Training machine learning models on Azure Databricks

Project on cloud computing

Izabela Gesla

January 2, 2022

This project was made on the Azure's data analytic platform - Databricks in the Machine Learning environment. Project was based on Microsoft End-to-End Example presenting how to build a model to predict the quality of Portugese "Vinho Verde" wine based on the wine's physicochemical properties. [1] For this purpose, datasets from the UCI Machine Learning Repository [2] was used.

Contents

[1] `https://docs.microsoft.com/en-us/azure/databricks/_static/notebooks/mlflow/mlflow-end-to-end-example.html`

[2] presented in modeling wine preferences by data mining from physicochemical properties [Cortez et al., 2009].

1 Set up

To build a model in Databricks environment, we need to launch the Azure platform and then create resource called Databricks. Then, we change the environment from "Data Science and Engineering" to "Machine Learning".

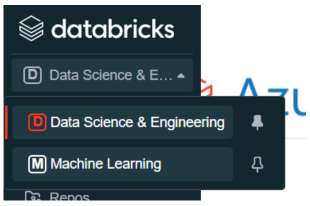
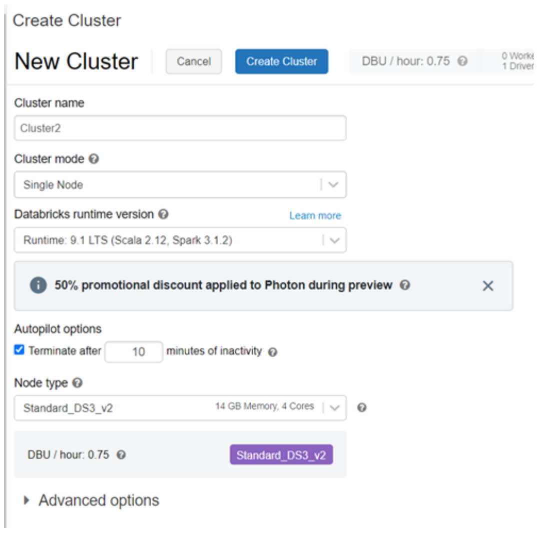


Figure 1: Changing

Then, we need to create a cluster on which we will run our notebook. To do so, we create Single Node Cluster.


Create Cluster

New Cluster Cancel Create Cluster DBU / hour: 0.75 0 Worke 1 Driver

Cluster name
Cluster2

Cluster mode
Single Node

Databricks runtime version Learn more
Runtime: 9.1 LTS (Scala 2.12, Spark 3.1.2)

50% promotional discount applied to Photon during preview

Autopilot options
Terminate after 10 minutes of inactivity

Node type
Standard_DS3_v2 14 GB Memory, 4 Cores

DBU / hour: 0.75 Standard_DS3_v2

Advanced options

Figure 2: Creating cluster

1.1 Installing packages

First of all, we need to install the necessary packages in the Databricks created workspace. For this particular project we will use mlflow, hyperopt, xgboost :

```
pip install mlflow
```

```
pip install hyperopt
```

```
pip install xgboost
```

2 Model implementing

2.1 Preparing data

Firstly, we need to export the datasets and load them to our workspace. To do so, go to link [3] and download files named "winequality-red.csv" and "winequality-white.csv". For uploading then into workspace we will use pandas function read from csv as following with local path in "[]" :

```
import pandas as pd

white_wine = pd.read_csv(r"[]/winequality-white.csv", sep=';')
red_wine = pd.read_csv(r"[]/winequality-red.csv", sep=';')
```

Then, we need to create a binary feature that decides whether the wine is red or white.

```
red_wine['is_red'] = 1
white_wine['is_red'] = 0
```

Then, merge the two DataFrames into a single dataset:

```
data = pd.concat([red_wine, white_wine], axis=0)
data.rename(columns=lambda x: x.replace(' ', '_'), inplace=True)
data.head()
```

And to plot for example the distribution of the quality of wine seaborn distplot function:

```
import seaborn as sns
sns.distplot(data.quality, kde=False)
```

