Project Final Report:

Employee Reviews Application

Spring2024

Supervised By: Pr.Asmae Mourhir

Adnane Kesraoui

Table of Contents

Project Inception 6

Business case Description 6

Business value of using ML: 6

Feasibility Analysis 6

Literature Review 6

Model Choice 7

Uses 7

Direct Use 7

Out of Scope Use 8

Bias, Risks, and Limitations 8

Recommendations 8

Training Data 9

Metrics for business goal eval and baseline 9

ML Pipeline Development - From a Monolith to a Pipeline 9

Ensuring ML Pipeline Reproducibility 9

Project structure definition and modularity 9

Code Versioning 9

Data versioning 10

Setting up the machine learning pipeline under an MLOps platform: 11

Pipeline Components 11

Data Validation and Verification 11

Preprocessing and Feature Engineering 12

Integration of model training and offline evaluation into the ML pipeline: 12

Development of Model Behavioral Tests 13

Code Snippets 14

Model Deployment 16

ML System Architecture 16

Application Deployment 16

Model Service Deployment 16
Front-end client Deployment 17
Integration and Deployment 18
Packaging and Containerization 18
Integration with CI/CD Pipeline 19
Hosting the application: 19
Model Serving and online testing 20
Model serving runtime 20
Serving Mode 20
Online Testing 20
Monitoring and Continual Learning 20
Resource Monitoring 20
Data Distribution Drift Monitoring 21
Continual Learning 23
Pipeline Orchestration 25
Conclusion 25
Summary of achievements 25
Lessons Learned: 25
Future Work: 26

Table of Figures:

Figure 1: Model Metrics Comparison 10
Figure 2: Data validation schema using Great Expectations 11
Figure 3: Reading, and Preprocessing steps. 12
Figure 4: Step to store the processed texts in a feature store. 13
Figure 5: Data Tests to ensure Input feature code is correct. 14
Figure 6:ML Infrastructure Test to ensure Integration with Feature storage. 14
Figure 7: Monitoring Test to check for Data Drift 14
Figure 8: Unit Tests 15
Figure 9: ML System Architecture for Employee Reviews. 16
Figure 10: API to access the containerized Model on Google Cloud. 17
Figure 11: Google Cloud Run Dashboard showcasing Model Deployment. 17
Figure 12: Employee Reviews Frontend App. 18
Figure 13: Dockerfile used for container building. 18
Figure 14: CI/CD Dashboard 19
Figure 15: Google Cloud Run hosting interface. 19
Figure 16: Google Cloud resource monitoring Dashboard. 21
Figure 17: Data Drift Logs. 21
Figure 18: Alert Creation Dashboard. 24
Figure 19: Email Receival. 24
Figure 20: Incident Reporting 24

Project Inception

Business case Description

The business problem we are addressing is the need for efficient and accurate analysis of employee feedback to improve workplace satisfaction and performance. Traditional methods of analyzing employee surveys are time-consuming and often fail to capture the nuanced sentiments expressed by employees. This project aims to develop a machine learning model that can automatically analyze employee reviews, categorizing them into positive, negative, and neutral sentiments. By doing so, organizations can gain actionable insights more quickly and accurately, allowing them to address issues proactively and foster a more positive work environment.

Business value of using ML:

Applying machine learning to this problem offers several benefits:

- Efficiency: ML models can process large volumes of data much faster than manual analysis, providing timely insights.
- Accuracy: Advanced NLP models can understand the context and sentiment of feedback more accurately than traditional methods.
- Scalability: The solution can easily scale to handle increasing amounts of data as the organization grows.
- Consistency: Automated analysis ensures consistent evaluation criteria, reducing the potential for human bias.
- Proactive Management: Timely insights enable organizations to address issues before they escalate, improving employee satisfaction and retention.

Feasibility Analysis

Literature Review

The field of Natural Language Processing (NLP) has seen significant advancements in recent years, particularly with the development of transformer-based models like BERT, RoBERTa, and their variants. These models have demonstrated state-of-the-art performance on various NLP tasks, including sentiment analysis. Research indicates that fine-tuning pre-trained transformer models on domain-specific data can lead to substantial improvements in accuracy and robustness for sentiment classification tasks. Key studies have shown the effectiveness of models such as DistilBERT and RoBERTa in handling the nuances of sentiment in textual data, making them suitable candidates for our employee feedback analysis.

Model Choice

This model leverages the power of RoBERTa, a robustly optimized transformer-based architecture, for sentiment analysis of employee reviews. The primary objective is to automatically analyze the sentiment expressed in written employee feedback, providing valuable insights into the overall sentiment of reviews.

- Developed by: Daniel Loureiro, Francesco Barbieri, Leonardo Neves, Luis Espinosa Anke, Jose Camacho-Collados
- Model type: Language model.
- Language(s) (NLP): en
- License: cc0-1.0
- Parent Model: cardiffnlp/twitter-roberta-base-2021-124m
- Resources for more information: https://arxiv.org/abs/1907.11692, https://arxiv.org/abs/2202.03829

Uses

Direct Use

The model is designed for direct use in sentiment analysis of written employee feedback without the need for fine-tuning or integration into a larger ecosystem/app.

Automatic Real-time Employee Feedback Gathering: As employees don't want to keep filling out long and redundant questionnaires, traditional surveys are becoming outdated. ML systems use efficient tools for real-time data collection.

Reliable Employee Feedback data: Ensure that the employees' opinions are not affected by bias, resulting in accurate insights.

Continuous Improvement: By leveraging ML for feedback analysis, organizations can have a clear picture of the areas where they are lacking and can further refine their employee workplaces continuously based on the data analysis feedback graphs.

Out of Scope Use

While the model is designed for sentiment analysis of employee reviews, there are certain use cases and scenarios for which it may not perform well. Out-of-scope uses include:

Financial Decision-Making: This model is not intended for making financial or business decisions solely based on sentiment analysis. Decisions involving significant financial implications should incorporate additional factors and expert judgment.

Legal Compliance: The model should not be solely relied upon for legal compliance decisions. It is not designed to provide guidance on legal matters or ensure compliance with employment laws and regulations.

High-Stakes HR Decisions: Critical HR decisions such as terminations or promotions should not be solely based on the model's sentiment analysis. Human judgment and comprehensive evaluation are essential in such cases.

Non-Workplace Sentiment Analysis: The model is tailored for employee reviews and may not generalize well to sentiment analysis in contexts unrelated to workplace feedback.

Bias, Risks, and Limitations

Significant research has explored bias and fairness issues with language models (see, e.g., Sheng et al. (2021) and Bender et al. (2021)). Predictions generated by the model may include disturbing and harmful stereotypes across protected classes; identity characteristics; and sensitive, social, and occupational groups.

Recommendations

To ensure responsible and ethical use of the Employee Review Model, the following recommendations are provided:

Human Oversight: Employ human oversight in decision-making processes influenced by the model's sentiment analysis. Human judgment is essential for understanding the broader context, addressing nuances, and making well-informed decisions.

Bias Awareness and Mitigation: Regularly assess and mitigate potential biases in the model's predictions, especially concerning protected classes, identity characteristics, and sensitive groups. Implement bias-aware evaluation metrics and address bias during model development and updates.

Continuous Model Monitoring: Implement a continuous monitoring system to track the model's performance over time. Regularly update the model to adapt to evolving language patterns and ensure its effectiveness in sentiment analysis.

Training Data

The model is trained on the "Employees Reviews Dataset". For further information, refer to the following data card: kmrmanish/Employees_Reviews_Dataset.

Metrics for business goal eval and baseline

Employee Satisfaction Index (ESI): A numerical representation of overall employee satisfaction derived from sentiment analysis.

Baseline: The initial ESI score serves as a benchmark. Post-implementation, changes in ESI can be compared against this baseline to assess the impact of the model.

ML Pipeline Development - From a Monolith to a Pipeline

Ensuring ML Pipeline Reproducibility

Project structure definition and modularity

The project follows a structured layout proposed by Cookiecutter Data Science to ensure modularity and organization. The main folder where data is stored is in the /data/validation folderr

Code Versioning

For code versioning, GitHub is the platform of choice, providing a robust platform for tracking changes, collaboration, and managing different versions of the project code. This ensures that all modifications are well documented and enables rolling back.

Data versioning

Data versioning is managed using Data Version Control (DVC), with Google Drive as the remote storage for datasets. This allows for efficient tracking of data changes while enabling collaboration by centralising data access.

Experiment tracking and model versioning:

MLFlow is utilized for experiment tracking and model versioning. This tool logs various metrics and parameters for each experiment, enabling comprehensive tracking and comparison of different models and their performance. The metadata store provided by MLFlow automatically captures and stores relevant information for data and model samples, ensuring reproducibility.

