

Paul Gimeno

Dr. Benjamin Harvey

SEAS 6407 – Data Analytics Foundations and Practicum

05 December 2020

Real time text classification with Spark and Delta Lake

1. Project goal and motivation

Consumption and application of machine learning in the age of big data can provide significant value to businesses or individuals when benefits of obtaining insights outweigh the cost of obtaining it. Being able to bulk process and obtain insights from large sets of data can be daunting when actionable insights are needed on a continuous basis. Fortunately, when done correctly, the benefits of delivering real-time insights can provide significant advantages as we are able to make decisions that matter in the moment. Being able to classify a streaming feed of text for categorical context is a great use case for such tasks. These tasks can power live recommender systems, filter threats from communication logs or simply provide us an up to the minute summary of today's news. Streaming text data has many use cases in the real world and therefore will be the primary use case basis for the project.

The goal of this project is to present a scalable real-time text classification machine learning workflow that can:

1. Scale the job from one machine into a distributed cluster of machines.

2. Support the changing nature of on-going experiments and various implementations of each machine learning models.
3. Work with data storage layers that can support evolving input data sources and external sources for our streaming text data.

This project details an end-to-end lifecycle for real time data consumption, analysis, delivery and code maintenance for continuous information delivery through Spark and the Delta Lakehouse architecture. In particular, the project will apply a binary text classification task to an inbound text stream of review comments. For our simplified case, we will classify whether an incoming text review talks about a pizza establishment (or not) in the first 180 characters of the text.

II. Project Structure

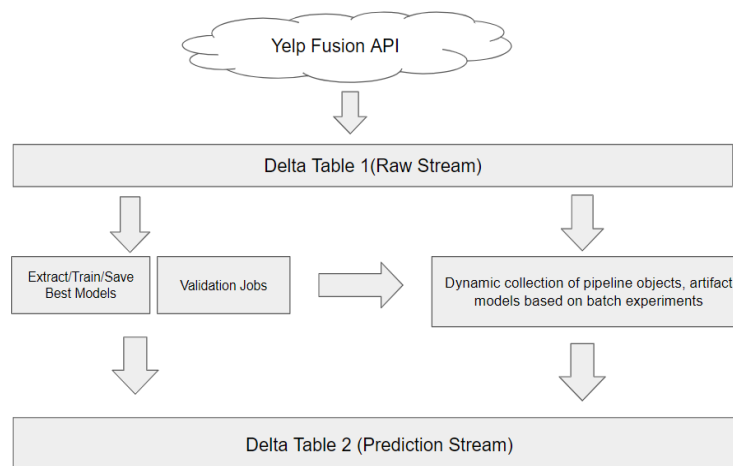


Figure 1: Project Structure

A. Project Data Source

Common among many data projects require the use of remote data sources hosted through cloud servers or on-premises data marts. In both cases, the mechanism to access database entry points are similarly structured where and often bundled in a common API across many languages. The project will utilize Yelp's Fusion REST API (<https://www.yelp.com/fusion>) endpoints as the primary basis for obtaining training data for our text classification task as well serve as our input streaming review stream to demonstrate the streaming capabilities of Spark's structured streaming.

The primary end points used for the project are:

1. Yelp's business search endpoint:

<https://api.yelp.com/v3/businesses/{id}>

2. Yelp's business review endpoint <https://api.yelp.com/v3/businesses/{id}/reviews>

API Limits

Many commercial API providers may provide limits on the numbers of API calls allowed. Yelp Fusion limits 5000 calls per day and so request methods must be properly structured in a way that maximizes the records fetched per request while minimizing redundant calls. This was handled by utilizing 'offset' parameters in the data request calls which effectively partitions the result set to be handled in segments only once.

Source Data Transformations

While the Spark's SQL engine supports nested map and struct objects (nested dictionaries and lists within columns) within its environment, it can't be assumed that we'll be ingesting the primary data into spark all the time. For this purpose, it was necessary to flatten all nested structures within each JSON result set before it can be saved into our primary storage layers in the delta environment. JSON flattening (exploding columns) was done using Python's Pandas library prior to saving.

