Custom model tracking with MLflow and remote mySQL docker database

Setting up MySQL database:

run docker desktop and open wsl linux on windows

powershell > wsl

- `docker pull mysql/mysql-server:latest`
- `docker run --name=mysql1 -p 3306:3306 -p 33060:33060 -d mysql/mysql-server:latest`
- We map the ports 3306 and 33060 to the docker container so that we can access the database outside of the docker container.
- **docker ps**

  , let's configure the password for the *root* user. To do that we will need the automatically generated password for *root*.

  > `docker logs mysql1 2>&1 | grep GENERATED`

  > `docker exec -it mysql1 mysql -u root -p`

  Mysql will ask for the password, you must enter the password generated by mysql which we saw earlier.

  After this, we can change the password for the *root* user.

  Replace the string **'password'** with the actual password that you want to set. To make the access available for the *root* user

from outside the container, we will update the host value from *'localhost'* to *'%'*.

```
> alter user 'root'@'localhost' identified by 'password';
```

```
>update mysql.user set host = '%' where user='root';
```

```
>SELECT host, user FROM mysql.user;
```

```
> create database mlflow;
```

```
add new user account
```

```
>CREATE USER 'senol'@'localhost' IDENTIFIED BY 'password';
> GRANT ALL PRIVILEGES ON *.* TO 'senol'@'localhost'
   ->    WITH GRANT OPTION;
> CREATE USER 'senol'@'%' IDENTIFIED BY 'password';
> GRANT ALL PRIVILEGES ON *.* TO 'senol'@'%'
   ->    WITH GRANT OPTION;
```

```
>quit;
```

Now open conda shell on windows and run mlflow ui with parameters below:

- **mlflow ui --backend-store-uri "mysql://senol:password@localhost:3306/mlflow"**

then run below code in pycharm and refresh mlflow ui from web browser to see the new runs.

```
from sys import version_info
import cloudpickle
import pandas as pd
```

```
import mlflow.pyfunc
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

#import os
#print("ENV:",os.getenv('MLFLOW_TRACKING_URI'))

#
# Good and readable paper from the authors of this package
# http://comp.social.gatech.edu/papers/icwsm14.vader.hutto.pdf
#
#remote_server_uri = "http://localhost:5000"  # set to your server URI.
Local web server. Artifacts are in local mlruns.
#mlflow.set_tracking_uri(remote_server_uri)

user = 'senol'
password = 'password'
hostname = 'localhost'
port = 3306
database = 'mlflow'
uri = f'mysql://{user}:{password}@{hostname}:{port}/{database}'
print(uri)
mlflow.set_tracking_uri(uri)

mlflow.set_experiment("/SERVE_CUSTOM_NODEL_vader")

INPUT_TEXTS = [{'text': "This is a bad movie. You don't want to see it! :-
)"},
           {'text': "Ricky Gervais is smart, witty, and creative!!!!!!
:D"},
           {'text': "LOL, this guy fell off a chair while sleeping and
snoring in a meeting"},
           {'text': "Men shoots himself while trying to steal a dog, OMG"},
           {'text': "Yay!! Another good phone interview. I nailed it!!"},
           {'text': "This is INSANE! I can't believe it. How could you do
such a horrible thing?"}]

PYTHON_VERSION = "{major}.{minor}.{micro}".format(major=version_info.major,
                                                  minor=version_info.minor,
                                                  micro=version_info.micro)
def score_model(loaded_model):
    # Use inference to predict output from the customized PyFunc model
    for i, text in enumerate(INPUT_TEXTS):
        text = INPUT_TEXTS[i]['text']
        m_input = pd.DataFrame([text])
        scores = loaded_model.predict(m_input)
        print(f"<{text}> -- {str(scores[0])}")

# Define a class and extend from PythonModel
class SocialMediaAnalyserModel(mlflow.pyfunc.PythonModel):

    def __init__(self):
      super().__init__()
      # embed your vader model instance
      self._analyser = SentimentIntensityAnalyzer()

   # preprocess the input with prediction from the vader sentiment model
    def _score(self, txt):
      prediction_scores = self._analyser.polarity_scores(txt)
      return prediction_scores

    def predict(self, context, model_input):
```

```
        # Apply the preprocess function from the vader model to score
        model_output = model_input.apply(lambda col: self._score(col))
        return model_output


model_path = "vader_model"
reg_model_name = "NewPyFuncVaderSentimentAnalysis"
vader_model = SocialMediaAnalyserModel()



# Save the conda environment for this model.
conda_env = {
    'channels': ['defaults', 'conda-forge'],
    'dependencies': [
        'python={}'.format(PYTHON_VERSION),
        'pip'],
    'pip': [
        'mlflow',
        'cloudpickle=={}'.format(cloudpickle.__version__),
        'vaderSentiment==3.3.2'
    ],
    'name': 'mlflow-env'
}

# Save the model
with mlflow.start_run(run_name="Vader_Sentiment_Analysis") as run:
    model_path = f"{model_path}-{run.info.run_uuid}"
    mlflow.log_param("algorithm", "VADER")
    mlflow.log_param("total_sentiments", len(INPUT_TEXTS))
    #mlflow.pyfunc.save_model(path=model_path, python_model=vader_model,
conda_env=None)

    # Use the saved model path to log and register into the model registry
    mlflow.pyfunc.log_model(artifact_path=model_path,
                            python_model=vader_model,
                            registered_model_name=reg_model_name,
                            conda_env=conda_env)

# Load the model from the model registry and score
model_uri = f"models:/{reg_model_name}/1"
print("model uri",model_uri)
loaded_model = mlflow.pyfunc.load_model(model_uri)
score_model(loaded_model)
```