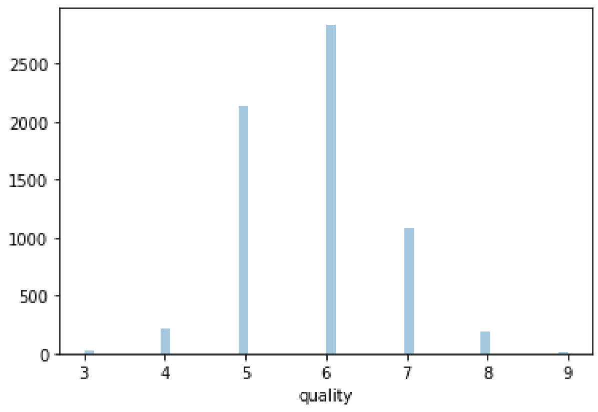


Figure 3: Plot

Then, we decide that a high quality wine is if it has a score greater than 7.

```
high_quality = (data.quality >= 7).astype(int)
data.quality = high_quality
```

We can also visualise correlation between features in data with matplotlib:

```
import matplotlib.pyplot as plt

dims = (3, 4)
```

[3]https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/

```python
f, axes = plt.subplots(dims[0], dims[1], figsize=(25, 15))
axis_i, axis_j = 0, 0
for col in data.columns:
  if col == 'is_red' or col == 'quality':
    continue # Box plots cannot be used on indicator variables
  sns.boxplot(x=high_quality, y=data[col], ax=axes[axis_i, axis_j])
  axis_j += 1
  if axis_j == dims[1]:
    axis_i += 1
    axis_j = 0
```

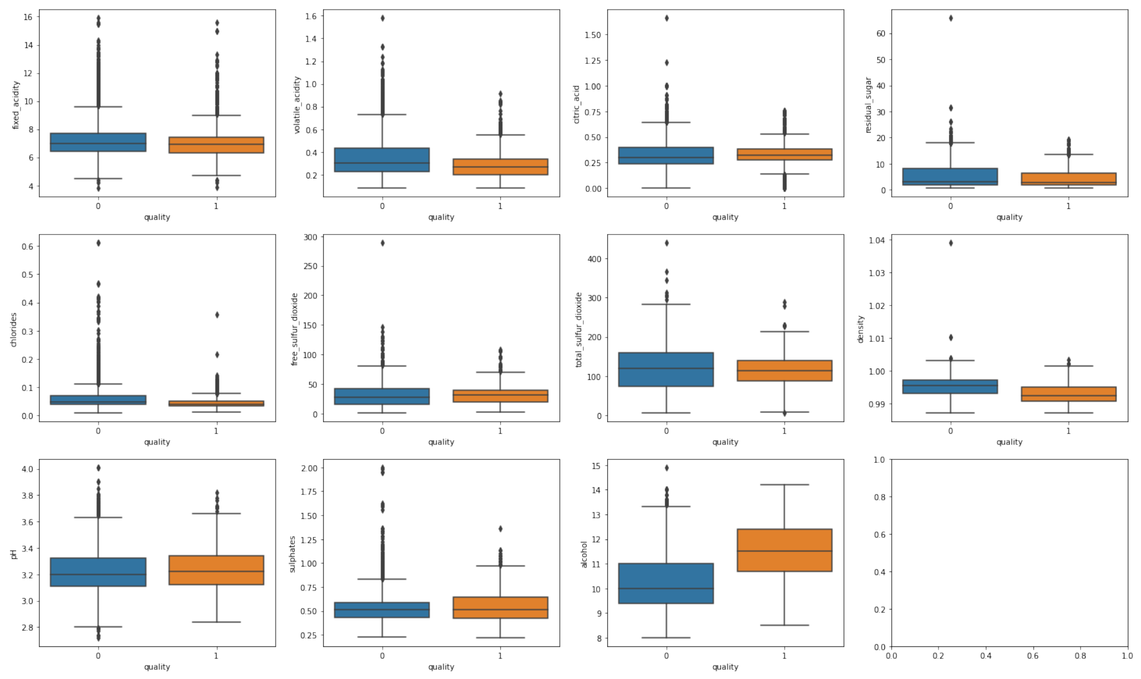


Figure 4: Bar plots

In the above plots, we can see, that few variables stand out as good predictors of quality, for example the alcohol box plot. High alcohol content is correlated with quality.