B. Delta Table sinks

The project uses Delta table storage directories to ingest incoming data from Yelp's Fusion API as well as act as our output data sink to collect our text classification predictions. There are two Delta tables in this project design: The first table (Delta table 1) acts as a raw data sink for which the main driver will ingest raw data object and convert it to a structured Spark dataframe. The second table (Delta table 2) is our prediction table which receives input from both batch processing tasks as well as machine learning tasks that originate from the first input stream.

Delta Table advantages:

Delta table offers the rigidity of a traditional database management system where an input data schema can be enforced. For machine learning tasks and common reporting libraries, maintaining data type and column dimensions as data is being received and transformed is paramount to the sustainability of on-going machine learning and data engineering pipelines.

While this may sound too restrictive and rigid in the case where a data source may evolve over time, Delta tables interestingly offer us the flexibility to allow for permissive configurations that can update our schema without interfering with downstream tasks.

The Delta infrastructure can be rigid and flexible in the way it implements data updates. Any time a Spark job completes an update, it internally creates historical commits in a log file for which the engine uses to determine which version of the table that has records are current. By only committing completed jobs, Delta tables give us guarantees that corrupt overwrite tasks won't wipe the existing data on the table.

The commit/checkpointing design also provides us a few benefits in concurrent table access. Simultaneous writes to the same record in a Delta table will allow for both records to be logged in the commit history and no data is lost. The commit log enables us to rollback versions of our rows and partitions and it also allows us to revisit a snapshot of our data at any point. For our text classification project, we will be using two input streams to just one data table in collecting our predictions. The first input will be feeding streaming data into the delta table while the other will feed data through batch processing at any time without interfering with streaming data.

The last important benefit a Delta table can provide (among many) is the ability to invoke RDD transformations to it using a unified suite of language API's. We are no longer tied to the SQL language to do most of the transformations but instead, we can invoke table transformations through common language API's like Python and Scala. For an end-to-end project, having this flexibility ensures that our project can be easily maintained in the long run with evolving machine learning libraries and ever-growing suite of utility libraries.

C. **Project Environment**

The Databricks community edition supplies us a kernel which can coordinate the Delta storage engine as well as the interface needed to run our main driver. The project utilizes the Python programming language and will utilize the Pyspark's Mllib libraries within the Databricks environment as our core machine learning packages in our text classification task. The project is organized into 3 notebooks and an experiment URI.

Notebook 1: Project Init Notebook

Our project init notebook holds all the project's global functions in one namespace for organization and set-up. For projects that are expected to scale, considerations in module packaging for each project must be taken in consideration.

Notebook 2: Training Notebook

The project training notebook encapsulates all initial data acquisition tasks for training, modeling and project set-up. The modules contain the sequential transformations of our input training dataframe and save these transformations for later use via pipeline objects. Pipeline objects can be reused within the project environment by loading it to another notebook, porting it to another experiment or sharing it across a cloud repository. Pipeline objects that are defined by this notebook are also loaded into our streaming tasks as new data is ingested into the delta table and scored by our best predictive model. A project can contain multiple training notebooks that may correspond to its own experiment if needed.

Notebook 3: Streaming Driver Notebook

This notebook serves as the primary driver for fetching our yelp input text input stream into Delta table #1 (our raw stream). The raw stream applies top-level data validation and de-duplication and ensures that the data dimensions are appropriately structured prior to entering our predefined pipeline objects (which vectorizes our data) and predictive model objects that we saved from our experiments. Data flows through the pipeline and is received by Delta table #2 which saves our predicted text

Notebook 4: Batch Notebook

The batch validation notebook contains methods and workflow that append data to Delta table #2 via batch processing. This notebook feeds predictions and data alongside our ingest stream and can run concurrent with the Spark's dataframe writer. This notebook represents tasks outside of experimentation and modeling and re-uses all of the pipeline objects and models already defined by our training notebook

Experiment URI

This project uses Mlflow tracking to track, save and save versions of our machine learning project. For this project, the initial training and modeling is saved to a global experiment ID for which we are able to save all improvements and tweaks to our experiment model. This

experiment repository will be the basis for future queries on our best experiments and models to be used in our text classification project.







