AIN-3009 Term Project: MLflow applied to sentiment classification AI model.

Bahçeşehir University

Artificial Intelligence Engineering Department

Mohamed Ali Maghmoum

May 2024

Abstract

this project involves the development, optimization, and deployment of a sentiment classification model using TensorFlow Keras, integrated with MLflow for comprehensive lifecycle management. The Main focus of the project is to showcase the cababilities of MLflow and how we can use it to help use develop our models.

The process began with the structured training of a TensorFlow Keras model, systematically logged and managed using MLflow to ensure reproducibility and efficient experiment tracking. Hyperparameter tuning was conducted via Keras Tuner, with MLflow documenting each configuration's performance, thus identifying the most effective model settings. Subsequently, the optimal model was deployed as a REST API, using MLflow's serving capabilities, which provided practical, real-time sentiment analysis applications. Post-deployment, the system included continuous performance monitoring to track model efficacy and detect any decline in performance over time. This setup not only ensures ongoing accuracy and reliability but also supports dynamic adjustments to the model, facilitating adaptive maintenance in response to evolving data trends. This report encapsulates the complete machine learning lifecycle, from conception through deployment and ongoing management, demonstrating a scalable approach to machine learning projects.

Keywords: model lifecycle management, hyperparameter tuning, model deployment, performance monitoring, experiment tracking.

Contents

1- Domain and Dataset

The model we will be developing will be a classification model that will work on a dataset of movie reviews, and will classify them based on the sentiment of the review. This is a very useful tool for all types of websites that allows users to write reviews about products listed on their site. Sentiment classification can also be used to detect unwanted and harmful reviews, in the case of targeted harassment, it's quite an effective tool that has many different uses.

Sentiment analysis is a critical area in natural language processing that involves determining the emotional tone behind a body of text. The dataset utilized for training and testing the sentiment classification model is derived from the NLTK library's "movie_reviews" dataset. This dataset comprises movie reviews labeled as either positive or negative, providing a binary classification challenge that is ideal for training deep learning models to detect sentiment. The reviews are pre-processed and vectorized to transform the textual data into a format suitable for the neural network. This dataset not only serves as a valuable resource for developing sentiment analysis models but also offers a balanced framework for evaluating the effectiveness of various machine learning techniques in text classification.

Table 1. Example of an entry on the Database

review	lebal
two teen couples go to a church party , drink and then drive . they get into an accident . one of the guys dies , but his girlfriend continues to see him in her life , and has nightmares . what ' s the deal ? watch the movie and " sorta " find out . . . critique : a mind - fuck movie for the teen generation that touches on a very cool idea , but presents it in a very bad package . which is what makes this review an even harder one to	neg

2- setting up the environment

for this project, I utilized the Databricks Community Edition environment as the foundational platform for development and execution. Databricks provides a powerful cloud-based environment that integrates with a variety of tools and libraries. The environment is best suited for this project due its seamless integration with Mlflow, it doesn't require additional installations or third-party hosts for the UI, and running and monitoring experiments is very easy and simple to do.

It is important to make sure you are running on a machine-learning environment and notebook, data-science and engineering environments don't have support for MLflow, so make sure you get the right

set up, for dependencies, you can install them into the notebook environment by running #pip commands, or using the UI to find your environment and look for packages to install, you may need to detach and reattach the environment for it to register your new libraries, but the user-friendly UI will let you know when you need to do that with a convenient pop-up.