At the last point of preparing data to further steps, we check if there are missing values:

```python
data.isna().any()
```

If there is non, we can start our training part.

2.2 Training data

To start, we split the data into training and validation sets. For training data, we use the scikit-learn tool, which is made for predictive data analysis and machine learning in Python. For splitting, we import train_test_split to split data into random train and test subsets.

```python
from sklearn.model_selection import train_test_split

train, test = train_test_split(data, random_state=123)
X_train = train.drop(["quality"], axis=1)
X_test = test.drop(["quality"], axis=1)
y_train = train.quality
```

```
y_test = test.quality
```

Now, we can build a baseline model. We will use random forest classifier (random decision trees), because the output is binary and there may be interactions between multiple variables. This method is a learning model for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For keeping track of model accuracy we will use MLflow, also to save the model for later use. The predict method of sklearn's RandomForestClassifier returns a binary classification (0 or 1). The following code creates a wrapper function, SklearnModelWrapper, that uses the predict_proba method to return the probability that the observation belongs to each class.

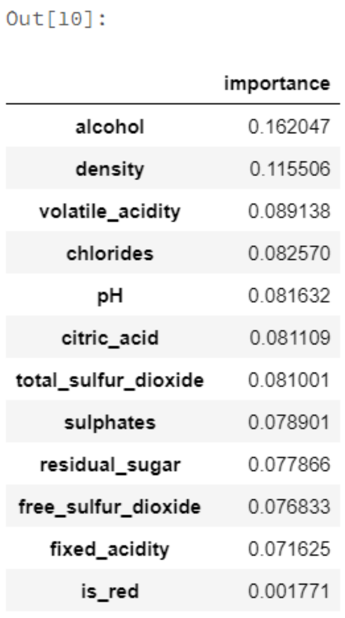
Formula mlflow.start_run creates a new MLflow run to track the performance of this model. Within the context, you call mlflow.log_param to keep track of the parameters used, and mlflow.log_metric to record metrics like accuracy.]

Function predict_proba returns [prob_negative, prob_positive], so slice the output with [:, 1]

```
import mlflow
import mlflow.pyfunc
import mlflow.sklearn
import numpy as np
import sklearn
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from mlflow.models.signature import infer_signature
from mlflow.utils.environment import _mlflow_conda_env
import cloudpickle
import time


class SklearnModelWrapper(mlflow.pyfunc.PythonModel):
  def __init__(self, model):
    self.model = model

  def predict(self, context, model_input):
    return self.model.predict_proba(model_input)[:,1]

with mlflow.start_run(run_name='untuned_random_forest'):
  n_estimators = 10
  model = RandomForestClassifier(n_estimators=n_estimators, random_state=np.random.
    RandomState(123))
  model.fit(X_train, y_train)

  predictions_test = model.predict_proba(X_test)[:,1]
  auc_score = roc_auc_score(y_test, predictions_test)
  mlflow.log_param('n_estimators', n_estimators)

  mlflow.log_metric('auc', auc_score)
  wrappedModel = SklearnModelWrapper(model)

  signature = infer_signature(X_train, wrappedModel.predict(None, X_train))

        additional_conda_deps=None,
        additional_pip_deps=["cloudpickle=={}".format(cloudpickle.__version__), "scikit-
    learn=={}".format(sklearn.__version__)],
        additional_conda_channels=None,
    )
  mlflow.pyfunc.log_model("random_forest_model", python_model=wrappedModel, conda_env=
    conda_env, signature=signature)
```