					Parameters	Metrics	
Run Name	User	Source	Version	Models	type	accuracy	error_rate
Random Forest Classi...	pgimeno@gwu.edu	 1_Training	-	 spark	RandomForestClassifi...	0.94	0.06
Naive Bayes	pgimeno@gwu.edu	 1_Training	-	 spark	NaiveBayes_b2c8abe...	0.57	0.43
Logistic Regression	pgimeno@gwu.edu	 1_Training	-	 spark	LogisticRegression_1...	0.94	0.06
Load more							

Figure 2: Mlflow experiment repository

III. Modeling and Training

Prior to setting up the prediction stream, the project must deploy an initial experiment and lay the groundwork for how the data pipeline is managed in a generalized manner. The project uses Pyspark native machine learning libraries and transformer methods to demonstrate how Spark's pipeline objects can significantly enhance the data transformation workflow. To properly classify the incoming text stream, the review text must need to be transformed in a way that i can be mathematically compared:

A. Text transformation and vectorization.

The project uses a feature vectorization method called Term frequency-inverse document frequency (TF-IDF) to rank the importance of each word in each discrete text stream and make sense of its importance in the entire set. Set in this instance, is the collective words used in the training data set. Each review goes through a series of

transformations before it can be used in any of our classification algorithms. The purpose of these transformations are to convert the text into vectors in a format where we are able to deal with each word count and convert them to vectorized columns. These transformations are as follows:

1. String Tokenization:

Tokenization refers to the process of deconstructing a sentence, paragraph or blob of text into a discrete list of words. Pyspark's tokenization function organizes each individual word into a list. This allows us to see each review as a list of words and operate in its discrete parts.

2. Removal of stop words

Stop words refer to inconsequential or common words that can be removed in document word cloud. Examples are word like here, of, seem, is, every, much. Every document will contain one or more of each words as it is common within the English language

3. Count Vectorizer

Once the text has been tokenized and stripped off Pyspark's stop words vocabulary, it is then passed into a count vectorizer transformer. This transformer counts how many times each word appears in one document (review). It introduces a hashing structure in the following format: {word: <count>}. This will be useful as this describes the term frequency of each word.

4. IDF

IDF is a transformer that takes our transformed vectors from the count vectorizer transformer and applies an inverse log scaling to certain terms. Terms that appear frequently in the corpus of the document is heavily scaled down. This mechanism aids in differentiating words that correlate to pizza establishments by using significant uncommon terms.

5. Vector Assembler

Vector assembler combines our list of transformed columns into a single vector column. This transformer arranges our raw features in a form that is suited for our classification algorithms in Pyspark's Mllib library.

B. Algorithm selection

The project uses three classification algorithms to identify a suitable classification model for the incoming text stream. These are A) Naive Bayes, B) Random Forest and C) Logistic Regression. Once each review text is vectorized and transformed by our pipeline object.

C. Findings

The initial training yielded the following results and model selection was based on the highest f1 score of the set experiment.

<i>Classifier</i>	<i>True Positive/False Positive</i>	<i>True Negative / False Negative</i>	<i>Accuracy</i>	<i>F1 Score</i>
Binomial Logistic Regression	401 / 173	207 / 107	68%	.741
Naive Bayes	365 / 121	259 / 143	70%	.734
Random Forest	506 / 370	10 / 2	58%	.731

Table 1: Classification Performance

IV. Model application to machine learning task

Using the best model from the initial experiment

Once the initial experimentation/training is complete, the remaining execution of the training notebook (first notebook) saves the experiment parameters into the project's experiment repository. Each run's metric is logged through Mlflow for model selection in the validation and streaming component of the project. Throughout this process, a model artifact is also saved which can be saved and loaded by another driver in the project. The convenience of organizing metrics along with runtime objects as artifacts enables a seamless long-term maintenance of the overall process.

