1. MLOps Diagram

A screenshot of a computer

Description automatically generated with medium confidence

1. Steps for ML model deployment
   1. Load training data (Train\_keystroke.csv) file into pandas dataframe using *read\_csv()* method.
   2. Create an empty dataframe with columns to contain mean and standard deviation values of two consecutive key strokes times (press/release-HT/PPT/RRT/RPT), in each row with class output as user ID. This can be done with nested for-loops logic.
   3. Initialize an instance of target classifier to be trained. For SVM, this is done using *sklearn.svm.SVC()*, for Random Forest (RF) classifier, *sklearn.ensemble.RandomForestClassifier()* can be used, and for XGboost (XGB) classifier, *xgboost.XGBClassifier()* is being used.
   4. Train each model using *<model>.fit()* method where ‘X’ input data and ‘y’ class output data should be passed into. ‘X’ and ‘y’ can be extracted from the last created dataframe having mean/stdev input values.
   5. Then *<model>.predict()* can be used to test the model.
   6. Afterwards, save the model in *.pkl* format file on local disk using *joblib.load()* method.
   7. Next Step is, create Azure account and create there a ML workspace on web interface. In this workspace upload all three models one by one, by uploading the *.pkl* file.
   8. Once the models are uploaded, then create REST endpoint for each model uploaded. For each model, select the option to deploy as a web service.
   9. While creating endpoint, name the endpoint and make it containerized instance. This requires uploading score python script file and conda/python dependencies file in .yml (YAML) format, as provided separately.
   10. For SVM endpoint creation, “*score.py*” and “*dependencies.yml*” have been used, for RF endpoint, *“score\_2.py*” and “*dependencies\_2.yml*” have been used, and for XGB endpoint, “*score\_3.py*” and *“dependencies\_3.yml*” have been used.
   11. Make sure to mention the correct and compatible version numbers of the packages required to be used.
   12. The deployment start immediately afterwards however, it takes some time. If deployment is successful it will show the status as ‘healthy’ and the REST endpoint url in the details section. Otherwise, status will be ‘unhealthy’ or ‘failed’ and no REST endpoint url will be shown. This can be due to conflicts among the modules incompatibilities during installation in the container.
   13. Once the endpoint url is shown then use it in REST API script to predict the class output on an input data. A prediction script is also provided in a separate python file which takes input as argument on commandline in JSON format then pass it onto webservice model to get a predicted user ID as class output. For example, >> *python3 predict\_user\_2.py ‘{"Model":"XGB","HT": {"Mean": 48.43, "STD": 23.34}, "PPT": {"Mean": 120.43, "STD": 37.41}, "RRT": {"Mean": 124.43, "STD": 45.34}, "RPT": {"Mean": 132.56, "STD": 47.12}}’*
   14. In this output from my end is “ predicted user = 57”.