Then, we will examine the learned feature importances output by the model as a sanity-check. A sanity check is a basic test to quickly evaluate whether a claim or the result of a calculation can possibly be true. We can compare the results with the first bad visualization in the preparing data section.

```
feature_importances = pd.DataFrame(model.feature_importances_, index=X_train.columns.
    tolist(), columns=['importance'])
feature_importances.sort_values('importance', ascending=False)
```



Out[10]:

	importance
alcohol	0.162047
density	0.115506
volatile_acidity	0.089138
chlorides	0.082570
pH	0.081632
citric_acid	0.081109
total_sulfur_dioxide	0.081001
sulphates	0.078901
residual_sugar	0.077866
free_sulfur_dioxide	0.076833
fixed_acidity	0.071625
is_red	0.001771

Figure 5: Properties

As we got the alcohol as the most important property, it is consistent with our previous assumptions visualized on bar plot.

2.3 Model registration

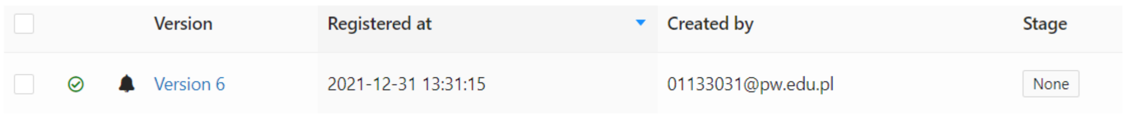
We will now register our model in MLflow Model Registry, to easily reference ti it from anywhere within Databricks. It is possible to do it both ways, by Azure platform [4] by clicking or programmatically as following:

```
run_id = mlflow.search_runs(filter_string='tags.mlflow.runName="untuned_random_forest"').
    iloc[0].run_id

model_name = "wine_quality"
model_version = mlflow.register_model(f"runs:/{run_id}/random_forest_model", model_name)

time.sleep(15)
```

[4]//docs.microsoft.com/en-us/azure/databricks/applications/machine-learning/manage-model-lifecycle/#create-or-register-a-model-using-the-ui

Because registering the model takes a few seconds, we added a small delay at the end.

Now, we can go to the Models page. To do so, click the the Models icon in the left sidebar, we can see model is registered, but with no stage.



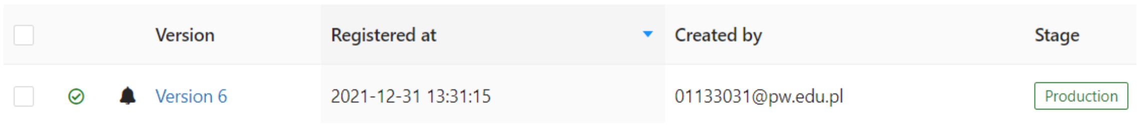
	Version	Registered at	Created by	Stage
	Version 6	2021-12-31 13:31:15	01133031@pw.edu.pl	None

Figure 6: Model registration

Next, transition this model to production and load it into this notebook from Model Registry.

```
from mlflow.tracking import MlflowClient

client = MlflowClient()
client.transition_model_version_stage(
    name=model_name,
    version=model_version.version,
    stage="Production",
    )
```

The Models page now shows the model version in stage ”Production”.



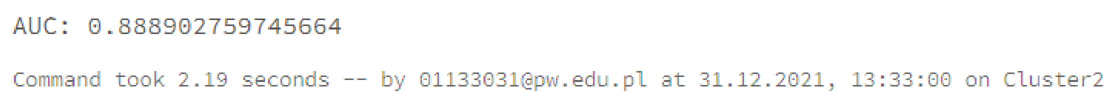
	Version	Registered at	Created by	Stage
	Version 6	2021-12-31 13:31:15	01133031@pw.edu.pl	Production

Figure 7: Model registration to status production

To show the parameter AUC (Area under the curve) you can type the following:

```
model = mlflow.pyfunc.load_model(f"models:/{model_name}/production")
print(f'AUC: {roc_auc_score(y_test, model.predict(X_test))}')
```

AUC is a measure of the classifier’s ability to discriminate between classes. The higher the AUC, the better the performance of the model.



```
AUC: 0.888902759745664

Command took 2.19 seconds -- by 01133031@pw.edu.pl at 31.12.2021, 13:33:00 on Cluster2
```

Figure 8: AUC

This result AUC = 0.88 shows that the model is quite good.

