Big Data Analytics in Business
Project

Name: Jenocent Edwardraj

Duration: 06/01/2025 – 07/08/2025

Contents

1. **Project Initiation** 2
2. **Analysis, Design, Environment Setup** 3
3. **Small Data Solution** 5
4. **Big Data Solution** 6
5. **Reflection / Learnings from the project** 8

1. Project Initiation

Project Topic

Sentiment Analysis of Customer Feedback

Project Description

This project involves analyzing customer feedback (e.g., product reviews, support messages) to determine overall sentiment: positive, neutral, or negative. The goal is to identify key themes in textual customer interactions and use Natural Language Processing (NLP) and machine learning models to classify sentiment, enabling actionable business insights.

Key Variables

- **Y variable:** Sentiment_Score (e.g., 0 = Negative, 1 = Neutral, 2 = Positive)
- **X variables:** Feedback_Text, Feedback_Length, Response_Time, Customer_Segment, Interaction_Channel

Why did you choose this topic?

I chose this topic because it introduces NLP-based sentiment modeling, which is not yet reflected in my current portfolio. It gives me an opportunity to work with unstructured textual data, an essential skill in both business analytics and broader data science.

How does this project align with your future professional goals?

Sentiment analysis is widely used in marketing, customer success, and product feedback loops. As a Data Science major with a business analytics concentration, this project directly strengthens my ability to deliver insights from customer-centric data in industry roles, especially in tech, consulting, or product analytics.

What ML algorithm(s) are you considering to use in this project?

I plan to use **TF-IDF with Logistic Regression**, **Naive Bayes**, and **Random Forest**. If time permits, I'll also explore **Spark NLP** for a scalable solution in PySpark.

What skills you expect to learn/strengthen through this project?

- PySpark MLlib pipeline construction
- NLP (tokenization, vectorization, text classification)
- Big Data processing and distributed training
- Sentiment modeling and interpretability
- Real-world data wrangling and model evaluation

2. Analysis, Design, Environment Setup

Check the small and big datasets.
Do you feel that the synthetic values generated for all columns (variables) are reasonable?

Yes, the synthetic values are reasonable. The Feedback_Text contains realistic short review-style comments, the Sentiment_Score values follow a reasonable distribution (more positives), and Response_Time, Customer_Segment, and Interaction_Channel reflect real-world diversity. The dataset also includes a few outliers and null values by design, which adds realism and allows for testing data cleaning.

Which is your Y variable that you want to predict?
Sentiment_Score (0 = Negative, 1 = Neutral, 2 = Positive)
Which are the X variables that you plan to use to predict Y?

Feedback_Text, Feedback_Length, Response_Time, Customer_Segment, and Interaction_Channel

Go through the solution code generated by ChatGPT.
Briefly explain the step-by-step flow of the code.

1. Load the CSV data into a Spark DataFrame.
2. Display a sample of the dataset and perform exploratory data analysis.
3. Clean the data by dropping null values.
4. Tokenize the feedback text and convert it into numerical vectors using CountVectorizer.
5. Convert the Sentiment_Score into a numeric label using StringIndexer.
6. Split the data into training and testing datasets.
7. Train two classification models: Logistic Regression and Naive Bayes.
8. Predict sentiment and evaluate both models using accuracy.
9. Output prediction samples and recommend deploying the pipeline to the big dataset in later steps.

Do you feel ChatGPT generated reasonable code?

Yes, the code is well-structured, complete, and applicable to a real sentiment classification pipeline using PySpark. It also includes logical preprocessing, model training, and evaluation steps.

Write code to generate descriptive statistics for the small dataset. Copy-paste the descriptive statistics of the small dataset here.

```
import pandas as pd

df = pd.read_csv("sentiment_small_dataset.csv")

df.describe(include='all')
```

```
[2] from google.colab import files
    uploaded = files.upload()
```

Choose Files sentiment_..._dataset.csv

- **sentiment_small_dataset.csv**(text/csv) - 5242463 bytes, last modified: 6/6/2025 - 100% done

```
Saving sentiment_small_dataset.csv to sentiment_small_dataset.csv
```

```
import pandas as pd
df = pd.read_csv("sentiment_small_dataset.csv")
df.describe(include='all')
```

	Sentiment_Score	Feedback_Text	Feedback_Length	Response_Time	Customer_Segment	Interaction_Channel	Survey_ID	Time_Stamp	Country
count	50000.000000	50000	50000.00000	49495.000000	50000	50000	50000	50000	50000
unique	NaN	12	NaN	NaN	3	3	50000	49901	5
top	NaN	Needs improvement	NaN	NaN	Standard	Chat	a3db378c-0fd0-4352-9bc1-e3b3719582d1	2024-04-04 13:14:02	India
freq	NaN	4261	NaN	NaN	16898	16713	1	2	10073
mean	1.308440	NaN	119.20576	30.543744	NaN	NaN	NaN	NaN	NaN
std	0.779767	NaN	1004.05039	17.076530	NaN	NaN	NaN	NaN	NaN
min	0.000000	NaN	10.00000	1.000000	NaN	NaN	NaN	NaN	NaN
25%	1.000000	NaN	16.00000	15.710000	NaN	NaN	NaN	NaN	NaN
50%	2.000000	NaN	17.00000	30.520000	NaN	NaN	NaN	NaN	NaN
75%	2.000000	NaN	20.00000	45.380000	NaN	NaN	NaN	NaN	NaN
max	2.000000	NaN	10000.00000	60.000000	NaN	NaN	NaN	NaN	NaN