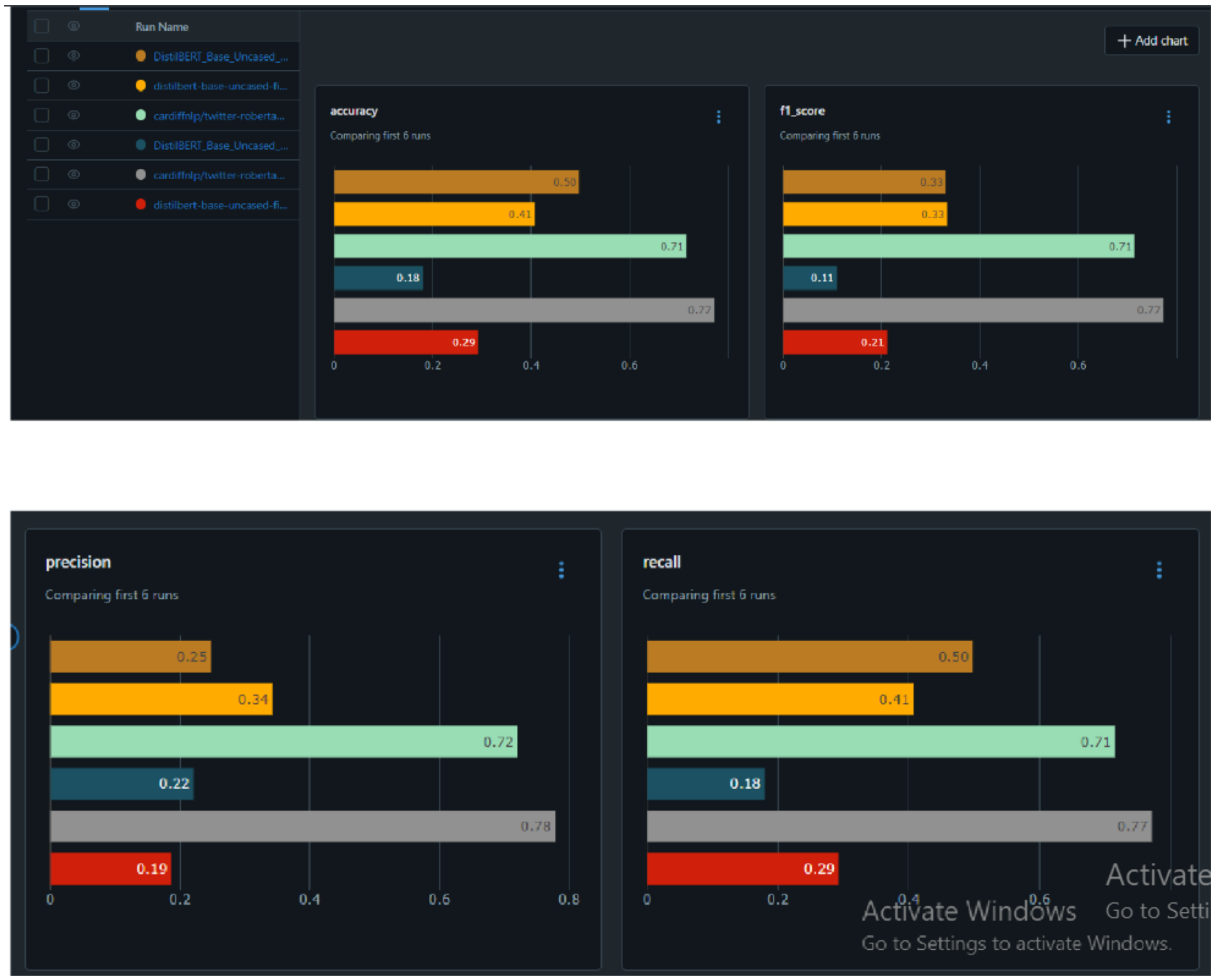


Figure 1: Model Metrics Comparison

Setting up the machine learning pipeline under an MLOps platform:

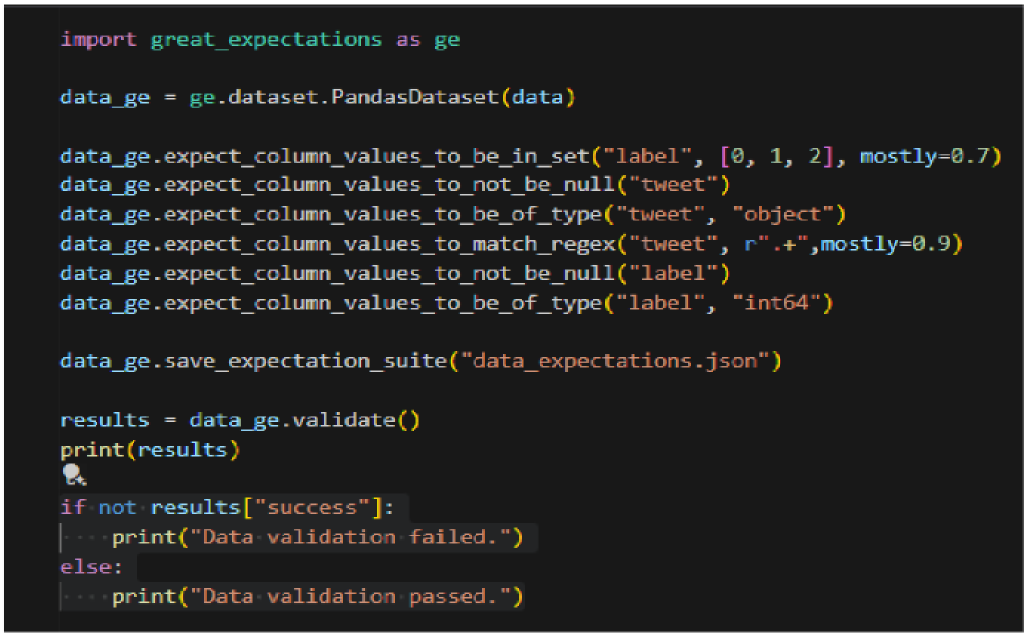
ZenML is used to set up the machine learning pipeline. It provides a framework for defining and managing the different steps of the ML workflow, from data ingestion and preprocessing to model training, evaluation, and deployment. This ensures a seamless integration and monitoring of the entire pipeline.

Pipeline Components

Data Validation and Verification

The data validation process is conducted using Great Expectations, a tool that allows for schema inference, anomaly detection, and data statistics generation. The process includes:

- Schema inference: Defining and testing expectations on the dataset to infer its schema indirectly.
- Anomaly checking: Running expectations checks and identifying any anomalies in the dataset. If any expectations fail, the output is logged, and corrective measures are taken.



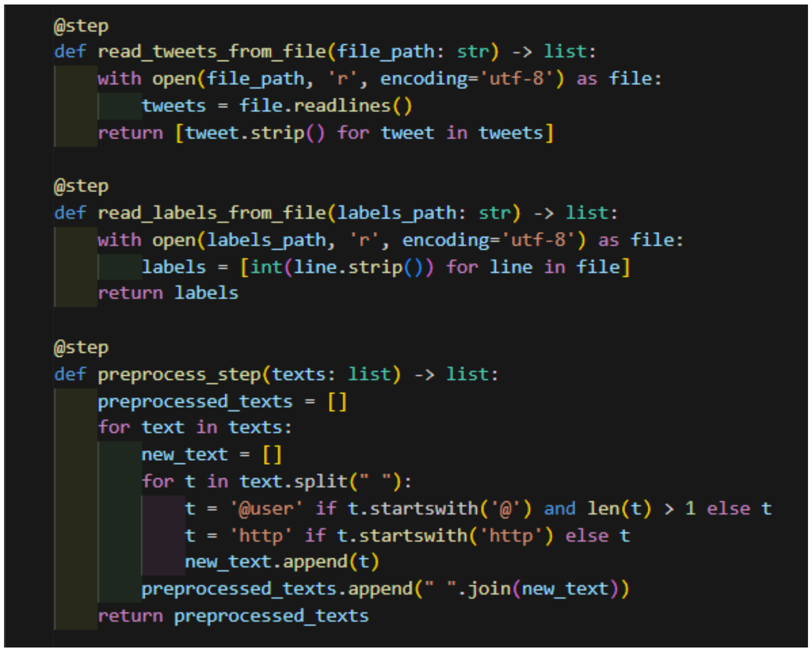
```python
import great_expectations as ge

data_ge = ge.dataset.PandasDataset(data)

data_ge.expect_column_values_to_be_in_set("label", [0, 1, 2], mostly=0.7)
data_ge.expect_column_values_to_not_be_null("tweet")
data_ge.expect_column_values_to_be_of_type("tweet", "object")
data_ge.expect_column_values_to_match_regex("tweet", r".+",mostly=0.9)
data_ge.expect_column_values_to_not_be_null("label")
data_ge.expect_column_values_to_be_of_type("label", "int64")

data_ge.save_expectation_suite("data_expectations.json")

results = data_ge.validate()
print(results)

if not results["success"]:
    print("Data validation failed.")
else:
    print("Data validation passed.")
```

Figure 2: Data validation schema using Great Expectations

Preprocessing and Feature Engineering

Preprocessing steps are implemented to ensure that the data is in the correct format for model training. This includes:

- Reading data: Using the TweetsDataset class to handle data ingestion.
- Preprocessing: Tokenizing and preparing the text data using the Huggingface transformers library.
- Feature Storage: Storing Data in Cassandra Database to facilitate feature reusability.