2.4 Experiment with parallel hyperparameter tunning

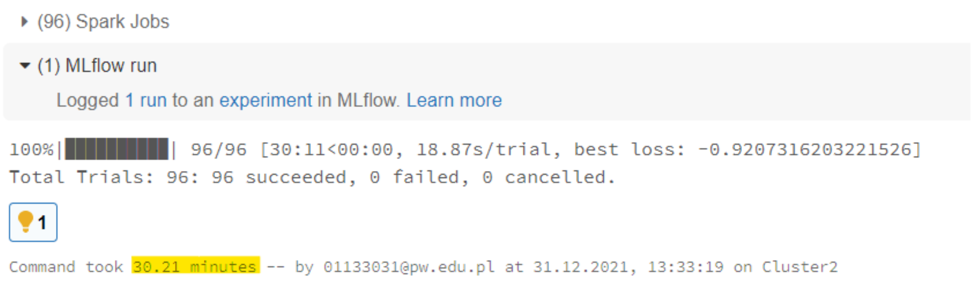
In this section we will try the hyperparameter tuning.

In machine learning, hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. Hyperparameter is a parameter whose value is used to control the learning process, when, the values of other parameters (typically node weights) are learned.

We will use the xgboost library to train a model to be more accurate. It performs a parallel hyper-parameter sweep to train multiple models in parallel. Using Hyperopt and SparkTrials. As we've done it before, we will use MLflow to track the performance of each parameter configuration.

```python
from hyperopt import fmin, tpe, hp, SparkTrials, Trials, STATUS_OK
from hyperopt.pyll import scope
from math import exp
import mlflow.xgboost
import numpy as np
import xgboost as xgb

search_space = {
  'max_depth': scope.int(hp.quniform('max_depth', 4, 100, 1)),
  'learning_rate': hp.loguniform('learning_rate', -3, 0),
  'reg_alpha': hp.loguniform('reg_alpha', -5, -1),
  'reg_lambda': hp.loguniform('reg_lambda', -6, -1),
  'min_child_weight': hp.loguniform('min_child_weight', -1, 3),
  'objective': 'binary:logistic',
  'seed': 123,
}

def train_model(params):
  mlflow.xgboost.autolog()
  with mlflow.start_run(nested=True):
    train = xgb.DMatrix(data=X_train, label=y_train)
    test = xgb.DMatrix(data=X_test, label=y_test)
    booster = xgb.train(params=params, dtrain=train, num_boost_round=1000,\
                        evals=[(test, "test")], early_stopping_rounds=50)
    predictions_test = booster.predict(test)
    auc_score = roc_auc_score(y_test, predictions_test)
    mlflow.log_metric('auc', auc_score)

    signature = infer_signature(X_train, booster.predict(train))
    mlflow.xgboost.log_model(booster, "model", signature=signature)

    return {'status': STATUS_OK, 'loss': -1*auc_score, 'booster': booster.attributes()}

spark_trials = SparkTrials(parallelism=10)

with mlflow.start_run(run_name='xgboost_models'):
    best_params = fmin(
    fn=train_model,
    space=search_space,
    algo=tpe.suggest,
    max_evals=96,
    trials=spark_trials,
    )
```

This code will take some time to finish, as you can see in the picture below it took about 30 minutes on our cluster.

