Mastering Snowflake: Sentiment Analysis and Performance Experiments

Veron Hoxha (veho@itu.dk)
IT University of Copenhagen
December 14, 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, 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 [1, 2, 3]. We then measure the query runtimes.

The report is divided into multiple sections, each contributing to the overall project. 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 code 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, SQL implementation methodology, UDTF implementation methodology, TPC-H queries, and in the end we show the number of runs we do to measure the performance and precision of our experiments.

2.1 Dataset and it's characteristics

The data we use are sample product reviews from Yelp (YelpReviewFull). 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 a text and the corresponding label [4, 5].

Dataset Structure An example from the YelpReviewFull test set looks as follows [4, 5]:

```
{
    '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"` [4, 5].
- **"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 [4, 5].

Data Splits In total there are *650,000* training samples with total size of *299.4 MB* and *50,000* testing samples with size of *23.5 MB* which are used in the project. The files are in Parquet format.

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 [6]. 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 322,8MB. 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 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 [7], 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. It is important to note that while the warehouse used in this code above was already created, new warehouses can be easily created to accommodate different project needs. The creation of a new warehouse, if necessary,

can be done using the following example SQL command:

```
CREATE OR REPLACE WAREHOUSE <name> WITH WAREHOUSE_SIZE = '<size>',
AUTO_SUSPEND = 10;
```

Different names can be used based on personal preference and different warehouse sizes can be used starting from `XSMALL, SMALL, MEDIUM, LARGE`, etc [8]. We need to make sure to set `AUTO_SUSPEND = 10` when creating a new warehouse. This means the warehouse will be suspended after 10 seconds of inactivity.

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 [9].

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 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`. For the steps how to store the `password` in `.env` file there is a guide inside `tpch_benchmark/tpch_benchmark.ipynb` notebook.

2.4 SQL Implementation Methodology

In the SQL-based implementation of sentiment analysis, the core steps of the implementation were as follows:

- **Environment Setup:** We initialized a clean working environment by creating a new database and setting up the appropriate virtual warehouse in Snowflake as mentioned in Paragraph 2.3. This ensured that the system was ready to handle datasets efficiently.
- **Data Preparation and Preprocessing:** We created tables for storing stopwords, training data, and various intermediate results. A custom SQL function, `clean_text`, was implemented to remove stopwords, remove special characters, convert text to lowercase, perform tokenization, and stemming. This step was essential for preparing the textual data for the Naïve Bayes classifier.
- **Model Implementation:** The Naïve Bayes algorithm was implemented entirely within SQL. This process involved calculating word frequencies, prior probabilities, conditional probabilities, and other metrics for sentiment prediction, all within Snowflake itself. Largely, we followed the guidelines from Stanford University to compute these steps [10].
- **Performance Evaluation:** We measured the performance of the algorithm by calculating the whole implementation's execution time and precision on the test data. This allowed us to assess both the efficiency of the SQL implementation and the accuracy of the sentiment predictions.

2.5 UDTF Implementation Methodology

For the UDTF implementation, we used Python within the Snowflake environment to perform more easily the tasks that were not easily achievable using pure SQL. The Python based approach provided greater flexibility for model training and testing. The main steps in this methodology were:

- **UDTF Configuration:** As Python based UDTF was created within Snowflake this allowed us to use Python's extensive libraries for text processing and machine learning directly within the Snowflake environment.
- **Data Preparation and Preprocessing:** Within the UDTF, we applied a range of text preprocessing techniques, including tokenization, stopword removal, special characters removal, converting text to lowercase, and stemming. These operations were crucial for converting raw text into features suitable for sentiment analysis.