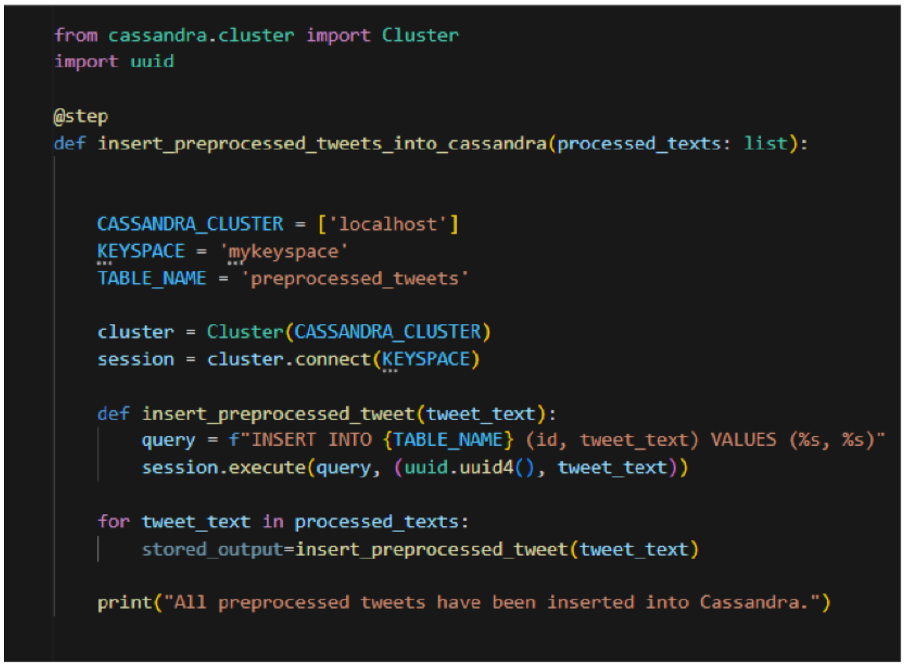
```python
@step
def read_tweets_from_file(file_path: str) -> list:
    with open(file_path, 'r', encoding='utf-8') as file:
        tweets = file.readlines()
    return [tweet.strip() for tweet in tweets]

@step
def read_labels_from_file(labels_path: str) -> list:
    with open(labels_path, 'r', encoding='utf-8') as file:
        labels = [int(line.strip()) for line in file]
    return labels

@step
def preprocess_step(texts: list) -> list:
    preprocessed_texts = []
    for text in texts:
        new_text = []
        for t in text.split(" "):
            t = '@user' if t.startswith('@') and len(t) > 1 else t
            t = 'http' if t.startswith('http') else t
            new_text.append(t)
        preprocessed_texts.append(" ".join(new_text))
    return preprocessed_texts
```

Figure 3: Reading, and Preprocessing steps.

Integration of model training and offline evaluation into the ML pipeline:

The model training and evaluation are integrated into the ZenML pipeline, ensuring that each step is logged and monitored:

- Model training: Utilizing the Huggingface transformers library to import and fine-tune the chosen model using the AdamW optimizer.
- Model evaluation: Assessing the model's performance using accuracy, precision, recall, and F1-score metrics. The results are logged in MLFlow for comparison and analysis.



```python
from cassandra.cluster import Cluster
import uuid

@step
def insert_preprocessed_tweets_into_cassandra(processed_texts: list):


    CASSANDRA_CLUSTER = ['localhost']
    KEYSPACE = 'mykeyspace'
    TABLE_NAME = 'preprocessed_tweets'

    cluster = Cluster(CASSANDRA_CLUSTER)
    session = cluster.connect(KEYSPACE)

    def insert_preprocessed_tweet(tweet_text):
        query = f"INSERT INTO {TABLE_NAME} (id, tweet_text) VALUES (%s, %s)"
        session.execute(query, (uuid.uuid4(), tweet_text))

    for tweet_text in processed_texts:
        stored_output=insert_preprocessed_tweet(tweet_text)

    print("All preprocessed tweets have been inserted into Cassandra.")
```

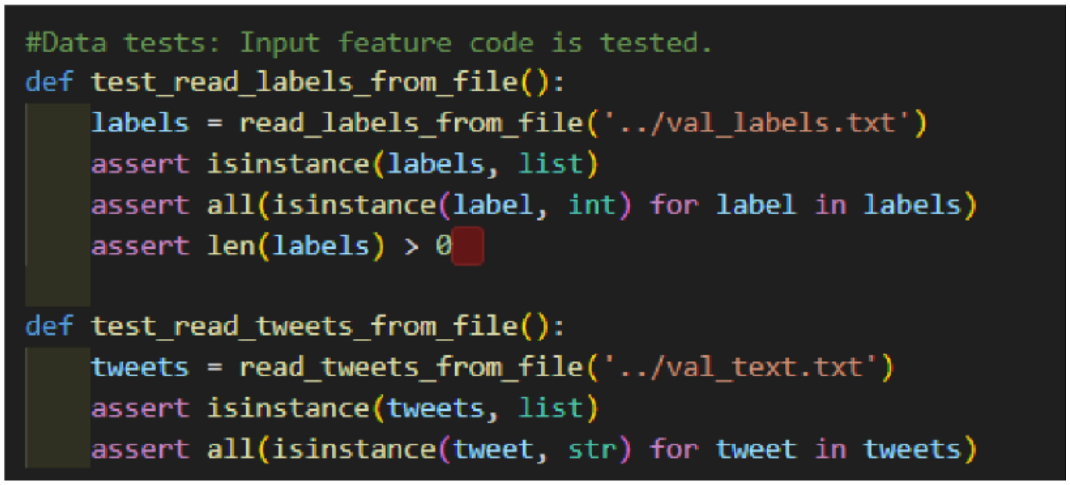
Figure 4: Step to store the processed texts in a feature store.

Development of Model Behavioral Tests

Behavioral testing is implemented using Pytest to ensure the reliability and accuracy of the data pipeline:

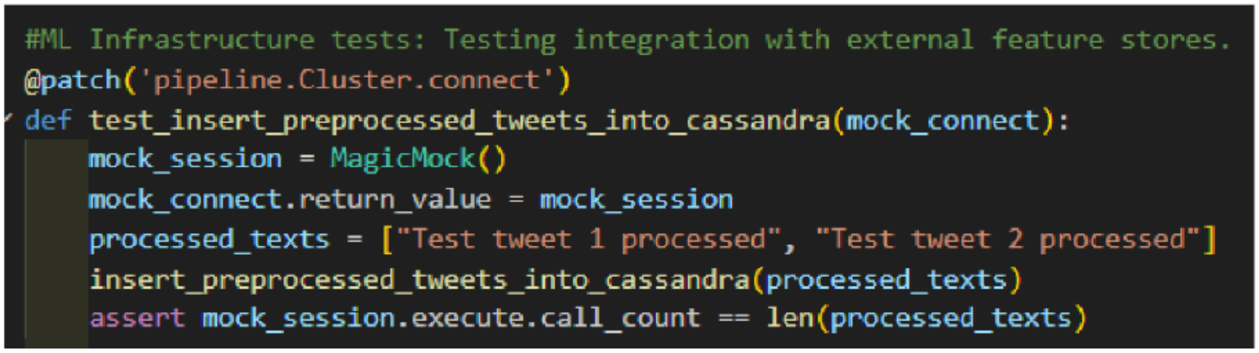
- Data Tests: Checking the functionality of reading labels and tweets from files.
- ML Infrastructure Tests: Mocking external dependencies, such as Cassandra database connections, to test the integration of the data pipeline.
- Unit Tests: Testing preprocessing, evaluation, and model inference steps to ensure correctness.
- Monitoring Tests: Checking for Data drifts amid production to prevent Model degradation.

Code Snippets



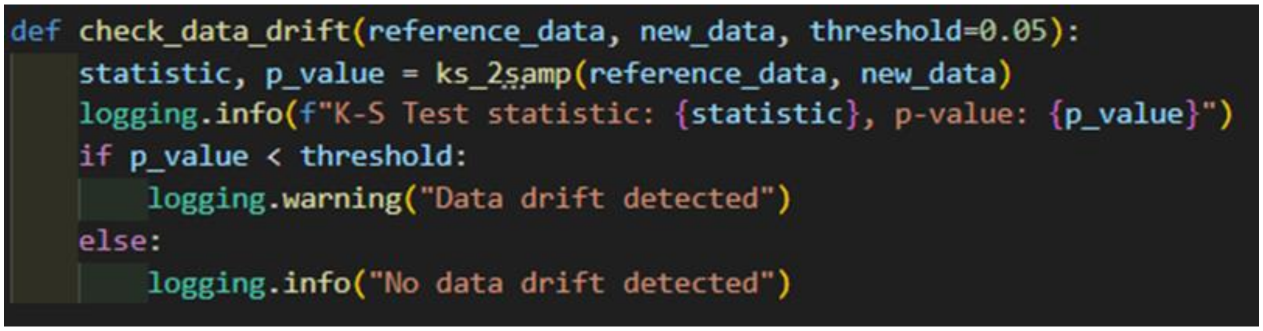
```python
#Data tests: Input feature code is tested.
def test_read_labels_from_file():
    labels = read_labels_from_file('../val_labels.txt')
    assert isinstance(labels, list)
    assert all(isinstance(label, int) for label in labels)
    assert len(labels) > 0

def test_read_tweets_from_file():
    tweets = read_tweets_from_file('../val_text.txt')
    assert isinstance(tweets, list)
    assert all(isinstance(tweet, str) for tweet in tweets)
```

Figure 5: Data Tests to ensure Input feature code is correct.