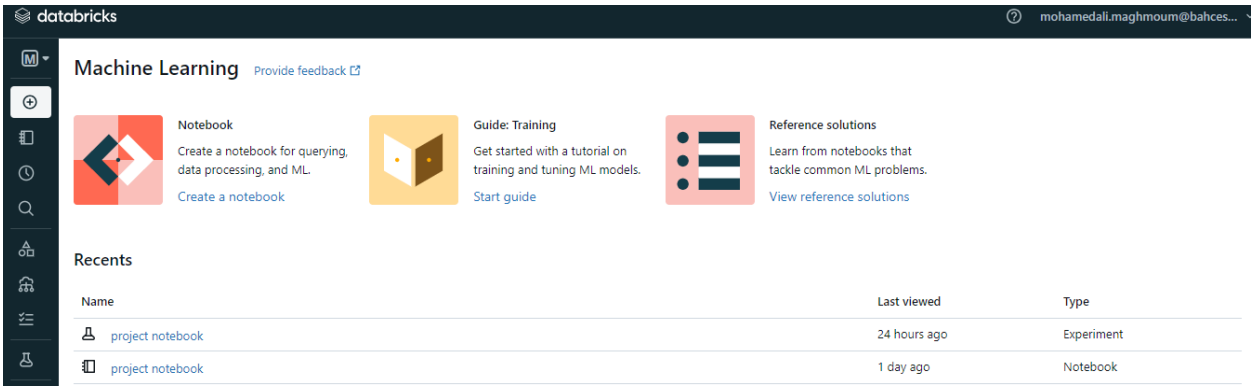

Figure 1 Databricks community environment UI

For our project dependencies, we have installed these libraries:

- **TensorFlow Keras**: TensorFlow is a well-known ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML, and developers easily build and deploy ML-powered applications. This library provides the needed framework for designing and training the sentiment analysis neural network models.
- **NLTK (Natural Language Toolkit):** NLTK is a leading platform for building Python programs to work with human language data (text). It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text-processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning. This toolkit is used for loading the movie review dataset and processing the text data.
- **Scikit-learn:** Scikit-learn is utilized here for its powerful preprocessing tools, which can be beneficial for encoding and vectorizing text data. It also offers robust tools for splitting the dataset into training and test sets.
- **Keras Tuner:** This is an easy-to-use, scalable hyperparameter tuning framework that solves the pain points of hyperparameter search. Integrated with TensorFlow, it lets you pick the optimal set of hyperparameters for your TensorFlow model, enhancing the model's performance by finding the best model configurations.
- **MLflow:** Integrated natively within Databricks, MLflow is used for managing the entire machine learning lifecycle, including experimentation, reproducibility, and deployment of ML models. It tracks experiments to record and compare parameters and results, manages and deploys models from diverse ML libraries to diverse deployment tools.
- **Pandas and NumPy:** These are fundamental packages for data manipulation and numerical computation, respectively.
- **Matplotlib:** Often used for creating static, interactive, and animated visualizations in Python, Matplotlib can be useful for plotting training histories, model performance metrics, and other relevant data visualizations to analyze the behavior and performance of the model over time.

- **Requests:** This library is used for sending HTTP requests using Python, which is particularly useful for interacting with the deployed model's REST API for sending data and receiving predictions during the model evaluation phase.

3- Data Import and preprocessing

3.1- data import

The movie_reviews dataset is a precompiled collection of movie reviews where each review has been manually tagged as either positive or negative. This dataset is part of the NLTK corpus, making it readily accessible for natural language processing tasks. The import process begins with loading the dataset using NLTK's corpus utilities, which allows for easy fetching of both the reviews and their associated labels. This is done through a structured query that loops over each file identifier corresponding to the reviews, categorizing them based on their sentiment classification. In the snippet shown in figure 2, document is a list of tuples where each tuple consists of a list of words (representing the review) and a label indicating the sentiment.

```
# Download NLTK data
nltk.download('movie_reviews')

# Load and prepare the dataset
documents = [(list(movie_reviews.words(fileid)), category)
             for category in movie_reviews.categories()
             for fileid in movie_reviews.fileids(category)]
reviews, labels = zip(*documents)
labels = np.array([1 if label == 'pos' else 0 for label in labels])
```

Figure 2 code snippet for loading the data

3.2- Data Preprocessing

The first step involves tokenizing the text, where each review is broken down into individual words or tokens. This is inherently handled during the data loading phase in NLTK but often needs further refinement such as removing stop words, punctuation, and performing stemming or lemmatization to reduce words to their base or root form.

For the model to process the textual data, the tokenized text must be converted into numerical format. This is achieved through vectorization. In this project, we use the Tokenizer class from Keras, which not only tokenizes the text but also converts it into sequences of integers where each integer represents a specific word in a dictionary of word indexes created by the tokenizer.