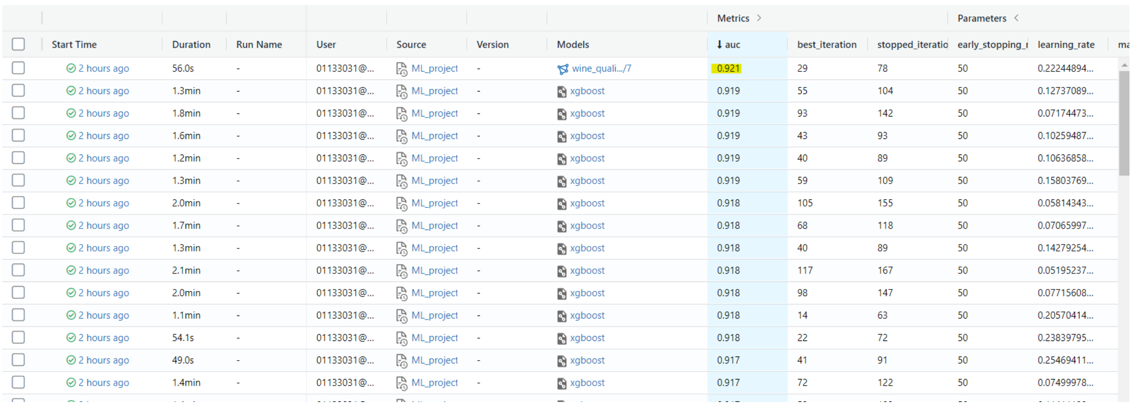


```
▸ (96) Spark Jobs
▾ (1) MLflow run
    Logged 1 run to an experiment in MLflow. Learn more
100%|██████████| 96/96 [30:11<00:00, 18.87s/trial, best loss: -0.9207316203221526]
Total Trials: 96: 96 succeeded, 0 failed, 0 cancelled.
1
Command took 30.21 minutes -- by 01133031@pw.edu.pl at 31.12.2021, 13:33:19 on Cluster2
```

Figure 9: Debugging a code

3 Results

To visualize the results we need to open the Experiment Runs sidebar. We can see the MLflow runs. Click on Date next to the down arrow to display a menu, and select 'auc' to display the runs sorted by the auc metric.