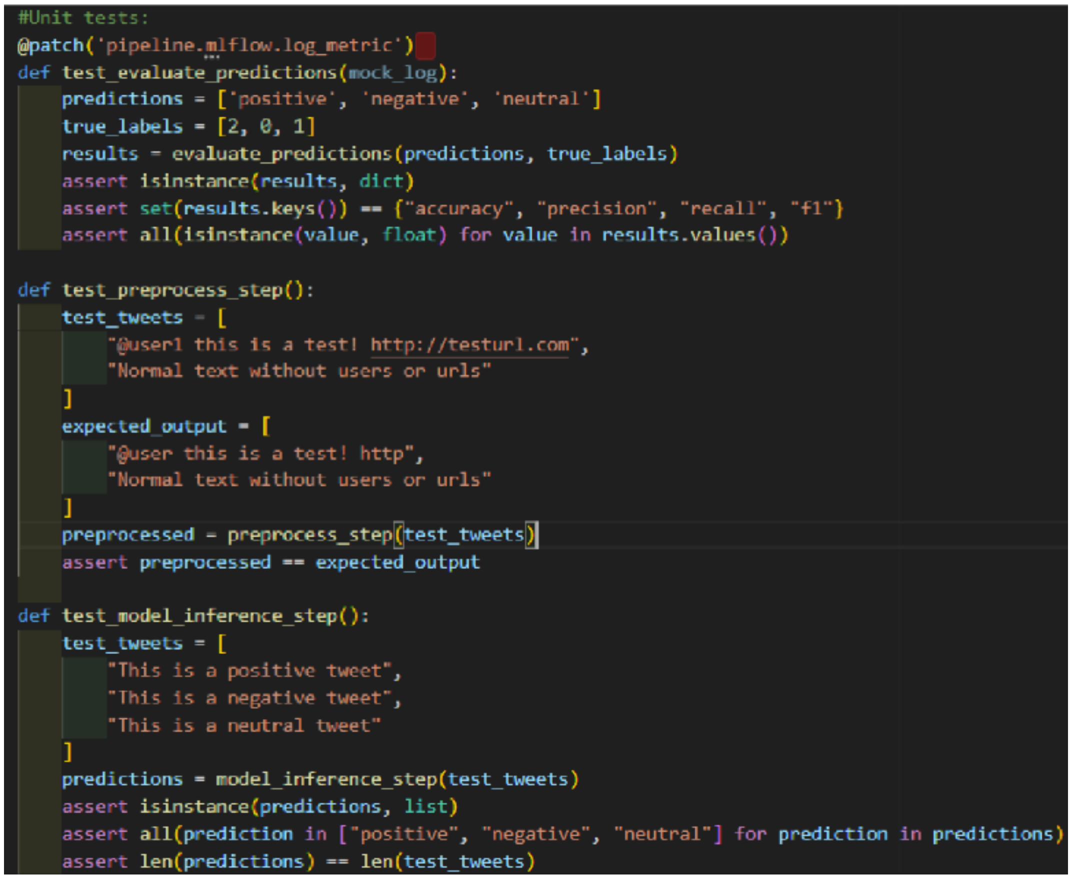
```python
#ML Infrastructure tests: Testing integration with external feature stores.
@patch('pipeline.Cluster.connect')
def test_insert_preprocessed_tweets_into_cassandra(mock_connect):
    mock_session = MagicMock()
    mock_connect.return_value = mock_session
    processed_texts = ["Test tweet 1 processed", "Test tweet 2 processed"]
    insert_preprocessed_tweets_into_cassandra(processed_texts)
    assert mock_session.execute.call_count == len(processed_texts)
```

Figure 6:ML Infrastructure Test to ensure Integration with Feature storage.



```python
def check_data_drift(reference_data, new_data, threshold=0.05):
    statistic, p_value = ks_2samp(reference_data, new_data)
    logging.info(f"K-S Test statistic: {statistic}, p-value: {p_value}")
    if p_value < threshold:
        logging.warning("Data drift detected")
    else:
        logging.info("No data drift detected")
```

Figure 7: Monitoring Test to check for Data Drift



```python
#Unit tests:
@patch('pipeline.mlflow.log_metric')
def test_evaluate_predictions(mock_log):
    predictions = ['positive', 'negative', 'neutral']
    true_labels = [2, 0, 1]
    results = evaluate_predictions(predictions, true_labels)
    assert isinstance(results, dict)
    assert set(results.keys()) == {"accuracy", "precision", "recall", "f1"}
    assert all(isinstance(value, float) for value in results.values())

def test_preprocess_step():
    test_tweets = [
        "@user1 this is a test! http://testurl.com",
        "Normal text without users or urls"
    ]
    expected_output = [
        "@user this is a test! http",
        "Normal text without users or urls"
    ]
    preprocessed = preprocess_step(test_tweets)
    assert preprocessed == expected_output

def test_model_inference_step():
    test_tweets = [
        "This is a positive tweet",
        "This is a negative tweet",
        "This is a neutral tweet"
    ]
    predictions = model_inference_step(test_tweets)
    assert isinstance(predictions, list)
    assert all(prediction in ["positive", "negative", "neutral"] for prediction in predictions)
    assert len(predictions) == len(test_tweets)
```

Figure 8: Unit Tests

Model Deployment

ML System Architecture

The ML system architecture includes different components which were used in order to build this full machine learning infrastructure, it regroups the different Technologies used throughout the whole finalized process.

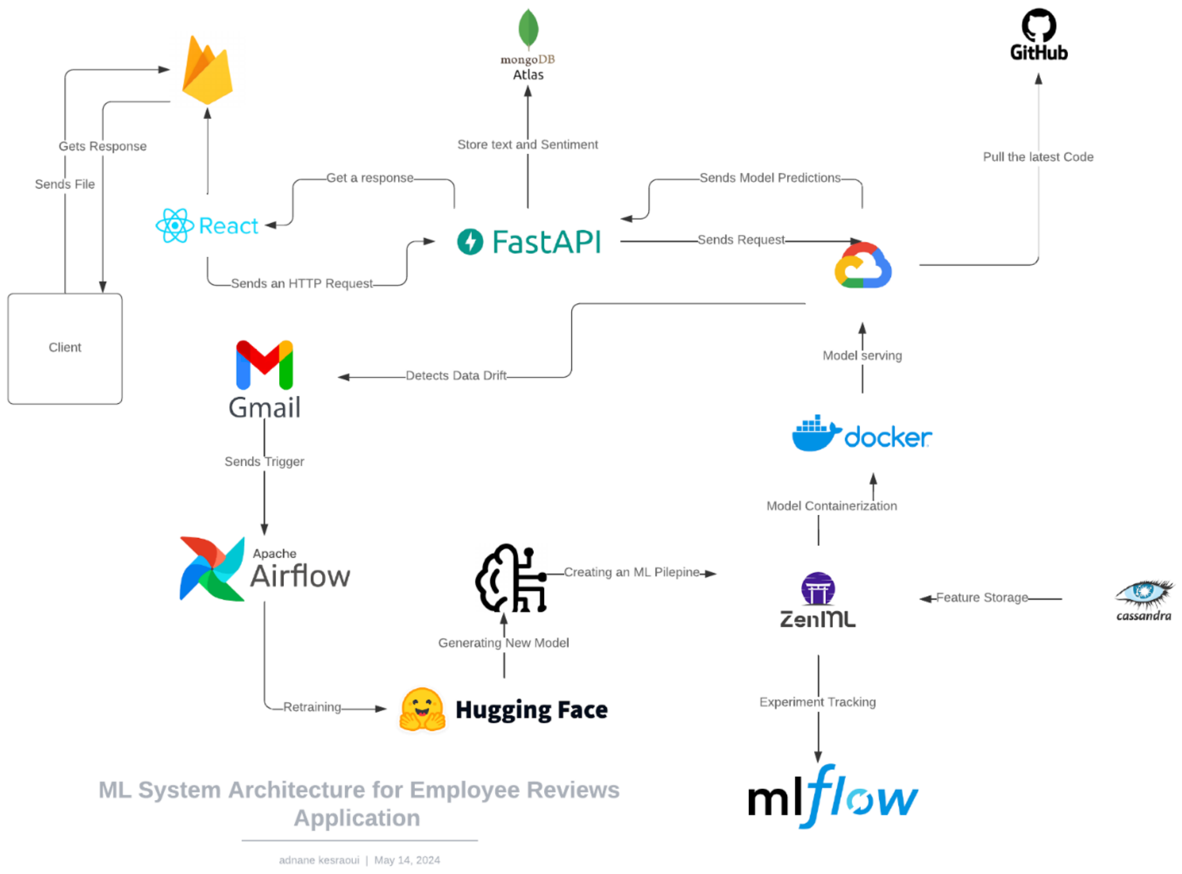


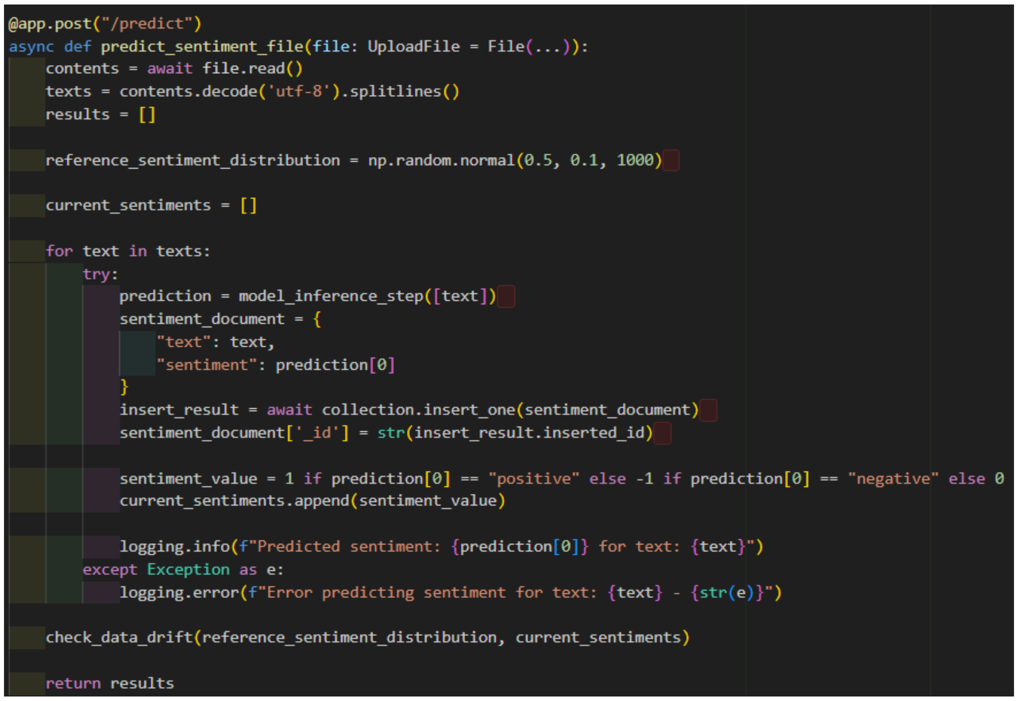
Figure 9: ML System Architecture for Employee Reviews.