After vectorization, sequences of integers (representing the reviews) can vary in length. Most neural network models, however, expect input data of uniform size. Therefore, padding is applied to standardize the lengths of the sequences. This ensures each input sequence into the model has the same length.

The final preprocessing step involves dividing the data into training and testing sets. This split is crucial for training the model on a subset of the data and then testing its performance on unseen data to evaluate its generalization ability

```
tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(reviews)
sequences = tokenizer.texts_to_sequences(reviews)
data = pad_sequences(sequences, maxlen=200)
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
```

Figure 3- code snippet of Data Preprocessing

4- Model setup

The model constructed for this sentiment classification project utilizes a structured and flexible design tailored to process textual data effectively. The architecture is built using TensorFlow Keras, with a focus on optimizing performance through hyperparameter tuning facilitated by Keras Tuner.

The architecture begins with an Embedding layer, which is critical for text-processing applications. This layer converts the integer-encoded reviews into dense vectors of a fixed size (128 in this case), where each word is represented by a 128-dimensional vector. The embedding layer not only reduces the dimensionality of the input data but also captures some contextual relationships between words. The input length is capped at 200 words, meaning each review is truncated or padded to this length.

Following the embedding layer, the model includes several Dense layers, which are the building blocks of neural networks providing the learning capabilities. The number of these layers and their units are not fixed but are determined through hyperparameter tuning.

- **Number of Layers**: The hp.Int('n_layers', 1, 3) function from Keras Tuner allows the tuning process to explore between 1 to 3 dense layers, providing flexibility in how deep the model can be.
- **Units in Each Layer**: Each dense layer's size, i.e., the number of neurons, is also tuned from 64 to 256 through hp.Int('n_units', min_value=64, max_value=256, step=32). This range gives the model sufficient capacity to learn complex patterns in the data without immediately overfitting.
- **Dropout Layers** are interspersed between dense layers, where hp.Float('dropout_rate', 0.1, 0.5) determines the dropout rate. This technique helps prevent overfitting by randomly setting a fraction of input units to 0 at each update during training time, which helps to make the network more robust and less likely to rely on any single input feature.

A Global Average Pooling layer follows the dense layers. This layer simplifies the output by calculating the average output of each feature map in the previous layer, which helps reduce the model's complexity and computational cost.

The final layer is a Dense layer with a single neuron, using a sigmoid activation function. This setup is typical for binary classification tasks, as it outputs a probability indicating the likelihood of a review being positive.

The model is compiled with the Adam optimizer, a popular choice for deep learning applications due to its efficient handling of sparse gradients and adaptive learning rate capabilities. The loss function used is binary crossentropy, which is suitable for binary classification problems. Accuracy is monitored as the performance metric, providing a straightforward interpretation of the model's effectiveness in classifying the sentiments of reviews.

The model training involves fitting the model to the training data, where the number of epochs and batch size can be specified based on experimental design or further tuned. Validation during training is crucial for monitoring overfitting and tuning the model's generalization ability.

```python
import tensorflow as tf
from tensorflow.keras import layers, models

def build_model(hp):
    model = models.Sequential()
    model.add(layers.Embedding(10000, 128, input_length=200))
    for i in range(hp.Int('n_layers', 1, 3)):
        model.add(layers.Dense(units=hp.Int('n_units', min_value=64, max_value=256, step=32),
                               activation='relu'))
        model.add(layers.Dropout(hp.Float('dropout_rate', 0.1, 0.5)))
    model.add(layers.GlobalAveragePooling1D())
    model.add(layers.Dense(1, activation='sigmoid'))

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

Figure 4- Model Build function

5- Model tracking and hyperparameter tuning

Now that the model is defined, it's time to set up a machine learning workflow that seamlessly integrates the TensorFlow model development with advanced hyperparameter optimization via Keras Tuner, all while tracking the process using MLflow within a Databricks environment.

The process begins with configuring MLflow to track the machine learning experiments. By setting the tracking URI to "Databricks", MLflow directs all experiment logs, including parameters, metrics, and models, to the designated Databricks workspace. This integration ensures that all experiment data is centrally stored and easily accessible, which is crucial for auditability and reproducibility in machine learning projects.

The mlflow.tensorflow.autolog() function is invoked to automatically capture detailed information from TensorFlow training sessions. This feature simplifies the process of logging, as it automatically records training metrics for each epoch, model configurations, and final model parameters without requiring manual intervention. It's particularly beneficial for ensuring comprehensive documentation of the model's training process, which aids in debugging and model comparison.

Keras Tuner is utilized here to enhance the model's performance by finding the optimal hyperparameters. Here we have used the RandomSearch strategy, which randomly selects combinations of hyperparameters within specified ranges. Each trial involves training the model on the training dataset while validating on a subset (10% of the training data) to gauge the model's performance without biasing its training. This approach allows for an unbiased assessment of each hyperparameter configuration's effectiveness based on validation accuracy.

Within an MLflow run context, the tuner.search() method is executed, which orchestrates the model training and validation across different hyperparameter configurations. this systematically records the outcomes of each configuration, directly integrating these results with MLflow for later analysis.

