Mastering Snowflake: Sentiment Analysis and Performance Experiments

Veron Hoxha (veho@itu.dk)
IT University of Copenhagen
November 25, 2024

1 Introduction

In this project, we implement the Naïve Bayes algorithm in SQL, specifically for sentiment analysis. We apply the Naïve Bayes algorithm using training data and evaluate its precision against test data. Given the complexity and occasional impracticality of using loops in SQL, we address this challenge by performing sentiment analysis using a User-Defined Table Function (UDTF). We implement the Naïve Bayes algorithm through UDTF in Python and then analyze its precision against the test data. Additionally, we compare both implementations in terms of the work required, complexity of the code, precision, and performance in terms of execution time.

To delve deeper into performance evaluations, we conduct performance experiments using TPC-H. We select three out of the twenty-two queries specifically Query 1, Query 5, and Query 18[1][2][3] and use these queries to run performance experiments on four different sizes of TPC-H data with Scaling Factors (SF) of 1, 10, 100, and 1000, as well as four different sizes of virtual warehouses: XS, S, M, and L. We then measure the query runtimes.

In Section 2, we describe the experimental methodology of the project. In Section 3, we present the results of our experiments. In Section 4, we interpret the results and discuss additional findings. Finally, in Section 5, we summarize our findings.

Detailed project codes and additional resources are available on GitHub at https://github.com/veronhoxha/snowflake-sentiment-analysis-and-performance-experiments.

2 Experimental Methodology

In this section we describe the experimental design of our project where we explain the dataset we used for performing sentiment analysis and it's characteristics, Snowflake client used, Snowflake configuration, TPC-H queries, and in the end we show the number of runs we do to measure the performance and precision of our experiments.

[1]Sample Data TPC-H:https://docs.snowflake.com/en/user-guide/sample-data-tpch
[2]TPC-H official website:https://www.tpc.org/
[3]TPC-H documentation:https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf

2.1 Dataset and it's characteristics

The data we use are sample product reviews from Yelp (YelpReviewFull) [1] [2]. It is extracted from the Yelp Dataset Challenge 2015 data. The dataset is mainly used for text classification in the context of **"given the text" -> "predict the sentiment"**. The reviews are mainly written in english and a typical data point, comprises of a text and the corresponding label.

Dataset Structure An example from the YelpReviewFull test set looks as follows:

```
{
    'label': 0,
    'text': 'I got \'new\' tires from them and within two weeks got
        a flat. I took my car to a local mechanic to see if i could
        get the hole patched, but they said the reason I had a flat
        was because the previous patch had blown - WAIT, WHAT? I
       just got the tire and never needed to have it patched? This
       was supposed to be a new tire. \\nI took the tire over to
       Flynn\'s and they told me that someone punctured my tire,
       then tried to patch it. So there are resentful tire slashers
       ? I find that very unlikely. After arguing with the guy and
       telling him that his logic was far fetched he said he\'d
       give me a new tire \\"this time\\". \\nI will never go back
       to Flynn\'s b/c of the way this guy treated me and the
       simple fact that they gave me a used tire!'
}
```

The YelpReviewFull training set data has the exact same structure.

Data Fields

- **"text":** The review texts are escaped using double quotes ("), and any internal double quote is escaped by two double quotes (""). New lines are escaped by a backslash followed with an "n" character, that is `"\n"`.
- **"label":** Corresponds to the score associated with the review (between 1 and 5 stars) where 1 star review is equivalent to label 0 and 5 star review to label 4.

Data Splits In total there are *650,000* training samples with total size of *286 MB* and *50,000* testing samples with size of *22 MB* which are used in the project.