Application Deployment

Model Service Deployment

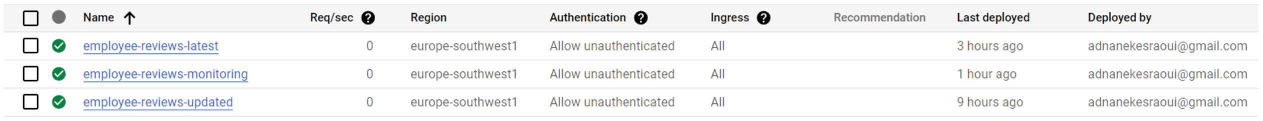
The model service is developed using FastAPI, providing a REST API that allows clients to interact with the model for inference. The API accepts input data in file format and returns predictions, facilitating easy integration with other systems.

The Model is then containerized using Docker and hosted serverless on Google Cloud.



```python
@app.post("/predict")
async def predict_sentiment_file(file: UploadFile = File(...)):
    contents = await file.read()
    texts = contents.decode('utf-8').splitlines()
    results = []

    reference_sentiment_distribution = np.random.normal(0.5, 0.1, 1000)

    current_sentiments = []

    for text in texts:
        try:
            prediction = model_inference_step([text])
            sentiment_document = {
                "text": text,
                "sentiment": prediction[0]
            }
            insert_result = await collection.insert_one(sentiment_document)
            sentiment_document['_id'] = str(insert_result.inserted_id)

            sentiment_value = 1 if prediction[0] == "positive" else -1 if prediction[0] == "negative" else 0
            current_sentiments.append(sentiment_value)

            logging.info(f"Predicted sentiment: {prediction[0]} for text: {text}")
        except Exception as e:
            logging.error(f"Error predicting sentiment for text: {text} - {str(e)}")

    check_data_drift(reference_sentiment_distribution, current_sentiments)

    return results
```

Figure 10: API to access the containerized Model on Google Cloud.



Name	Req/sec	Region	Authentication	Ingress	Recommendation	Last deployed	Deployed by
employee-reviews-latest	0	europe-southwest1	Allow unauthenticated	All		3 hours ago	adnanekesraoui@gmail.com
employee-reviews-monitoring	0	europe-southwest1	Allow unauthenticated	All		1 hour ago	adnanekesraoui@gmail.com
employee-reviews-updated	0	europe-southwest1	Allow unauthenticated	All		9 hours ago	adnanekesraoui@gmail.com

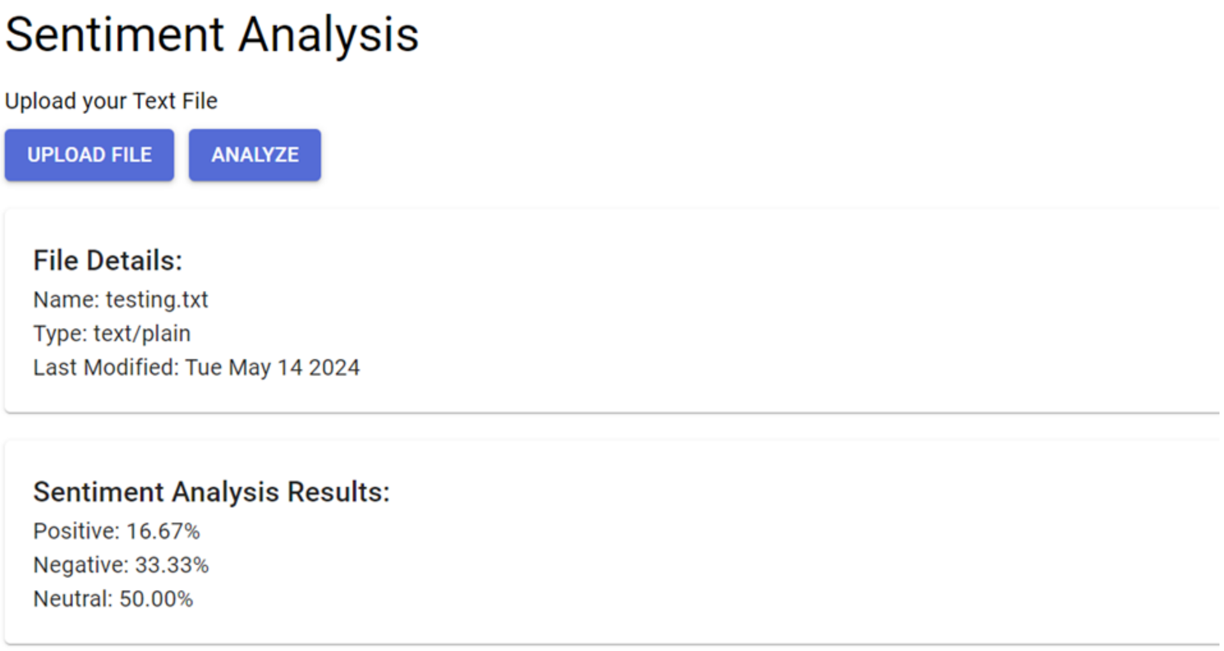
Figure 11: Google Cloud Run Dashboard showcasing Model Deployment.

Front-end client Deployment

The front-end client interface allows users to upload a text file and analyze its sentiment. Below is a screenshot of the front-end application:

The interface includes:

- Upload File Button: Allows users to select and upload a text file.
- Analyze Button: Initiates the sentiment analysis process on the uploaded file.
- File Details Section: Displays information about the uploaded file.
- Sentiment Analysis Results: Shows the percentage of positive, negative, and neutral sentiments detected in the text.



Sentiment Analysis

Upload your Text File

File Details:
Name: testing.txt
Type: text/plain
Last Modified: Tue May 14 2024

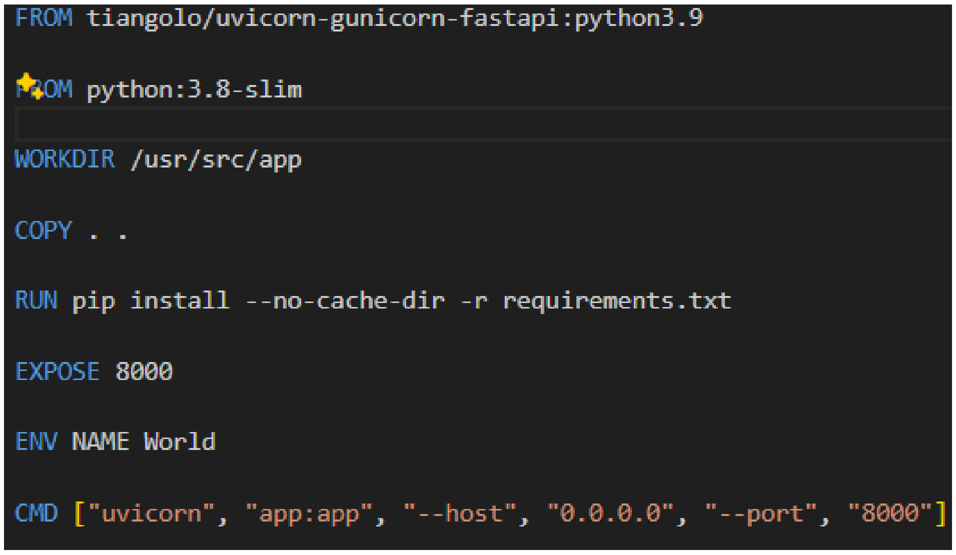
Sentiment Analysis Results:
Positive: 16.67%
Negative: 33.33%
Neutral: 50.00%

Figure 12: Employee Reviews Frontend App.

Integration and Deployment

Packaging and Containerization

The application is packaged and containerized using Docker. The Dockerfile includes the base image and necessary dependencies specified in the requirements.txt file. The Docker build command is used to create a Docker image of the application.