```python
import mlflow
from kerastuner.tuners import RandomSearch

mlflow.set_tracking_uri("databricks")
mlflow.tensorflow.autolog()

tuner = RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=50,
    executions_per_trial=1,
    directory='my_dir',
    project_name='SentimentAnalysis',
    overwrite=True
)

with mlflow.start_run():
    tuner.search(X_train, y_train, epochs=5, validation_split=0.1)
    best_model = tuner.get_best_models(num_models=1)[0]

    test_loss, test_acc = best_model.evaluate(X_test, y_test)
    print("Test Accuracy: ", test_acc)
    mlflow.log_metric("test_accuracy", test_acc)
```

Figure 5 hyperparameter tuning and MLflow logging

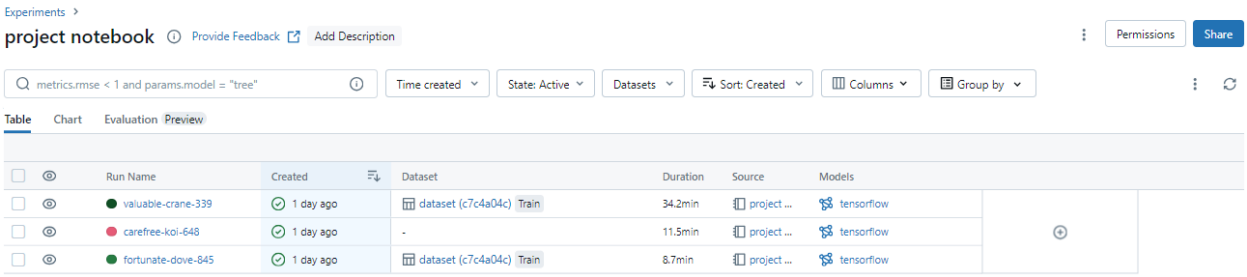

Figure 6 MLflow experiments interface

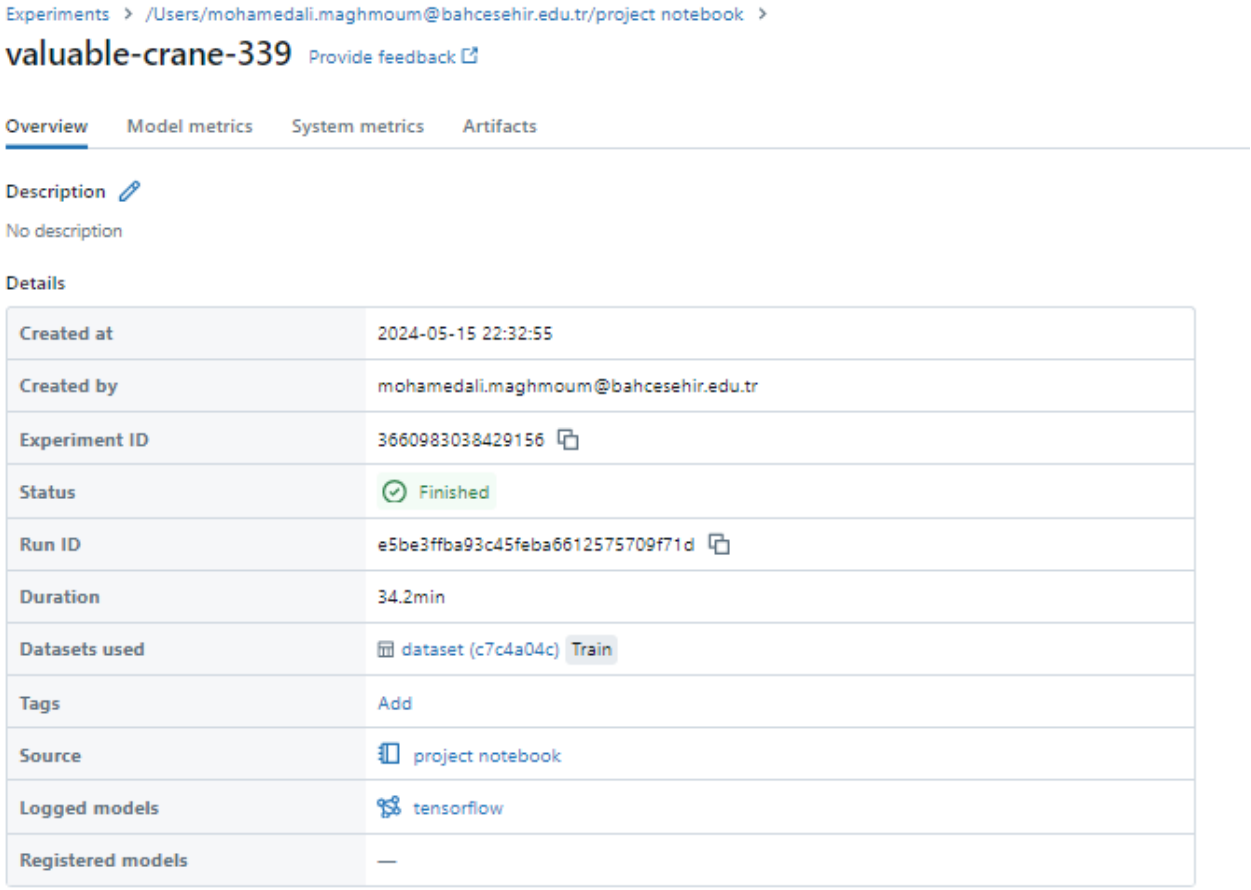

Figure 7 experiment details within MLflow

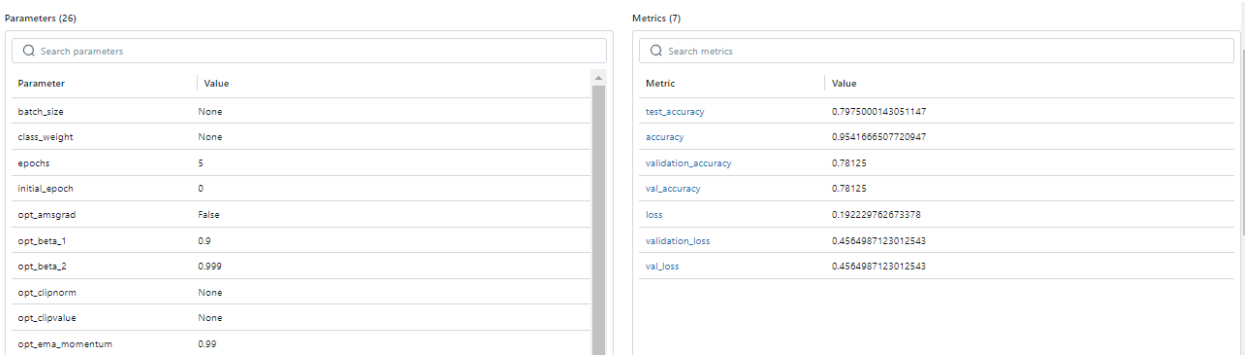