Downloading Data Before downloading data we needed to make sure we have Git LFS (Large File Storage) installed in order to download the full dataset. Afterwards, in order to download the dataset in our local machine the following command was ran:

```
git clone https://huggingface.co/datasets/Yelp/yelp_review_full
```

2.2 Snowflake Client Used

SnowSQL was used as the CLI client, which is the command line client for connecting to Snowflake to execute SQL queries and perform all DDL and DML operations, including loading data into and unloading data out of database tables[4]. The data was downloaded to our local machine as described in Paragraph 2.1. Subsequently, to perform the tasks in this project, the data needed to be uploaded into Snowflake. In this case, a command-line tool was necessary because the Snowflake UI restricts uploading datasets larger than 250MB, and our total dataset size is 308MB. The simplest method to accomplish this was to install the SnowSQL client on our machine as we did and connect to the Snowflake account through SnowSQL using our login credentials. The account name I used for SnowSQL is *"sfedu02-gyb58550"*, as provided by the professor.

The process followed to load the data through SnowSQL was taken from the official Snowflake documentation, which offers a detailed tutorial on how to load and upload Parquet data[5], combined with assistance from the exercise sessions in class. The following commands were executed sequentially to load the data into Snowflake using the SnowSQL CLI:

```
-- setting up of the database and warehosue
CREATE DATABASE IF NOT EXISTS COYOTE_DB;
USE DATABASE COYOTE_DB;
USE WAREHOUSE COYOTE_WH_XS;

-- creating tables
CREATE OR REPLACE TABLE yelp_training(val variant);
CREATE OR REPLACE TABLE yelp_testing(val variant);

-- creating a file format for parquet files and  a temporary stage
CREATE OR REPLACE FILE FORMAT parquet_format TYPE = parquet;
CREATE OR REPLACE TEMPORARY STAGE coyote_stage FILE_FORMAT =
   parquet_format;

-- putting the training and testing files into the stage
PUT file://Users/veronhoxha/Desktop/yelp_review_full/
   yelp_review_full/train-00000-of-00001.parquet @coyote_stage;
PUT file://Users/veronhoxha/Desktop/yelp_review_full/
   yelp_review_full/test-00000-of-00001.parquet @coyote_stage;

-- copying data into tables using the correct file format
COPY INTO yelp_training FROM @coyote_stage/train-00000-of-00001.
   parquet FILE_FORMAT = parquet_format;
COPY INTO yelp_testing FROM @coyote_stage/test-00000-of-00001.
   parquet FILE_FORMAT = parquet_format;
```

After following those steps the data will be loaded into Snowflake and ready to be used.

[4]SnowSQL (CLI client):https://docs.snowflake.com/en/user-guide/snowsql

[5]Loading and unloading Parquet data:https://docs.snowflake.com/en/user-guide/tutorials/script-data-load-transform-parquet

2.3 Snowflake Configuration

Sentiment Analysis using SQL and a UDTF - Configuration For the sentiment analysis tasks using SQL and a UDTF, the Large (L) virtual warehouse was used, in combination with the `COYOTE_DB` database. The approach for writing the UDTF was guided by the official Snowflake documentation[6].

Performance Experiments Using TPC-H - Configuration Performance experiments involving TPC-H were conducted in a Python notebook (`.ipynb` file) while externally connecting to Snowflake. The initial connection setup was as follows:

```
conn = snowflake.connector.connect(
    user='COYOTE',
    password=PASSWORD,
    account='sfedu02-gyb58550',
    schema='TPCH_SF1',
    warehouse='COYOTE_WH_XS',
    database='SNOWFLAKE_SAMPLE_DATA'
)
```

In this configuration, the username `COYOTE` was assigned to me from the proffesor, the password was securely stored in a `.env` file, and the account information was provided by the professor as well. The initial schema used was `TPCH_SF1`, and the initial virtual warehouse was `COYOTE_WH_XS`. Adjustments to the schema and virtual warehouse were made later as necessary to accommodate various experiments. The `SNOWFLAKE_SAMPLE_DATA` database was selected as it contains the TPC-H data within Snowflake. In order to run the experiments a person needs to use it's own `user`, `password` and `account`.