Model application in validation phase

Once we have acquired an appropriate training model, it is then invoked and loaded both our streaming driver as well as our batch validation notebooks. The batch validation notebook takes in new data from our second delta table that includes all the streaming predictions and can validate the newly acquired datasets. The validation notebook can enhance training data sets and/or serve as the primary driver for loading and querying across experiments for the best model to use in the streaming notebook.

Model application in the streaming phase

The input stream reader that collects data from the first Delta table and saves predictions to the second Delta tables undergoes predefined transformations which use a predefined pipeline to transform the raw data stream and pre-defined fitted model to use in making the prediction. The batch validation notebook in the project saves the best model to use in the global namespace and it is used by the streaming notebook as it runs and processes new data input continuously.

VI. Project maintenance and continuous experimentation

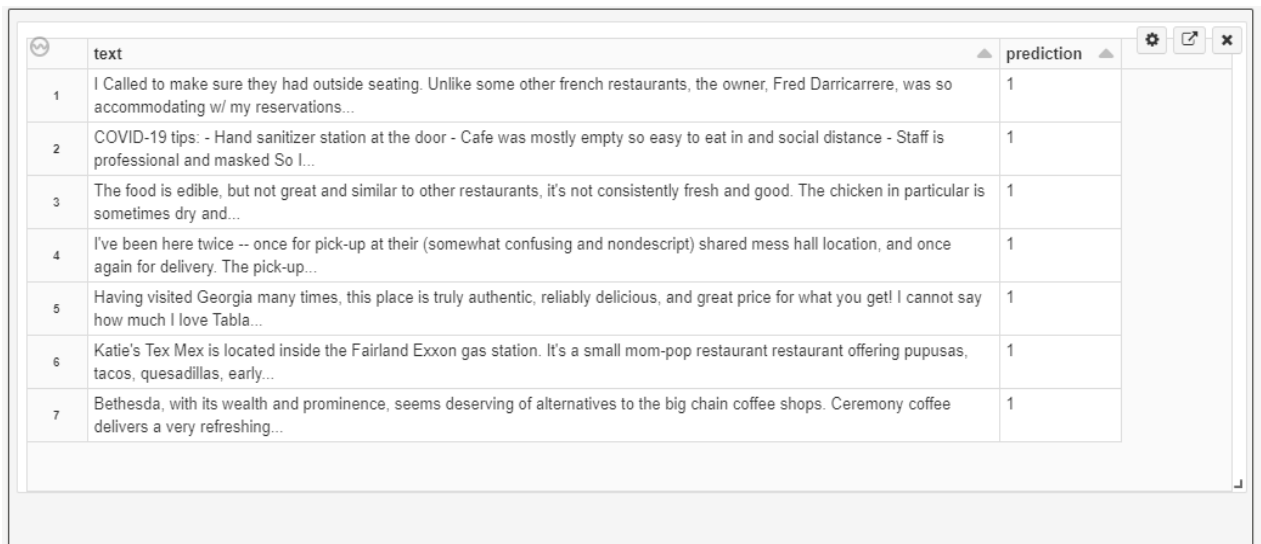
Once the models have been selected and loaded in our streaming driver (streaming notebook) it can begin predicting new input yelp reviews new data arrives. This process can run in perpetuity where new data arrives in Delta table 1, gets processed and transformed

incrementally by our intermediary stream reader which then performs the predictions saves the predictions into Delta table 2.

Our training notebook can continuously run multiple text classification experiments alongside the streaming task and update and modify the parameters used in the classification algorithms. Updated models can be uploaded to the global project repository within Mlflow so that it can be queried through validation processes. In the case where the model needs to be updated in the active streaming job, our batch drivers can simply update to a new global model artifact URI which is saved in a global variable that our streaming job can reference to.