3. Small Data Solution

Which IDE are you using to build and test the Small data solution? Google Colab or a different environment?

Google Colab

Did you successfully run your code in the small data environment?

Yes, I successfully ran the full PySpark ML pipeline in Google Colab using the small dataset from my GCP bucket.

Which models did you train and evaluate (e.g. linear regression, random forest etc.)

- Logistic Regression
- Naive Bayes

What evaluation metric you used? (e.g. F1-score, AUC, RMSE, MSE)

I used both Accuracy and F1-score via PySpark's MulticlassClassificationEvaluator.

List (or copy-paste) the model evaluation results (e.g. model 1 -> F1: 0.46, model 2 -> F1: 0.64)

Logistic Regression Accuracy: 0.5113

Naive Bayes Accuracy: 0.5113

Logistic Regression F1 Score: 0.3460

Naive Bayes F1 Score: 0.3460

Based on the small data set, which algorithm/model seems to be performing the best?

Both models performed identically in this run. Logistic Regression and Naive Bayes had the same accuracy and F1 scores.

Are you happy with the "best" model? Or you would prefer to check if there are better models?

I would prefer to explore better models. The current scores are low, which suggests that a more advanced algorithm (e.g. Random Forest, Gradient Boosted Trees) or improved feature engineering may be needed.

Briefly share the challenges you faced in this step

The main challenge was loading the CSV file from GCS into Spark within Google Colab. Initially, I tried using a gs:// path, which failed due to the lack of a GCS-compatible file system in Colab's Spark environment. I resolved this by using the gcsfs library to download the file locally. Additionally, I had to ensure the pipeline structure was correct and avoid reusing the same pipeline object across both models.

4. Big Data Solution

Did you make any change to the PySpark code after moving it from Google Colab to GCP Dataproc Jupyter? If yes, briefly mention the changes made to the code.

Yes, I made a few necessary changes to adapt the code for GCP Dataproc:

1. **Removed Google Colab-specific authentication code**: I deleted the line from google.colab import auth and auth.authenticate_user() since Dataproc already has direct access to Google Cloud Storage and does not require manual authentication.
2. **Removed the gcsfs download block**: In Colab, I used gcsfs to manually download the CSV file from GCS into the local Colab environment. In Dataproc, I directly accessed the CSV file using its gs:// path inside the spark.read.csv() function, so the extra download logic and gcsfs library were no longer needed.
3. **Commented out the !pip install lines**: Since PySpark is pre-installed on Dataproc clusters, I removed the !pip install pyspark gcsfs line.

These small adjustments ensured the same PySpark pipeline ran smoothly in the Dataproc environment.

List (or copy-paste) the model evaluation results for SMALL DATA (e.g. model 1 -> F1: 0.46, model 2 -> F1: 0.64)

[14]:
```
# Model Evaluation (Accuracy + F1)

# Accuracy
evaluator = MulticlassClassificationEvaluator(labelCol='label', predictionCol='prediction', metricName='accuracy')
lr_accuracy = evaluator.evaluate(lr_predictions)
nb_accuracy = evaluator.evaluate(nb_predictions)

print(f"Logistic Regression Accuracy: {lr_accuracy:.4f}")
print(f"Naive Bayes Accuracy: {nb_accuracy:.4f}")

# F1 Score
f1_eval = MulticlassClassificationEvaluator(labelCol='label', predictionCol='prediction', metricName='f1')
lr_f1 = f1_eval.evaluate(lr_predictions)
nb_f1 = f1_eval.evaluate(nb_predictions)

print(f"Logistic Regression F1 Score: {lr_f1:.4f}")
print(f"Naive Bayes F1 Score: {nb_f1:.4f}")
```

```
Logistic Regression Accuracy: 0.5113
Naive Bayes Accuracy: 0.5113
[Stage 47:============================>                            (1 + 1) / 2]
Logistic Regression F1 Score: 0.3460
Naive Bayes F1 Score: 0.3460
```

List (or copy-paste) the model evaluation results for BIG DATA (e.g. model 1 -> F1: 0.46, model 2 -> F1: 0.64)

