Final Project

This is an individual project. This project is meant to assess your ability to use spark to handle streaming data, use spark for fitting a machine learning model, and a few other things we've talked about in class (and a few we haven't :).

Goals

For this, we'll

- write a Jupyter notebook that fits a machine learning model using `pyspark`'s `MLlib` module
- in that same notebook you'll write code to read in a stream of data (we'll produce that data ourselves using a `.py` file)
- we'll use the model to do predictions on the stream and write those out to the console!

The data is modified from the UCI machine learning repository. The study was about relating power consumption from different zones of Tetouan city to various factors like time of day, temperature, and humidity. - You'll have a chunk of the (modified) data to build your model on. - You will then stream data to a folder in the form of CSV files. You'll be monitoring this folder. When data comes in you'll use your model to predict on the incoming data!

Fitting Your Model (50 pts)

Create a Jupyter notebook for the modeling fitting part and the Streaming part below.

The file `power_ml_data.csv` is available at the URL: https://www4.stat.ncsu.edu/~online/datasets/power_ml_data.csv

You should read this data into a standard pandas data frame using the `pd.read_csv()` function. Convert this to a spark data frame (SQL or pandas is fine for now). We are going to treat the `Power_Zone_3` variable as our response variable. We can use all of the other variables as predictors. (Imagine we know that the `Power_Zone_3` reading is going to go offline in the future and we need to be able to predict that value appropriately.)

Once you have it in a **spark data frame** (SQL or pandas until we start our pipeline) we want to fit an elastic net model using CV (no training/test split, just using CV to choose our tuning parameters) with the following steps:

- Summarize the data set via common numerical summaries (within spark!) (20 pts)
  - Find means, standard deviations, min, max, and median values for all the variables where that makes sense
  - Find the correlations between all variables where that makes sense to do
  - Create a one-way contingency table of the `Month` variable. Repeat for the `Hour` variable.
  - Create a two-way contingency table for the `Month` and `Hour` variables
  - Group the data by `Month` and find the means of the numeric variables
  - Group the data by `Month` and find the standard deviations of the numeric variables
- Now we'll switch to using a Spark SQL style data frame for the remaining tasks (5 pts)
  - The `Hour` column is likely not stored as a `DoubleType`. Use a spark SQL data frame method to cast the variable as a `DoubleType`
- The rest of the operations below should use `MLlib` functions to set up a pipeline where we will do an Elastic Net fit (25 pts)

  - Binarize the `Hour` column based on the column being less than 6.5 or not (night vs day essentially)
  - One-hot encode the `Month` column
  - Run a PCA fit on the `Temperature`, `Humidity`, `Wind_Speed`, `General_Diffuse_Flows`, and `Diffuse_Flows` columns. To do this, I first used a `VectorAssembler()` call to place these variables in a column together for use with the `PCA()` estimator. Once fitted, the you'll have a PCA transformer we'll use on our pipeline.
  - For your elastic net model you'll use the fitted PCA features, binary `Hour` variable, `Power_Zone_1`, `Power_Zone_2`, and `Month` indicator variables as your predictors.
  - You should rename the `Power_Zone_3` variable appropriately to be your response.
  - This ends the pipeline of transformations. You'll then use the `CrossValidator()` function and the `LinearRegression()` function to fit an elastic net model. You should do the following grid for the `regParam` and `elasticNetParam`: All combinations of
    * `regParam`: 0, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.98, 0.99, 1
    * `elasticNetParam`: 0, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.98, 0.99, 1
  - Fit the model!
  - Find the training set RMSE as done in the notes by using your fitted model as a transformer.
  - Take the outputted transformations from the model (the predictions) and create a `residual` column (`label - prediction`). The `.withColumn()` method is handy.

Streaming Part (40 pts)

There is another file available at: https://www4.stat.ncsu.edu/~online/datasets/power_streaming_data.csv

Download this file and store it where you can find it. We'll be randomly sampling rows from this to output to `.csv` files that you'll be reading in.

Reading a Stream (15 pts)

- We're going to read in a stream in the form of `.csv` files. Create a folder where you will be sending your `.csv` files.
- Setup the schema for the stream (be sure to make `Hour` a double variable so you don't need to worry about that transform that wasn't part of our pipeline)
- Set up the `readStream`. Be sure to add `header = True` as you'll likely be outputting files with a header and we don't need to read that in.

Transform/Aggregation Step (20 pts)

- Now, we'll do two separate things on the stream and join them together:
  - With your stream, use your model transformer to obtain predictions from the incoming data. On the resulting predictions also create a `residual` column as noted in the previous section (return only the `label`, `prediction` and `residual` columns from this part)
  - With your stream, also create a second pipeline you'll send the new data through that does similar transformations to what we did above (everything up to the model fit part). One big difference, rather than just transforming and returning the `label` and `features` column, return all of the columns from the original data and the transformations you did (including the `label` and `features` columns).
  - Now join these two data frames together based on the `label` variable which should be common to both.
    * Note that each data frame is created from the same stream of data! You don't need two streams, you can use the same stream and just do two separate transformations on it, combining it with a `.join()` method from one of the SQL style data frames you are dealing with.

Writing Step (5 pts)

- Now write your stream to the console using the `append` output mode.
- Start the query!

Produce Data (10 pts)

You should have the streaming data downloaded. **Create a `.py` file** that reads that into a pandas (regular) data frame and does the following.

- Writes a loop (say 50 iterations) to:
  - Randomly sample three rows and output those to a `.csv` file in the folder you are watching with your stream.
  - Be sure not to write out the indices. You can leave the column names as long as you handle that on your stream appropriately
  - Pause for 10 seconds in between outputting of data sets
  - Submit this loop in a python console.
- While the loop runs, you should see output in your notebook!
- **Create a one minute video** that shows you start the query, start the loop, and your spark session outputting the combined prediction/transformation data to the console. This can easily be done with zoom. Please don't make the video very long as I want you to upload it to Moodle! Note: If you don't include the video you will lose more than 10 points of credit.d

To Submit

Upload the `.py` file with the code for reading and writing the streaming data (comment this file and use good programming practices), the `.ipynb` file that contains your model fitting and streaming setup (with a readable narrative about what you are doing and seeing of course), and the one-minute video demonstration of your streaming analysis!

Notes on grading:

- For each item in the rubric, your grade will be lowered one level for each each error (syntax, logical, or other) in the code and for each required item that is missing or lacking a description.
- **You should use Good Programming Practices when coding (see wolfware). If you do not follow GPP you can lose up to 25 points on the project.**

Be sure to include comments describing what you are doing, even when not explicitly asked for! Points will be deducted from appropriate sections as appropriate.