Figure 8 matrices and parameters of the experiments

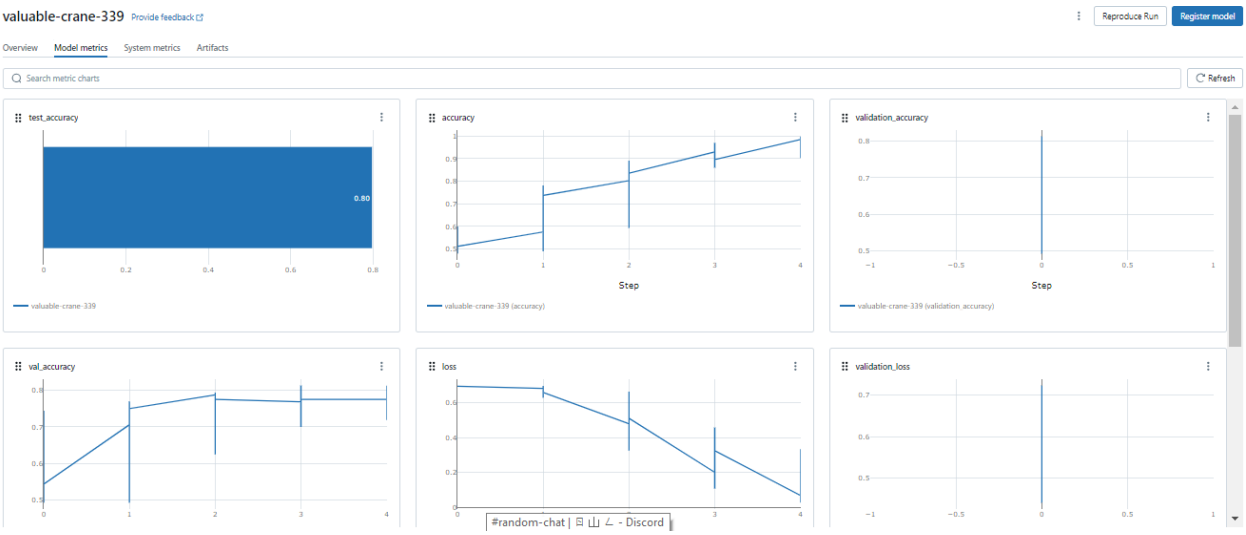

Figure 9 model matrices graphs

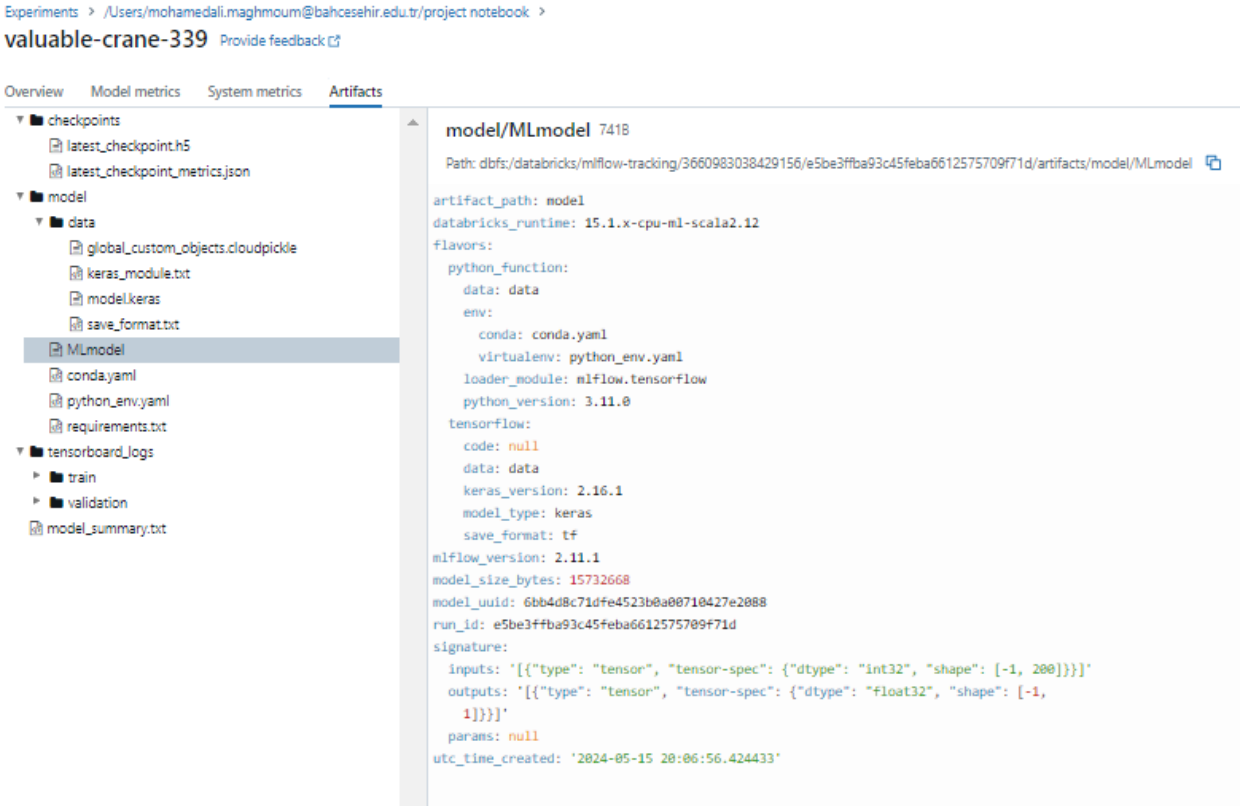

Figure 10 model artifact directory

6- Model Deployment and Monitoring

Deploying and monitoring the model efficiently are crucial steps to ensure the model remains useful in real-world applications. Using MLflow's model serving capabilities, the best-performing model, identified through rigorous training and hyperparameter tuning, is packaged and deployed. MLflow

allows the model to be served as a REST API, which can be easily accessed for making predictions from various frontend interfaces or other services. This deployment strategy is particularly advantageous as it simplifies the integration of the machine learning model into production environments, allowing for seamless updates and scalability.

To ensure the deployed model continues to perform well under changing real-world conditions, a robust monitoring system is set up. This system tracks performance metrics such as accuracy, precision, recall, and others over time. Additionally, performance can be monitored through the logging of response times and the frequency of prediction requests, which helps in identifying operational bottlenecks. MLflow facilitates this by logging these metrics during the model's operation, allowing for the continuous assessment of the model's effectiveness. Depending on the application, real or simulated incoming data are used to test the model periodically. This could involve scheduled re-evaluations with new data batches to simulate changing conditions, ensuring the model adapts and maintains high performance standards.