VII. Insight delivery and project expansion

While Databrick's internal environment can display live predictions in a dashboard-like setting, the beauty of having a centralized repository through Mlflow and Delta tables is that these resources can be referenced outside the Databricks environment through REST API interfaces, command line interface applications, git repositories or a front facing data storage sink that can provide dashboarding BI capabilities on the web. The prediction data stream (Delta table 2) can be expanded where it can be read by another stream that feeds into a database management system or diverge into other ecosystems that can utilize the predictions into triggers for web applications.



	text	prediction
1	I Called to make sure they had outside seating. Unlike some other french restaurants, the owner, Fred Darricarrere, was so accommodating w/ my reservations...	1
2	COVID-19 tips: - Hand sanitizer station at the door - Cafe was mostly empty so easy to eat in and social distance - Staff is professional and masked So I...	1
3	The food is edible, but not great and similar to other restaurants, it's not consistently fresh and good. The chicken in particular is sometimes dry and...	1
4	I've been here twice -- once for pick-up at their (somewhat confusing and nondescript) shared mess hall location, and once again for delivery. The pick-up...	1
5	Having visited Georgia many times, this place is truly authentic, reliably delicious, and great price for what you get! I cannot say how much I love Tabla...	1
6	Katie's Tex Mex is located inside the Fairland Exxon gas station. It's a small mom-pop restaurant restaurant offering pupusas, tacos, quesadillas, early...	1
7	Bethesda, with its wealth and prominence, seems deserving of alternatives to the big chain coffee shops. Ceremony coffee delivers a very refreshing...	1

Figure 3: Databricks live prediction feed

VIII. Conclusion and Summary

To summarize the main components of this text classification project workflow, we can describe four distinct components that make up the entirety of the workflow. The first component is the raw data ingest process which is described by the continuous fetching of API calls to Yelp's fusion API server and into our first Delta table. The second component refers to the initial experimentation that is needed to make use of the raw data stream from the first Delta table. In the case of this project, we have used a training notebook within Databricks to drive the training, the testing, the model selection and experimentation into a workspace Mlflow library which tracks the performance of our experimentation. The third component is our batch validation and pipeline set-up. Once we have completed our preliminary experiments, we are now ready to set-up pipelines that is able to transform incoming text into vectorized columns of matrices ready to be consumed by classification algorithms. The batch and validation process are responsible for querying the best model in our experimentation within the Mlflow repository and

selecting the appropriate artifacts and models to load to be used in driving our real-time predictions. This process can run concurrent writes to Delta table 2 along with the streaming prediction stream. The fourth and last component is the streaming prediction task that is set up by instantiating a stream reader from Delta table 1. This stream reader invokes all the global objects, variables and model artifacts we have set-up in our training and batch notebooks in order to transform and make predictions of new incoming texts into Delta table 2.

Throughout the process, we may wish to validate the predictions in Delta table 2. Luckily, Delta tables allow for concurrent reads and writes, and our batch validation driver can simply perform recurring batch appends to Delta table 2 without interfering with live updates being done by our prediction streams. Checkpointing and commit rollbacks is an option if we decide to revert partitions of our delta table into previous versions of it.

This simplistic design can be extended by addition of more input streams, more output streams, more experiments and a more elaborate model fitting pipeline. The extra streams can be supported by the same Delta tables we have set-up in this simple case since Delta tables are not entirely rigid in its design. Evolving schemas are entirely supported but strictly enforced at runtimes. The beauty of a “Delta lakehouse” is that it provides the rigidity of a database management system and the flexibility of a data lake. In the case of upscaling our experiments, pipelines and artifacts, the core of the project can remain intact and we’ll just focus on managing the machine learning project lifecycle within a common Mlflow repository interface. We can run variations of our experiments independent of our data transformations and streaming data pipelines.

References:

Cady, F. (2017). *The data science handbook*. Hoboken, NJ: John Wiley et Sons.

MLflow guide. (n.d.). Retrieved December 05, 2020, from <https://docs.databricks.com/applications/mlflow/index.html>

Unified Data Analytics. (2020, December 03). Retrieved December 05, 2020, from <https://databricks.com/>

Vakil, P., & Patano, F. (2020, November 03). Why Cloud Centric Data Lake is the future of EDW. Retrieved December 05, 2020, from <https://databricks.com/blog/2020/11/03/why-cloud-centric-data-lake-is-the-future-of-edw.html>

Yelp Fusion Endpoint documentation. (n.d.). Retrieved December 05, 2020, from <https://www.yelp.com/developers/documentation/v3>