**Managing ML Development Cycle using MLFlow**

For managing the entire Machine Learning development cycle, we used MLFlow. It is an open-source platform streamline the process of experimentation, tracking, deploying and collaboration.

In our ML model development with logistic regression, we experimented and tracked by logging the parameters for “regularization\_param”. We reproduced the experiments and compared our model with different values of “regularization\_param” like [100, 10, 1, 0.1, 0.01, 0.001, 0.0001]. The performance metrics we analysed are accuracy, precision, and f1-score.

The different experiments we did can be found in “runs.csv” located at “BankCustomerChurn/”

**C in the x-axis – regularisation value**

Chart, scatter chart

Description automatically generated

Figure 1: reg vs accuracy

Chart, scatter chart

Description automatically generated

Figure 2: reg vs precision

Chart, scatter chart

Description automatically generated

Figure 3: reg vs f1\_score

From all the above metrics, we analysed and concluded that the model works best when the “regularisation\_param” is **1.0**

**Creating MLFlow projects**

The ML model is packaged along with its dependencies, so that it can be reused, easily shared, and deployed. Also, the packaged model can be installed and run independently of the original training environment and perform consistently over time.

There are 2 files for we need to add for doing this process

* The “MLproject” file: It tells us where to look for the dependencies.
* The “conda.yaml” file: It specifies all the dependencies we need to run our project.

The files are located at the “src/models/<file>” path

Graphical user interface, text, application, email

Description automatically generated

Figure 4: MLproject file

Graphical user interface, text, application

Description automatically generated

Figure 5: conda.yaml

To run this project, go to “BankCustomerChurn/src” and type the following command with different values of reg.

**mlflow run models -P reg=0.1**

Graphical user interface, text, application

Description automatically generated

Figure 6: mlflow run

To run this project directly from GitHub, type the following command

**mlflow run** [**git@github.com:martinsejas/BankCustomerChurn.git**](mailto:git@github.com:martinsejas/BankCustomerChurn.git) **-P reg=0.1**

**MLFlow models**

We used the log\_model( ) to save the model. They can be found in the “mlruns/0/” folder, where each subdirectory represents different runs.

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**MODEL PREDICTION WITH THE TEST DATA**

**Test splitting the dataset**

Using the train\_test\_split from sklearn.model\_selection, we splitted the test dataset with 10% of the whole data.

The test dataset can be found at “project-outputs/test\_dataset.csv”

**Saving the Encoder, Scaler and polynomial model using Joblib**

With the help of joblib, we saved the fitted encoder, standard Scaler and polynomial model, which can be later used for prediction.

**joblib.dump(model\_name, “save\_model”)**

**ml**

**Saving the best MLFlow model**

The best MLFlow model we have is with regularisation parameter 1.0. By using the following code in the train\_model.py, we can save the model and use it for prediction.

**mlflow.sklearn.save\_model(poly\_reg\_model, “model\_logistic”)**

It creates a folder with dependencies files both for pyenv and conda, MLmodel, and model.pkl file.

The saved model can be found at “src/models/model\_logistic”

**Model prediction with MLFlow**

The test data is imported, and all the saved models are loaded using Joblib to preprocess the data before we do the prediction.

**joblib.load(‘model\_name’)**

The saved mlflow model is loaded using mlflow.sklearn

**mlflow.sklearn.load\_model(‘model\_logistic’)**

The loaded model is used to predict the preprocessed data and it is stored in “predict\_output.csv” which is in “project\_outputs” directory.