```python
import mlflow.tensorflow

model_path = "models/sentiment_analysis"
# Correcting the logging call:
mlflow.tensorflow.log_model(best_model, artifact_path=model_path)

# Constructing the URI to reference the logged model
run_id = mlflow.active_run().info.run_id
model_uri = f"runs:/{run_id}/{model_path}"

# Registering the model in the MLflow Model Registry
model_details = mlflow.register_model(model_uri=model_uri, name="SentimentAnalysisModel")

# Transitioning the registered model to the production stage
client = mlflow.tracking.MlflowClient()
client.transition_model_version_stage(
    name="SentimentAnalysisModel",
    version=model_details.version,
    stage="Production"
)


```

Figure 11 registering the model to production

7- MLflow Model Registry

The MLflow Model Registry is an integral part of managing the lifecycle of machine learning models. It provides a centralized hub to store, version, and manage models systematically. After deploying the model, it is registered in the MLflow Model Registry, which allows for easy tracking of different model versions and their performance metrics. This registry supports several model stages such as Staging,

Production, and Archived, facilitating a controlled and transparent transition of models from development to production.

The model registry is not available in the community edition of databricks, so we had to run the code in a different environment in order to activate it.

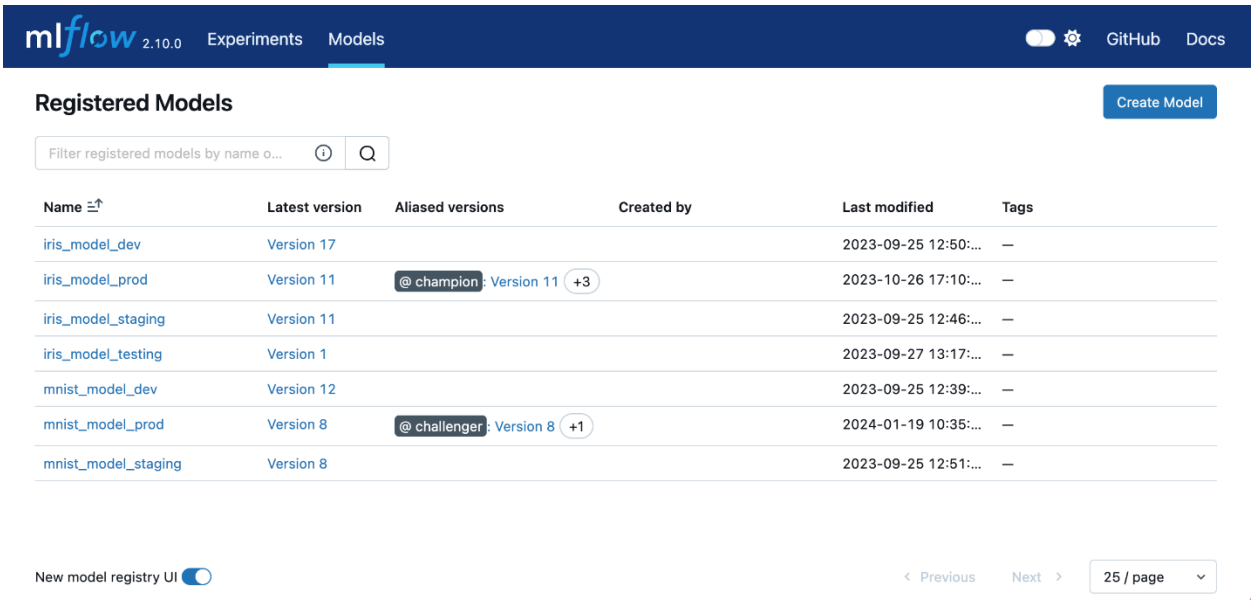

Figure 12 Interface of the model registry

The process typically involves initially registering the model as it enters the staging phase, where further integration tests or user acceptance tests might be performed. Based on the results and after thorough validation, the model can then be transitioned to the Production stage.

This stage transition is crucial for operational models as it ensures only fully vetted models are in production. Furthermore, the MLflow Model Registry allows for annotating models with descriptions, adding model version tags, and managing model permissions, which enhances collaboration among teams.Using the MLflow Model Registry thus not only simplifies the model management process but also enhances the governance and auditability of machine learning models.