[12]:
```
# Model Evaluation (Accuracy + F1)

# Accuracy
evaluator = MulticlassClassificationEvaluator(labelCol='label', predictionCol='prediction', metricName='accuracy')
lr_accuracy = evaluator.evaluate(lr_predictions)
nb_accuracy = evaluator.evaluate(nb_predictions)

print(f"Logistic Regression Accuracy: {lr_accuracy:.4f}")
print(f"Naive Bayes Accuracy: {nb_accuracy:.4f}")

# F1 Score
f1_eval = MulticlassClassificationEvaluator(labelCol='label', predictionCol='prediction', metricName='f1')
lr_f1 = f1_eval.evaluate(lr_predictions)
nb_f1 = f1_eval.evaluate(nb_predictions)

print(f"Logistic Regression F1 Score: {lr_f1:.4f}")
print(f"Naive Bayes F1 Score: {nb_f1:.4f}")
```

```
Logistic Regression Accuracy: 0.5000
Naive Bayes Accuracy: 0.5000
[Stage 44:============================>                            (1 + 1) / 2]
Logistic Regression F1 Score: 0.3333
Naive Bayes F1 Score: 0.3333
```

Based on the BIG dataset, which algorithm/model seems to be performing the best? Is it same as the best performing algorithm on the SMALL dataset.

Based on the BIG dataset, both **Logistic Regression** and **Naive Bayes** performed identically, with an **accuracy of 0.5000** and an **F1 score of 0.3333**.
Similarly, on the SMALL dataset, both models also showed **identical performance**, with an **accuracy of 0.5113** and an **F1 score of 0.3460**.

Thus, **no single model outperformed the other** on either dataset, both models performed equally on the SMALL and BIG datasets.

Briefly share the challenges you faced in this step

Some key challenges faced in this step included:

- **Code adaptation**: I had to remove Google Colab-specific code such as google.colab.auth and gcsfs because GCP Dataproc does not require or support these.
- **Kernel and environment setup**: Ensuring that the correct Python and Spark kernels were selected and compatible libraries were pre-installed.
- **Debugging missing modules**: Encountered ModuleNotFoundError when trying to run Colab-only commands like pip installs or local file writes.
- **File permission and gs:// errors**: I had to ensure that dataset paths were accurate and bucket permissions allowed Dataproc access.

5. Reflection / Learnings from the project

Briefly reflect and share your experience and learnings from this project.

This project provided a comprehensive and practical learning experience in the end-to-end design, implementation, and deployment of a machine learning (ML) pipeline using PySpark in both local (Google Colab) and distributed (GCP Dataproc) environments. From the beginning, the project required us to think through each phase of a real-world data science workflow, ranging from data ingestion, preprocessing, and exploration to model building, evaluation, and deployment. This hands-on approach was especially valuable in bridging the gap between theoretical knowledge and its actual application in a big data setting.

One of the most important lessons I learned was the importance of properly preparing and cleaning data before feeding it into any machine learning model. Even though this step may appear trivial at first, I quickly realized how a single column with an incorrect data type or a few missing values could break the entire pipeline. I learned how to use PySpark's versatile DataFrame API to handle missing values, cast data types, tokenize text data, and transform features efficiently in a distributed environment.

Working with text data also gave me insight into the unique preprocessing steps required for natural language processing (NLP). I became more comfortable with feature extraction techniques like tokenization, count vectorization, and string indexing, all of which are essential for turning raw customer feedback into a usable format for machine learning models. This process deepened my appreciation for the work that goes into preparing textual data, which is often messy and unstructured.

Modeling was another critical area where I gained practical experience. Comparing Logistic Regression and Naive Bayes side-by-side on the same datasets (small and large) taught me how to interpret model metrics such as accuracy and F1-score in the context of classification tasks. While the models performed similarly in this project, the process of building and evaluating them using Spark’s pipeline API helped me understand how modular and scalable ML development can be when using the right tools.

Deploying the same pipeline on GCP Dataproc introduced an entirely different set of challenges and learning opportunities. I faced several technical hurdles related to GCS path formatting, authentication differences between Colab and Dataproc, and missing modules like google.colab. I had to adapt the code to remove platform-specific lines and reconfigure the file loading process using Spark’s native capability to access data in cloud buckets. This taught me the importance of writing portable, cloud-agnostic code, an essential skill in real-world data engineering and machine learning workflows.

Furthermore, managing Spark kernels and understanding how distributed computing changes resource handling gave me a deeper insight into how big data infrastructure is configured and optimized. It was interesting to see that even though the dataset scaled up significantly, Spark's performance remained stable thanks to its parallel processing architecture.

Ultimately, this project helped me gain confidence in using PySpark for machine learning and provided a real glimpse into what production-grade data pipelines might look like in a cloud environment. I now better understand the workflow from ingesting raw data to training and evaluating ML models at scale. It also reinforced the value of problem-solving and debugging skills, especially when transitioning between different computing environments. These are foundational lessons that will carry forward into any advanced data science or engineering role I pursue in the future.