2.4 TPC-H Queries

For our performance experiments, we used TPC-H, a classic benchmark used to evaluate the performance of OLAP database systems. TPC-H data is already loaded in various sizes in Snowflake and you can find the data in different schemas within the `SNOWFLAKE_SAMPLE_DATA` database on Snowflake. Out of the twenty-two available queries, we selected three for our experiments. Each query had specific parameters that could be substituted with some specific values, as detailed on the official TPC-H benchmark official documentation[1][2][3]. The parameters that were picked by me and substituted are highlighted in bold next to their corresponding queries below. The complete queries are

[6]Writing a UDTF in Python:https://docs.snowflake.com/en/developer-guide/udf/python/udf-python-tabular-functions#label-udtf-python-create-handler

accessible in the GitHub repository within the `tpch_benchmark/tpch_benchmark.ipynb` file.

- Query 1 - a scan-intensive query (**DELTA = 90**).
- Query 5 - a join-intensive query (**REGION = ASIA; DATE = 1994-01-01**).
- Query 18 - an aggregate-intensive query (**QUANTITY = 313**).

2.5 Number of Runs to Measure Performance

To ensure the reliability and significance of our results, each implementation and experiment was executed three times. This approach allowed us to accurately measure both the runtime performance and precision of the sentiment analysis algorithms, as well as the runtime performance of the TPC-H queries.

3 Results

As specified in Subsection 2.5, each implementation and experiment was executed three times to ensure reliability and to mitigate variations in performance. In Subsections 3.1 and 3.2, we present the results for performance in terms of execution times and precision for the implementations of Sentiment Analysis using SQL and a UDTF, along with the query runtimes for the TPC-H experiments.

3.1 Sentiment Analysis using SQL and a UDTF

Table 1 presents the average execution times and precision of the implementations of Sentiment Analysis using SQL and a UDTF after three runs. We must consider that the "Average Time" can fluctuate, and various factors may impact it.

Implementation Type	**Average Time (s)**	**Average Precision (%)**
Sentiment Analysis in SQL	67.33	91.62
Sentiment Analysis using UDTF	88.10	92.93

Table 1: Average execution times and precision for both Sentiment Analysis implementations after three runs with a virtual warehouse of size L.

It is important to note that the performance, in terms of execution times and precision of both implementations, was measured manually by running each implementation three times.

3.2 Performance Experiments Using TPC-H

Figure 1 shows the average query execution times by schema and warehouse size for each query after three runs.