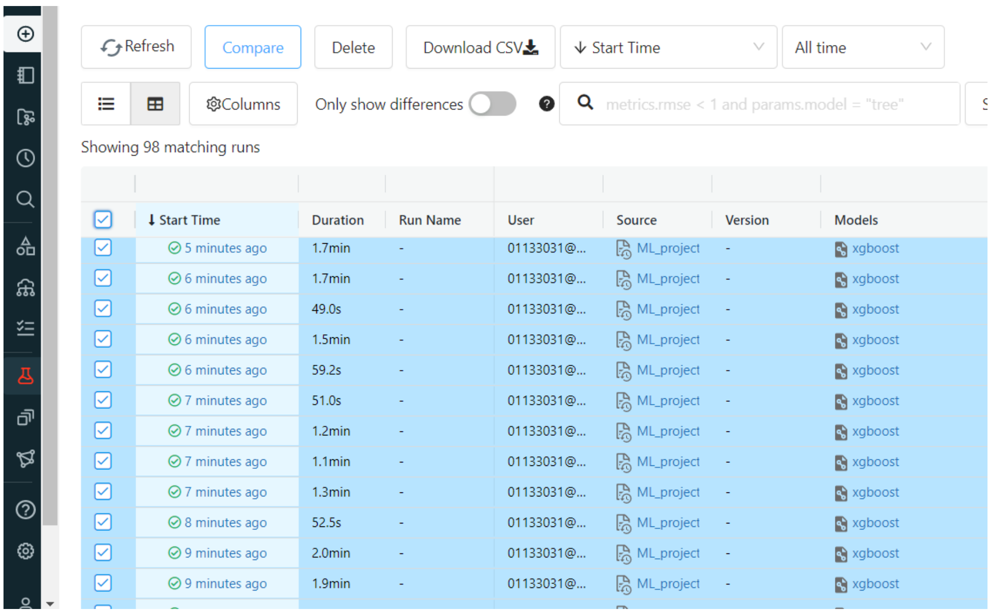


								Metrics >			Parameters <		
☐	Start Time	Duration	Run Name	User	Source	Version	Models	↓ auc	best_iteration	stopped_iteratio	early_stopping_r	learning_rate	ma
☐	2 hours ago	56.0s	-	01133031@...	ML_project	-	wine_quali.../7	0.921	29	78	50	0.22244894...	
☐	2 hours ago	1.3min	-	01133031@...	ML_project	-	xgboost	0.919	55	104	50	0.12737089...	
☐	2 hours ago	1.8min	-	01133031@...	ML_project	-	xgboost	0.919	93	142	50	0.07174473...	
☐	2 hours ago	1.6min	-	01133031@...	ML_project	-	xgboost	0.919	43	93	50	0.10259487...	
☐	2 hours ago	1.2min	-	01133031@...	ML_project	-	xgboost	0.919	40	89	50	0.10636858...	
☐	2 hours ago	1.3min	-	01133031@...	ML_project	-	xgboost	0.919	59	109	50	0.15803769...	
☐	2 hours ago	2.0min	-	01133031@...	ML_project	-	xgboost	0.918	105	155	50	0.05814343...	
☐	2 hours ago	1.7min	-	01133031@...	ML_project	-	xgboost	0.918	68	118	50	0.07065997...	
☐	2 hours ago	1.3min	-	01133031@...	ML_project	-	xgboost	0.918	40	89	50	0.14279254...	
☐	2 hours ago	2.1min	-	01133031@...	ML_project	-	xgboost	0.918	117	167	50	0.05195237...	
☐	2 hours ago	2.0min	-	01133031@...	ML_project	-	xgboost	0.918	98	147	50	0.07715608...	
☐	2 hours ago	1.1min	-	01133031@...	ML_project	-	xgboost	0.918	14	63	50	0.20570414...	
☐	2 hours ago	54.1s	-	01133031@...	ML_project	-	xgboost	0.918	22	72	50	0.23839795...	
☐	2 hours ago	49.0s	-	01133031@...	ML_project	-	xgboost	0.917	41	91	50	0.25469411...	
☐	2 hours ago	1.4min	-	01133031@...	ML_project	-	xgboost	0.917	72	122	50	0.07499978...	

Figure 10: Sorted by AUC

You can see, that the highest auc value is 0.92.

MLflow is used to track parameters and performance metrics of each run. To see the results, we can go to the "Experiment" section and to navigate to the MLflow Runs Table, you need to click the External Link icon at the top of the Experiment Runs sidebar and then select all the runs from experiment to compare.



Showing 98 matching runs

Start Time	Duration	Run Name	User	Source	Version	Models
5 minutes ago	1.7min	-	01133031@...	ML_project	-	xgboost
6 minutes ago	1.7min	-	01133031@...	ML_project	-	xgboost
6 minutes ago	49.0s	-	01133031@...	ML_project	-	xgboost
6 minutes ago	1.5min	-	01133031@...	ML_project	-	xgboost
6 minutes ago	59.2s	-	01133031@...	ML_project	-	xgboost
7 minutes ago	51.0s	-	01133031@...	ML_project	-	xgboost
7 minutes ago	1.2min	-	01133031@...	ML_project	-	xgboost
7 minutes ago	1.1min	-	01133031@...	ML_project	-	xgboost
7 minutes ago	1.3min	-	01133031@...	ML_project	-	xgboost
8 minutes ago	52.5s	-	01133031@...	ML_project	-	xgboost
9 minutes ago	2.0min	-	01133031@...	ML_project	-	xgboost
9 minutes ago	1.9min	-	01133031@...	ML_project	-	xgboost

Figure 11: Runs to compare

Now we can go deeply how the hyperparameter choice correlates with AUC. TO do so, you need to lick the "+" icon to expand the parent run, then select all runs except the parent, and click "Compare" and select the Parallel Coordinates Plot with parameters that we are interested in.