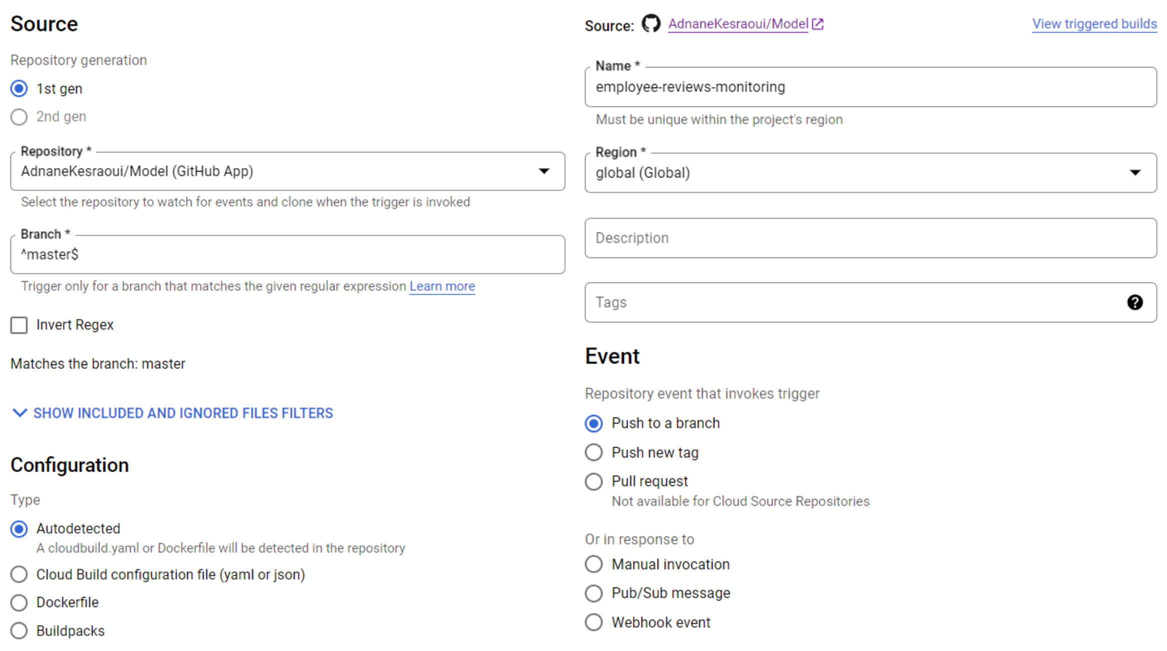


```
FROM tiangolo/uvicorn-gunicorn-fastapi:python3.9

FROM python:3.8-slim

WORKDIR /usr/src/app

COPY . .

RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 8000

ENV NAME World

CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]
```

Figure 13: Dockerfile used for container building.

Integration with CI/CD Pipeline

Google Cloud run enables its users to link their GitHub repositories with the deployed container and would configure it so that whenever new updates are pushed into the repo, the deployed container would Redeploy automatically with the updated version.



Source

Repository generation
- [✓ 1st gen]
- 2nd gen

Repository *: AdnaneKesraoui/Model (GitHub App)
Select the repository to watch for events and clone when the trigger is invoked

Branch *: ^master$
Trigger only for a branch that matches the given regular expression Learn more

- Invert Regex

Matches the branch: master

SHOW INCLUDED AND IGNORED FILES FILTERS

Configuration

Type
- [✓ Autodetected]
  A cloudbuild.yaml or Dockerfile will be detected in the repository
- Cloud Build configuration file (yaml or json)
- Dockerfile
- Buildpacks

Source: AdnaneKesraoui/Model — View triggered builds

Name *: employee-reviews-monitoring
Must be unique within the project's region

Region *: global (Global)

Description

Tags

Event

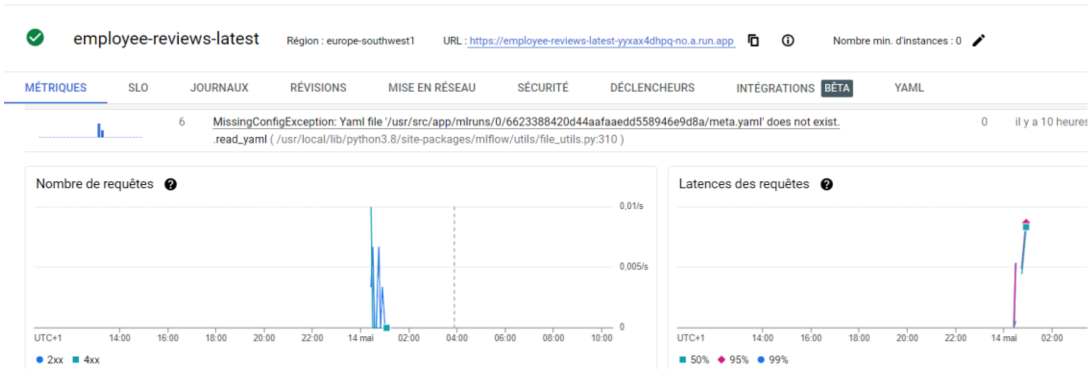
Repository event that invokes trigger
- [✓ Push to a branch]
- Push new tag
- Pull request
  Not available for Cloud Source Repositories

Or in response to
- Manual invocation
- Pub/Sub message
- Webhook event

Figure 14: CI/CD Dashboard

Hosting the application:

The application is deployed on Google Cloud Run, a serverless hosting platform. This provides scalable and cost-effective hosting for the containerized model API, ensuring high availability and performance.



employee-reviews-latest   Région : europe-southwest1   URL : https://employee-reviews-latest-yyxax4dhpq-no.a.run.app   Nombre min. d'instances : 0

```
6   MissingConfigException: Yaml file '/usr/src/app/mlruns/0/6623388420d44aafaaedd558946e9d8a/meta.yaml' does not exist.   0   il y a 10 heures
    .read_yaml ( /usr/local/lib/python3.8/site-packages/mlflow/utils/file_utils.py:310 )
```

Figure 15: Google Cloud Run hosting interface.

Model Serving and online testing

Model serving runtime

The Model is served Via Google Cloud Run while exposing its API endpoint with FastAPI to facilitate communication with other programs via Restful communication.

Serving Mode

The model is served in a batch mode, processing multiple requests at once. This mode allows the system to handle large volumes of data efficiently, providing sentiment analysis results for batches of employee feedback simultaneously. Users submit a text file to the system which then responds with the percentage of sentiments contained in that file.

Online Testing

Online testing involves running A/B tests to compare different versions of the model and optimize performance. The results of these tests are used to iteratively improve the model and ensure that it meets the desired performance criteria. Google Cloud Run Enables A/B testing through a Dashboard seamlessly.

Monitoring and Continual Learning

Resource Monitoring

Resource monitoring is essential to ensure the efficient use of computational resources during model inference and deployment. Google Cloud provides tools which are integrated into the system to monitor CPU and memory usage, latency, and throughput. Alerts are set up to notify the team if any resource usage exceeds predefined thresholds, enabling timely intervention to optimize performance and prevent potential bottlenecks.

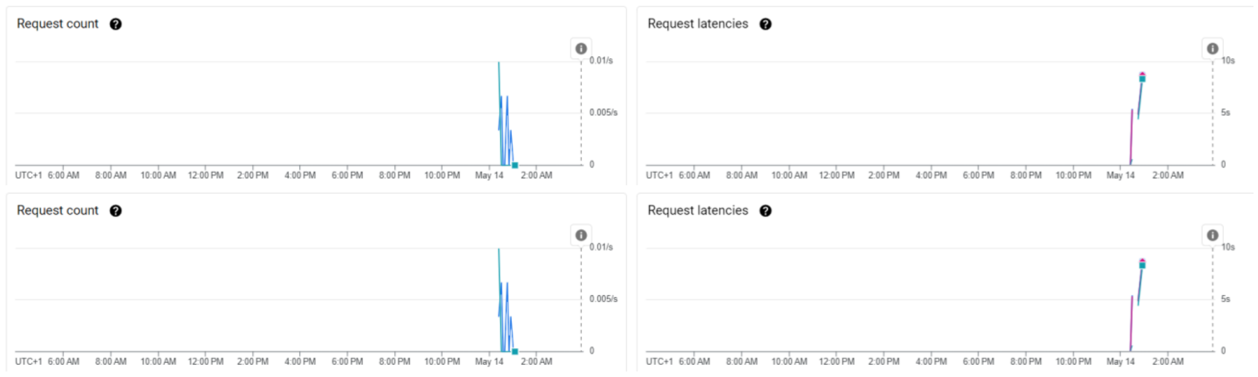
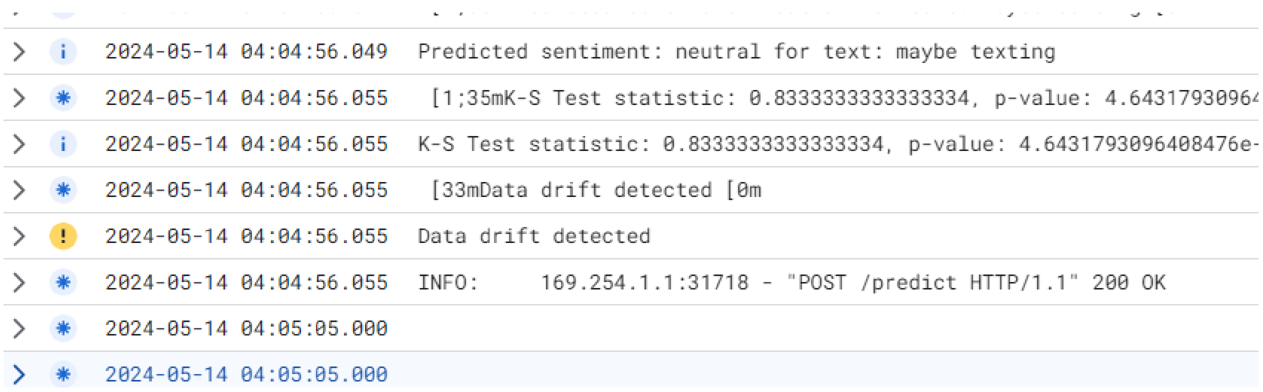


Figure 16: Google Cloud resource monitoring Dashboard.