- **Model Training and Prediction:** The Naïve Bayes classifier was trained using the processed training data within the UDTF. After training, the function was used to predict sentiment labels for new, unseen test data.

- **Performance Evaluation:** We evaluated the performance of the UDTF implementation by measuring execution time and calculating precision.

Due to the large and detailed nature of both implementations, please check the code in the repository available in Section 1 for more precise details. Many steps, such as environmental setup, data preparation and processing, and performance evaluation, are common to both approaches. The primary differences lie in the way the algorithm is implemented in SQL and the other in Python.

2.6 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 substituted values are highlighted in bold next to their corresponding queries below. The complete queries are accessible in the GitHub repository within the `tpch_benchmark/tpch_benchmark.ipynb` notebook.

- 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**).

Each query was executed across four different scaling factors of TPC-H, SF1, SF10, SF100, and SF1000. Additionally, we used four different sizes of virtual warehouses XS, S, M, and L to ensure a comprehensive performance evaluation. This setup resulted in each query being executed 16 times (4 scaling factors $\times$ 4 warehouse sizes), totaling 48 runs per query to cover all possible combinations of scaling factors and warehouse capacity.

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 Paragraph 2.6, 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 acknowledge that the average time and precision may have slight fluctuations each time the implementations are executed. Various factors, including system load, and configuration changes, can influence these metrics.

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.

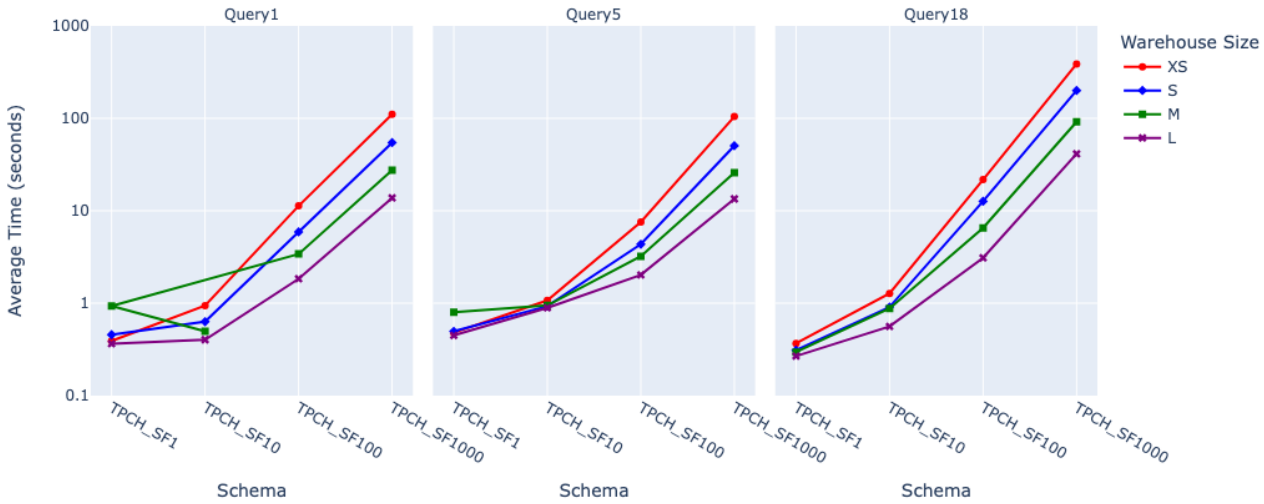

Figure 1: Average query execution times by schema and warehouse size for each TPC-H query after three runs.

4 Discussion

In this project, our primary objective was to explore the capabilities of Snowflake, a modern cloud data platform, to understand its efficiency in handling complex data analysis tasks. We focused on implementing a sentiment analysis algorithm using both SQL and UDTFs to assess their effectiveness and complexity. In the following subsections, we will dive deeper into a detailed comparison between these two implementation approaches. Additionally, we will review the performance outcomes of TPC-H queries to understand how well the system scales with increasing data volumes. Our discussions aim to interpret these results and discuss the findings that happened during the experimentations.

4.1 Comparison between the two implementations of the sentiment analysis algorithm

In Table 2 we show the key comparisons between both Sentiment Analysis implementations in terms of precision, performance in execution time, amount of work required and simplicty of code.

Implementation Type	**Sentiment Analysis using SQL**	**Sentiment Analysis using UDTF**
Precision	Achieved an average precision of 91.62%. The model performs quite effectively and achieves similar results as the UDTF.	Achieved an average precision of 92.93%. Slightly improving the precision but not something significant.
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 easy to write, making it easier to understand and maintain, especially for complex algorithms.

Table 2: Comparative analysis of Sentiment Analysis implementations.

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.

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 predictable changes in 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 expected 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 choosing the right warehouse sizes based on data volume and computational needs to optimize performance and cost.

References

[1] TPC-H Sample Data. URL https://docs.snowflake.com/en/user-guide/sample-data-tpch.

[2] TPC-H documentation. URL https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf.

[3] TPC-H official website. URL https://www.tpc.org/.

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

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

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

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

[8] Create a warehouse in Snowflake. URL https://docs.snowflake.com/en/sql-reference/sql/create-warehouse.

[9] Writing a UDTF in Python. URL https://docs.snowflake.com/en/developer-guide/udf/python/udf-python-tabular-functions.

[10] Stanford University guide for Naïve Bayes classification algorithm. URL https://web.stanford.edu/class/cs124/lec/naivebayes2021.pdf.