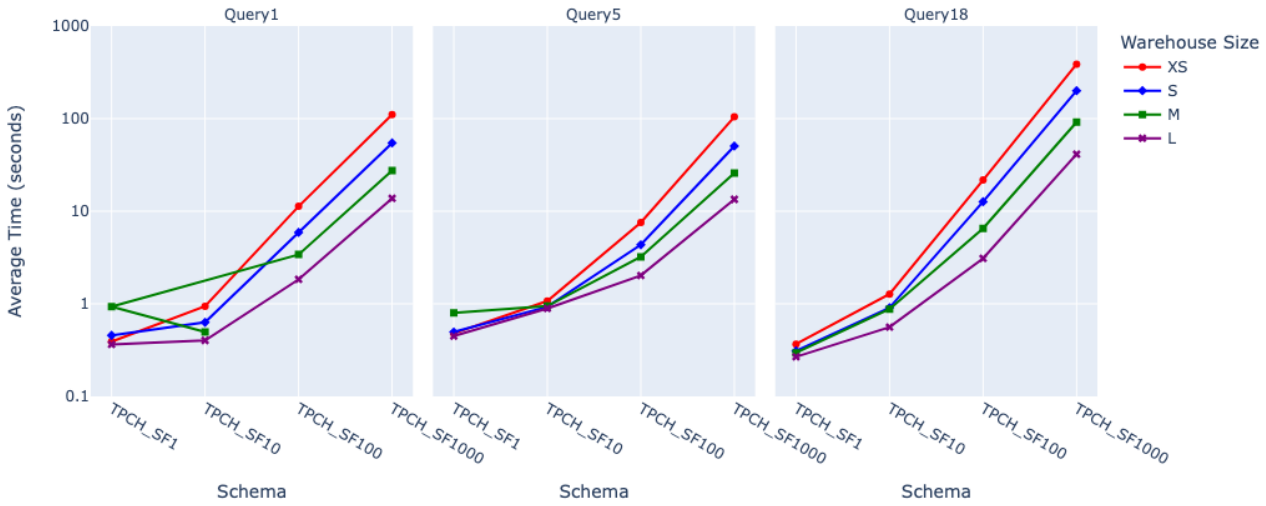

Figure 1: Average query execution times by schema and warehouse size for each TPC-H query after three runs.

4 Discussion

4.1 Comparison between the two implementations of the sentiment analysis algorithm

Table 2 shows the comparison between both Sentiment Analysis implementations in terms of precision, performance in execution time, amount of work required and simplicty of code.

4.2 Performance of TPC-H Queries

The results from our experiments with TPC-H queries were largely expected. As illustrated in Figure 1, the average execution time for all queries increases as the data scale factor (SF) progresses from SF1 to SF1000. This trend demonstrates that the systems scale effectively with the increase in SF and as the virtual warehouse capacity expands from XS to L. Although the scalability is not perfectly linear, with some deviations in performance particularly for Query 1 between SF1 and SF10, the overall scaling behavior is consistent, acceptable and aligns with expectations. For detailed query runtimes please refer to GitHub repository inside the `tpch_benchmark` folder.

5 Conclusion

This study systematically compared the performance of sentiment analysis implemented via SQL and Python UDTFs, as well as evaluated the scalability of TPC-H queries across different virtual warehouse sizes and scale factors. The findings confirm that while SQL provides faster execution times, Python UDTFs offer greater flexibility and slightly higher precision. Both systems demonstrated robust scalability, though with some deviations that were within acceptable limits.

Implementation Type	Sentiment Analysis using SQL	Sentiment Analysis using UDTF
Precision	Achieved an average precision of 91.62%. The model performs effectively but may miss some subtle sentiment nuances due to SQL's limitations in text processing.	Achieved an average precision of 92.93%. The Python UDTF handles complex text analysis better, slightly improving accuracy.
Performance in Execution Time	Faster execution time, averaging 67.33 seconds, benefiting from SQL's optimized query processing and set based operations.	Execution time averaged 88.10 seconds, slower due to the overhead of running Python code within the UDTF.
Amount of Work Required	Requires writing multiple SQL queries and managing several tables and views, which can be complex.	Sums up most of the logic within the Python UDTF, reducing the number of SQL scripts needed and simplifying setup.
Simplicity of Code	SQL code is lengthy and complex, making it harder to read and maintain, especially when implementing algorithms not naturally suited to SQL.	Python code is more concise and modular, making it easier to understand and maintain, especially for complex algorithms.

Table 2: Comparative analysis of Sentiment Analysis implementations.

For the TPC-H queries, the increase in execution time with larger data scale factors (SF1 to SF1000) and across warehouse sizes from XS to L suggests a predictable degradation of performance in line with system load increases. Moreover, the consistent performance across different warehouse sizes indicates that Snowflake's architecture effectively manages larger and more complex queries, aligning with the anticipated benefits of cloud data warehouse scalability.

In practical terms, the choice between SQL and Python for sentiment analysis should be guided by the specific requirements of the task, considering the trade offs between execution speed and analytical depth. This study based on the results also underscores the importance of selecting appropriate warehouse sizes based on the expected data volume and computational demands to optimize both performance and cost.

Overall, these insights not only validate the theoretical expectations but also provide actionable guidance for optimizing database operations in Snowflake.

References

[1] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

[2] Yelp. URL https://huggingface.co/datasets/Yelp/yelp_review_full.