Data Distribution Drift Monitoring

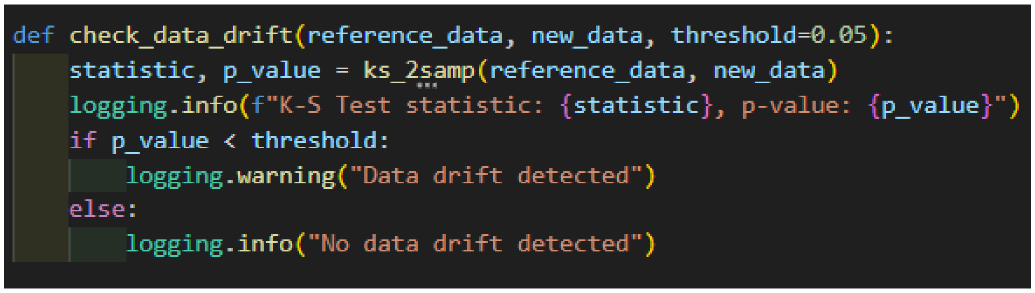
Continuous monitoring of model performance is implemented using Google Cloud Logging and Monitoring. Distribution drift monitoring is established using Kolmogorov-Smirnov (KS) tests to detect changes in the input data distribution that may affect model performance. If any significant drift is detected, logs are generated, and an alert is activated to run an Airflow automation. This automation triggers predefined workflows to address the data drift, such as retraining the model with updated data.



```
2024-05-14 04:04:56.049  Predicted sentiment: neutral for text: maybe texting
2024-05-14 04:04:56.055   [1;35mK-S Test statistic: 0.8333333333333334, p-value: 4.64317930964
2024-05-14 04:04:56.055  K-S Test statistic: 0.8333333333333334, p-value: 4.6431793096408476e-
2024-05-14 04:04:56.055   [33mData drift detected [0m
2024-05-14 04:04:56.055  Data drift detected
2024-05-14 04:04:56.055  INFO:     169.254.1.1:31718 - "POST /predict HTTP/1.1" 200 OK
2024-05-14 04:05:05.000
2024-05-14 04:05:05.000
```

Figure 17: Data Drift Logs.

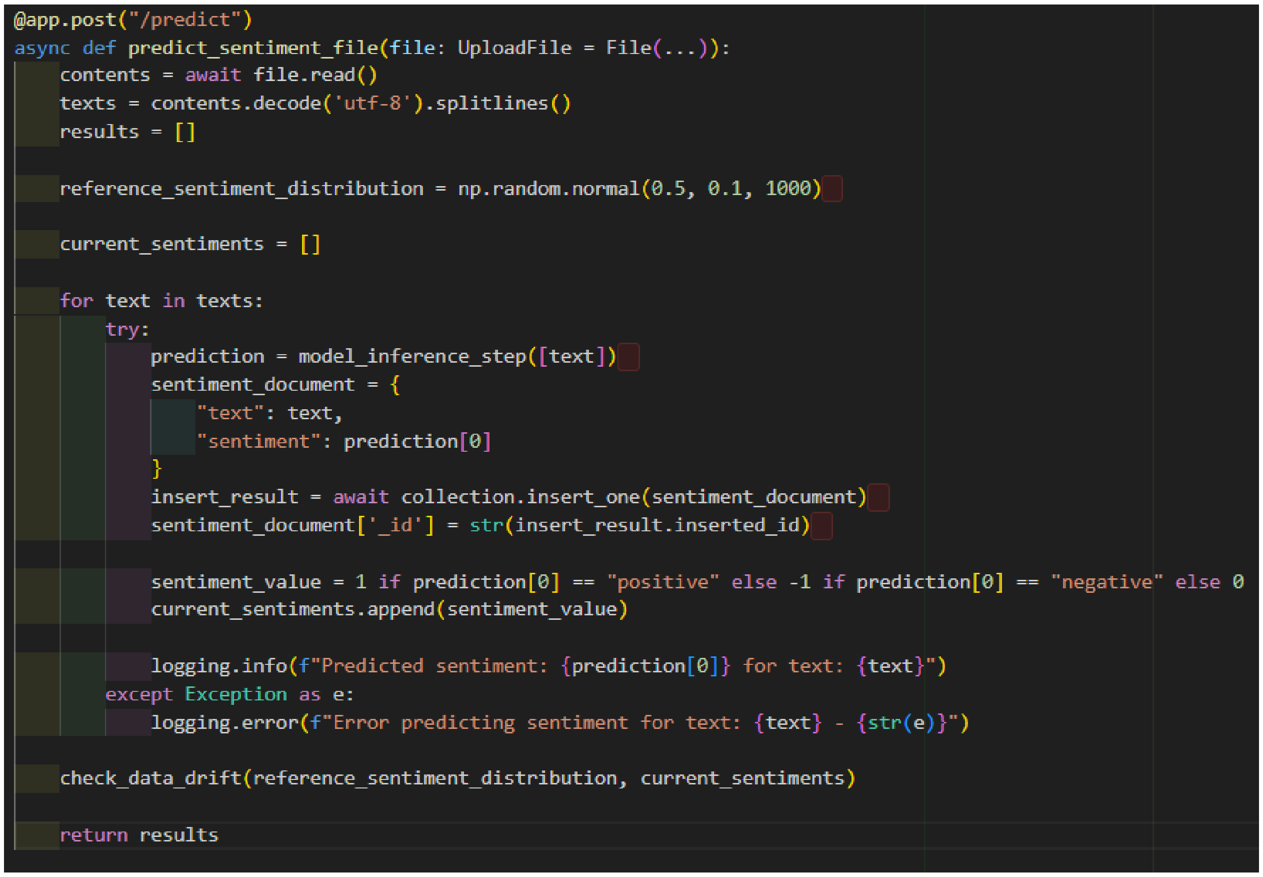
This is the function that checks for data drifts:



```python
def check_data_drift(reference_data, new_data, threshold=0.05):
    statistic, p_value = ks_2samp(reference_data, new_data)
    logging.info(f"K-S Test statistic: {statistic}, p-value: {p_value}")
    if p_value < threshold:
        logging.warning("Data drift detected")
    else:
        logging.info("No data drift detected")
```

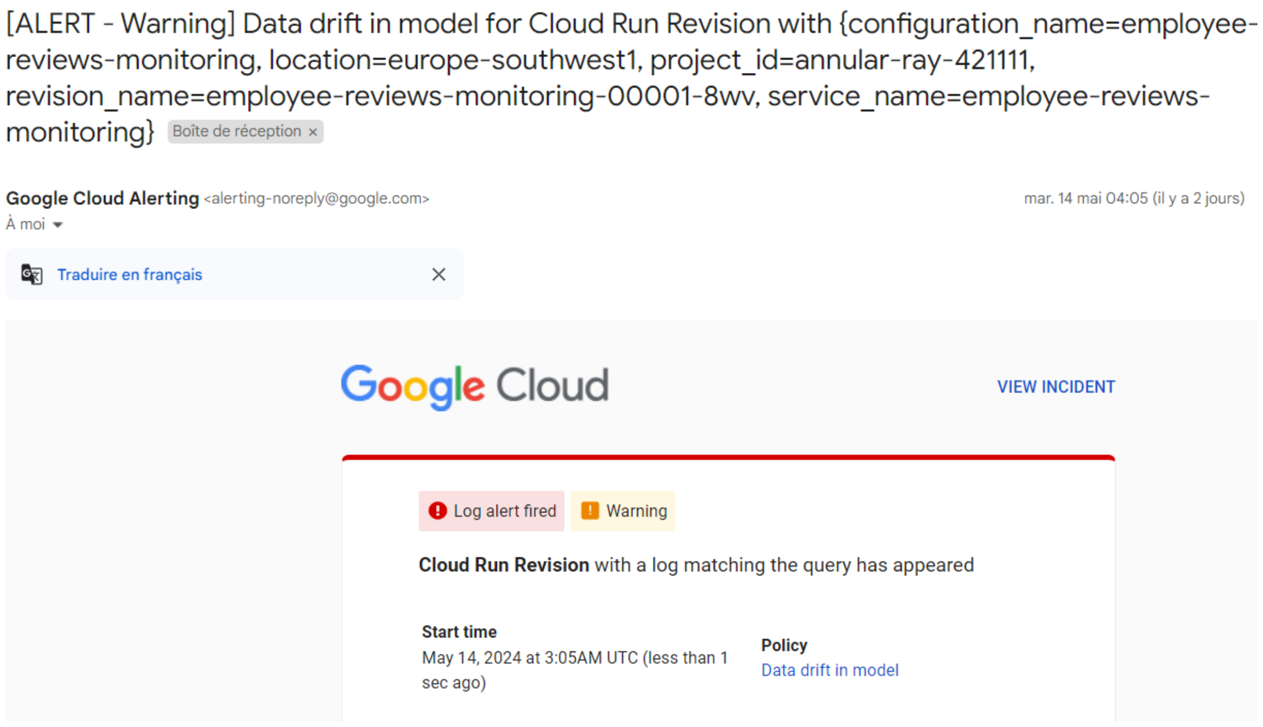
Given a provided data distribution, the function keeps tabs on the distribution provided and once the value of "p-value" drops bellow the threshold, sends a log warning.