This capability is particularly important in regulated industries where tracking the lineage and versioning of models is mandatory. By leveraging the Model Registry, organizations can ensure that their model deployment workflows are robust, repeatable, and transparent, thus supporting high standards of model governance and operational excellence.

8- Conclusion and insights

In conclusion, this project successfully demonstrated the development, tuning, and deployment of a sentiment analysis model using TensorFlow and Keras, integrated with MLflow for comprehensive lifecycle management. The use of the NLTK's `movie_reviews` dataset provided a solid foundation for training and evaluating the model, ensuring that it was robust enough to handle real-world textual data

effectively. Through meticulous experiment tracking, hyperparameter tuning via Keras Tuner, and seamless integration with MLflow.

My insights with this project is that it's a useful way to get a hands-on experience of what MLflow can do, if I can integrate the use of MLflow with many of my other projects, it can be a very valuable skill to have, MLflow is easy to work with, but its advantages and offerings are very extensive, so this will be one of my most useful projects that I have made.

Familiarizing myself with Databricks community edition was also a bonus, the environment has a high integration with MSflow and it made this project a lot easier to do, however, it is unfortunate that the environment doesn't provide integration for model registry, this required some last minute changes to run the code in a different environment in order to work with it.

The Idea of turning the model into an API is a very useful and helpful feature that will be functional in attempting to integrate the model with other systems, it helps us manage the models that we are in production or in development using a very easy to use UI.

9- Code repository

https://github.com/Magmuma/AIN-3009-Term-Project-MLflow-applied-to-sentiment-classification-AI-model.