query` produces a SQL query that converts RGB colours (in columns `'R'`,`'G'`,`'B'`) to LAB colours (columns `'labL'`,`'labA'`,`'labB'`). That might be convenient to combine with a SQLTransformer.

Create another pipeline for a LAB colour based model. It will be as before, but with a SQLTransformer in the pipeline to convert colour space. Output as before:

```
plot_predictions(lab_model, 'LAB', labelCol='word')
print('Validation score for LAB model:', score)
```

Tweak everything to get good results from each scenario without taking excessive processing time (but to be honest: getting the best possible results is a problem for another course). Compare the validation scores in the two scenarios. [❓]

When you submit the code, include the pipeline for both the RGB and LAB pipelines.

Predicting the Weather: How Hard Can It Be?

We know about machine learning. We have a bunch of weather data. Surely we can predict the weather with enough of both.

The Task

We will take another slice of the GHCN data: this time, daily maximum temperature values joined with the station locations (from their `ghcnd-stations.txt` file) so we have latitude, longitude, elevation of each observation.

This data can be found in the `tmax-1` to `tmax-4` data sets. It's CSV data with schema:

```
tmax_schema = types.StructType([
    types.StructField('station', types.StringType()),
    types.StructField('date', types.DateType()),
    types.StructField('latitude', types.FloatType()),
    types.StructField('longitude', types.FloatType()),
    types.StructField('elevation', types.FloatType()),
    types.StructField('tmax', types.FloatType()),
])
```

We would like to predict the tmax value based on the provided latitude, longitude, and elevation. The date is probably not a directly meaningful value, but can be transformed to the day-of-year (in a SQLTransformer).

In a program `weather_train.py`, **create a pipeline model** that uses the features latitude, longitude, elevation, and day-of-year to predict the tmax value. You should take the training/validation data set on the command line, and an output filename where we will save our trained model:

```
spark-submit weather_train.py tmax-1 weather-model
```

This is (1) a regression problem since you're trying to predict a continuous value, and (2) definitely not a linear regression. Your estimator should be some regression estimator chosen from those that Spark implements.

As before, you should split your input data into training and validation sets. Produce a score (r-squared and/or root mean square error) on the validation data using RegressionEvaluator to see how your model is doing. Choose a model and parameters to make reasonably good predictions.

In your code, save your **trained** model (a `PipelineModel` instance) in the filename from the command line like this:

```
model.write().overwrite().save(model_file)
```

Model Testing

A separate data set `tmax-test` has been provided: it should not be used in the training or parameter tuning processes, but can be used to evaluate your final model.

In the ZIP hint, you'll find a `weather_test.py` that will load your trained model, and evaluate on the data given:

```
spark-submit weather_test.py weather-model tmax-test
```

All of the provided data sets are generated by randomly selecting weather station, month pairs and keeping only the data for them. The test data has different random selections: the weather stations and times of the year will be different. [❓]

More Feature Engineering

Predicting the weather in the distant future is obviously hard. Maybe we can do better if we try an easier problem: let's predict the weather *tomorrow*.

Here is a SQL fragment that will help you add another feature: what was the temperature yesterday?

```
SELECT …, yesterday.tmax AS yesterday_tmax
FROM __THIS__ as today
INNER JOIN __THIS__ as yesterday
    ON date_sub(today.date, 1) = yesterday.date
    AND today.station = yesterday.station
```

Update your model to use this extra feature. Tune. Test. [❓]

Actually Predicting The Weather

Let's not forget that the goal of any ML problem isn't to get a good evaluator score: it's to actually make predictions.

This assignment is due on Friday November 18 2022. What will the tmax be on campus, *the day after* the assignment is due?

Our lab is at approximately 49.2771° latitude, -122.9146° longitude, elevation 330 m. You can assume the tmax on the due date will be 12.0°C. You can hard-code these values in `weather_tomorrow.py`, but take the model file on the command line:

```
spark-submit weather_tomorrow.py weather-model
```

Print your prediction for tomorrow's temperature:

```
print('Predicted tmax tomorrow:', prediction)
```

Questions

In a text file `answers.txt`, answer these questions:

1. What is your best guess for the slope and intercept of the streaming points being produced?
2. Is your streaming program's estimate of the slope and intercept getting better as the program runs? (That is: is the program aggregating all of the data from the start of time, or only those that have arrived since the last output?)
3. In the colour classification question, what were your validation scores for the RGB and LAB pipelines?
4. When predicting the tmax values, did you over-fit the training data (and for which training/validation sets)?
5. What were your testing scores for your model with and without the "yesterday's temperature" feature?
6. If you're using a tree-based model, you'll find a `.featureImportances` property that describes the relative importance of each feature (code commented out in `weather_test.py`) if not, skip this question). Have a look with and without the "yesterday's temperature" feature: do the results make sense and suggest that your model is making decisions reasonably? With "yesterday's temperature", is it just predicting "same as yesterday"?

Submission

Submit your results to the CourSys activity Assignment 10.

Updated Tue Nov. 15 2022, 17:20 by ggbaker.