This is the implementation of the data drift code, that's contained in the presentation:



```python
@app.post("/predict")
async def predict_sentiment_file(file: UploadFile = File(...)):
    contents = await file.read()
    texts = contents.decode('utf-8').splitlines()
    results = []

    reference_sentiment_distribution = np.random.normal(0.5, 0.1, 1000)

    current_sentiments = []

    for text in texts:
        try:
            prediction = model_inference_step([text])
            sentiment_document = {
                "text": text,
                "sentiment": prediction[0]
            }
            insert_result = await collection.insert_one(sentiment_document)
            sentiment_document['_id'] = str(insert_result.inserted_id)

            sentiment_value = 1 if prediction[0] == "positive" else -1 if prediction[0] == "negative" else 0
            current_sentiments.append(sentiment_value)

            logging.info(f"Predicted sentiment: {prediction[0]} for text: {text}")
        except Exception as e:
            logging.error(f"Error predicting sentiment for text: {text} - {str(e)}")

    check_data_drift(reference_sentiment_distribution, current_sentiments)

    return results
```

Due to not finding the training dataset data distribution values, a synthetic distribution generator was inputted to showcase the drift detection work. The Function tests the drift on each input and once the drift is detected sends a log warning to the google cloud logs which trigger an alert to send a message to a certain email account to notify the admin of such Problem.



[ALERT - Warning] Data drift in model for Cloud Run Revision with {configuration_name=employee-reviews-monitoring, location=europe-southwest1, project_id=annular-ray-421111, revision_name=employee-reviews-monitoring-00001-8wv, service_name=employee-reviews-monitoring} Boîte de réception ×

Google Cloud Alerting <alerting-noreply@google.com> mar. 14 mai 04:05 (il y a 2 jours)
À moi

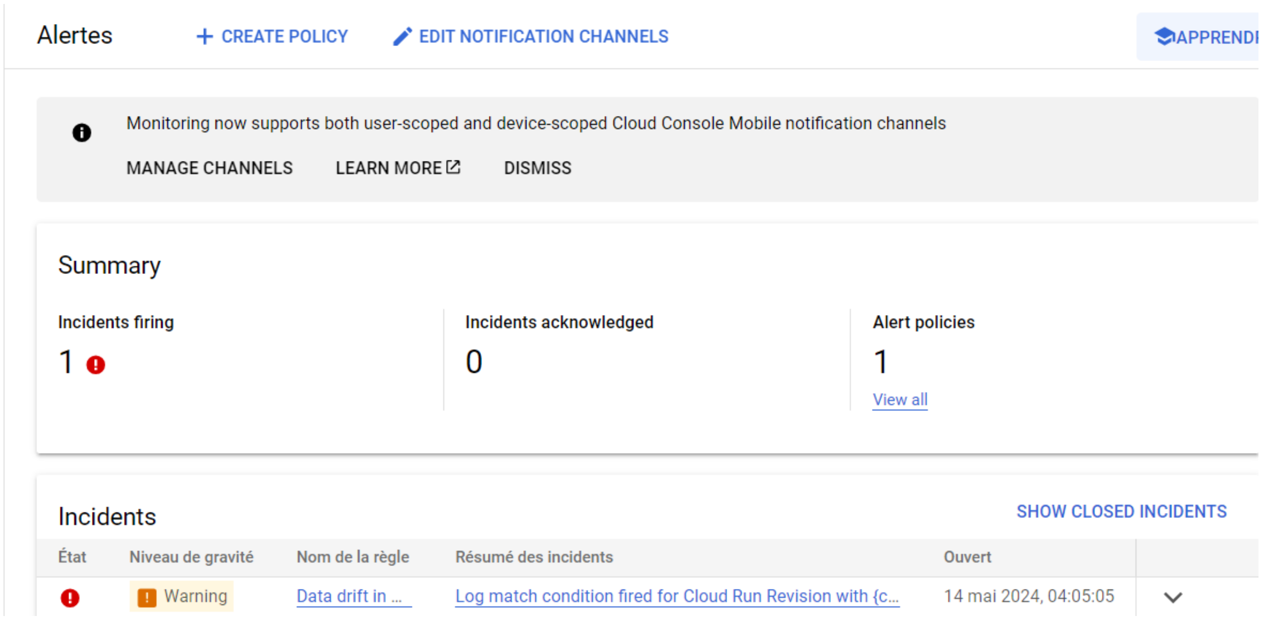
Cloud Run Revision with a log matching the query has appeared

Start time
May 14, 2024 at 3:05AM UTC (less than 1 sec ago)

Policy
Data drift in model

Continual Learning

The continual learning pipeline is designed to keep the model updated with new data and to prevent it from Degradation. This involves setting up a Continuous Training (CT) and Continuous Deployment (CD) pipeline using AirFlow. The pipeline automatically retrains the model with newly collected data when significant changes in data distribution are detected. The updated model is then deployed seamlessly into Google Cloud Run, ensuring that the system adapts to new trends and maintains high performance.



Monitoring now supports both user-scoped and device-scoped Cloud Console Mobile notification channels

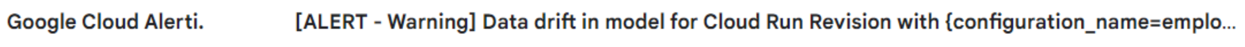
Summary

Incidents firing	Incidents acknowledged	Alert policies
1	0	1

Incidents

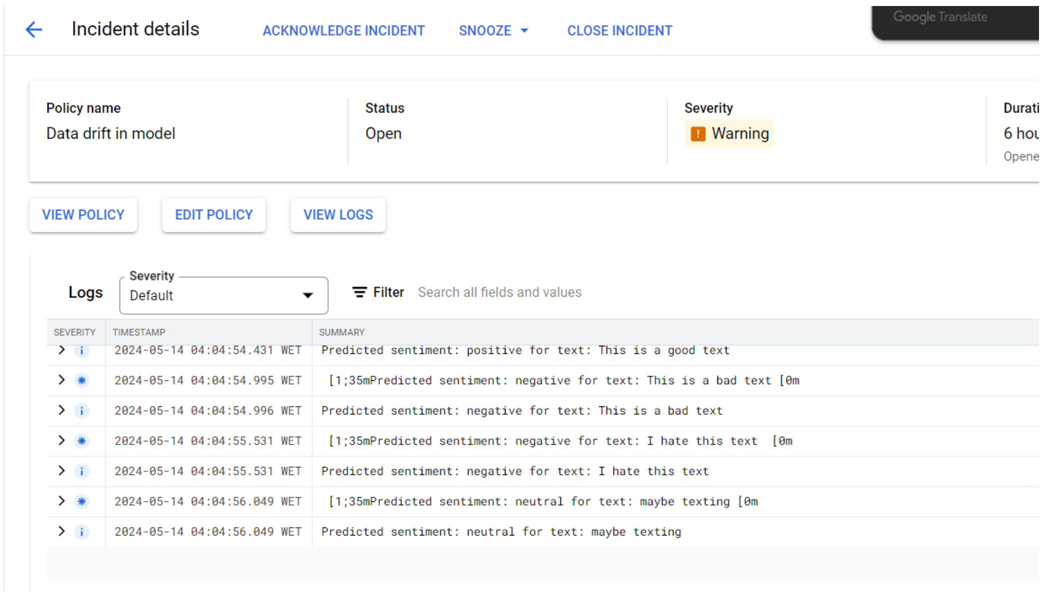
État	Niveau de gravité	Nom de la règle	Résumé des incidents	Ouvert
●	Warning	Data drift in ...	Log match condition fired for Cloud Run Revision with {c...	14 mai 2024, 04:05:05

Figure 18: Alert Creation Dashboard.



Google Cloud Alerti. [ALERT - Warning] Data drift in model for Cloud Run Revision with {configuration_name=emplo...

Figure 19: Email Receival.



Policy name	Status	Severity	Durati
Data drift in model	Open	Warning	6 hou Opene

Severity: Default

```
SEVERITY  TIMESTAMP                     SUMMARY
>  i      2024-05-14 04:04:54.431 WET   Predicted sentiment: positive for text: This is a good text
>  *      2024-05-14 04:04:54.995 WET    [1;35mPredicted sentiment: negative for text: This is a bad text [0m
>  i      2024-05-14 04:04:54.996 WET   Predicted sentiment: negative for text: This is a bad text
>  *      2024-05-14 04:04:55.531 WET    [1;35mPredicted sentiment: negative for text: I hate this text  [0m
>  i      2024-05-14 04:04:55.531 WET   Predicted sentiment: negative for text: I hate this text
>  *      2024-05-14 04:04:56.049 WET    [1;35mPredicted sentiment: neutral for text: maybe texting [0m
>  i      2024-05-14 04:04:56.049 WET   Predicted sentiment: neutral for text: maybe texting
```

Figure 20: Incident Reporting

Pipeline Orchestration

Pipeline Orchestration is done using Airflow, once an email is received a retraining process is triggered to get rid of the Data drift and redeploy.

Implementation

The implementation of the retraining pipeline was not implemented and deployed due to time constraints.

Conclusion

Summary of achievements

The project successfully developed and deployed a sentiment analysis model for employee feedback. Key achievements include:

- Establishing a reproducible ML pipeline using ZenML.
- Implementing robust data validation and preprocessing steps.
- Training and fine-tuning a high-performing sentiment analysis model using the Huggingface transformers library.
- Developing and deploying a front-end client and model service using FastAPI and Google Cloud Run.
- Setting up continuous monitoring, performance tracking, and a continual learning pipeline to ensure long-term model efficacy and adaptability.

Lessons Learned:

Several lessons were learned during the project:

- The importance of robust data validation and preprocessing to ensure high-quality input data.
- The value of continuous monitoring and feedback loops for maintaining model performance.
- The challenges of integrating different tools and platforms into a cohesive ML pipeline.
- Ways to implement ML pipelines, train models and Deploy full-fledged Applications from start to finish.

Future Work:

- Enhancing the front-end interface to provide more interactive and insightful visualizations.
- Exploring additional use cases for the sentiment analysis model, such as real-time feedback analysis during company meetings or events.
- Continuing to improve the model's explainability and interpretability, ensuring transparency and ethical use.