Let's take parameters:

-learning_rate (which is is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function),

-max_depth (xgboost parameter,which is defined as the longest path between the root node and the leaf node in random forest or a maximum number of nodes allowed from the root to the farthest leaf of a tree),

-mid_child_weight (xgboost parameter, it is the minimum weight (or number of samples) required in order to create a new node in the tree).

These two can be used to control the complexity of the trees.

Then, also two parameters concerning regularization, that is used to create less complex model when you have many features in your dataset, these are:

-reg_apha (xgboost parameter, L1 regularization term on weights) and

-reg_lambda (xgboost parameter, L2 regularization term on weights)

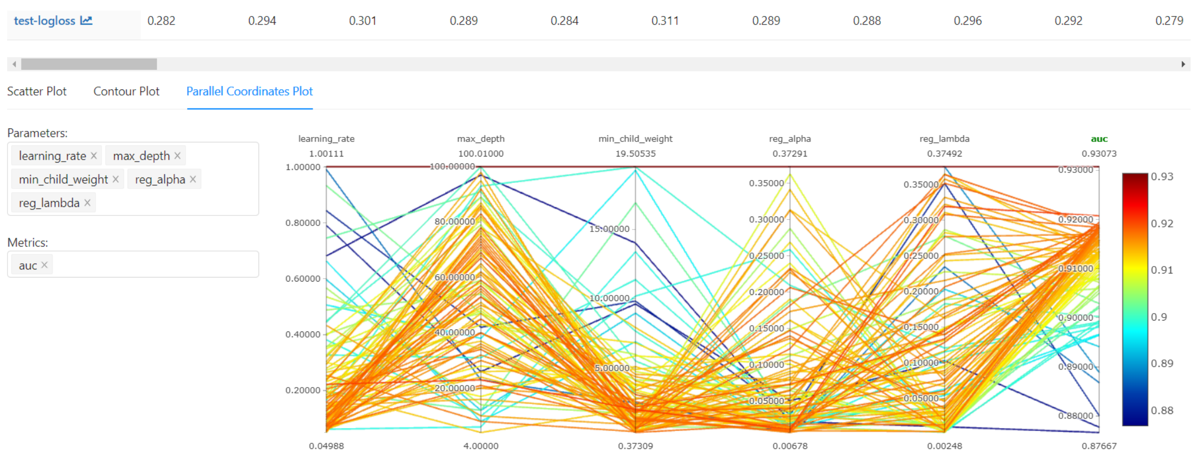
Increasing these values (alpha or lambda) will make model more conservative. A regression model that uses L1 regularization technique is called Lasso Regression (Least Absolute Shrinkage and Selection Operator and L2 is called Ridge Regression. L1 adds "absolute value of magnitude" of coefficient as penalty term to the loss function and L2 adds "squared magnitude". The main difference between these techniques is that Lasso shrinks the less important feature's coefficient to zero with removing some feature altogether. This works well for feature selection when we have a large number of features.[5]

And all these parameters described we need to compare with metric auc.

The Parallel Coordinates Plot is a useful tool to see the impact of parameters on a metric. The plot below highlights the highest AUC values:

We can see that all of the top performing runs have a low value for reg_lambda and learning_rate.

[5]Based on `https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c`

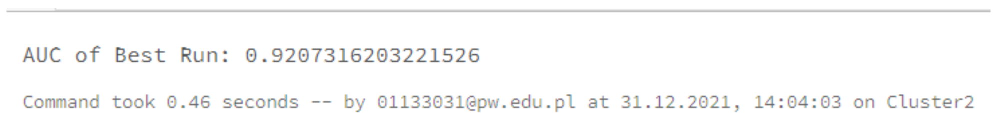


test-logloss	0.282	0.294	0.301	0.289	0.284	0.311	0.289	0.288	0.296	0.292	0.279

Figure 12: Coordinates Plot

Then, to saves the model to Model Registry with the best performing run we will run the following:

```python
best_run = mlflow.search_runs(order_by=['metrics.auc DESC']).iloc[0]
print(f'AUC of Best Run: {best_run["metrics.auc"]}')
```

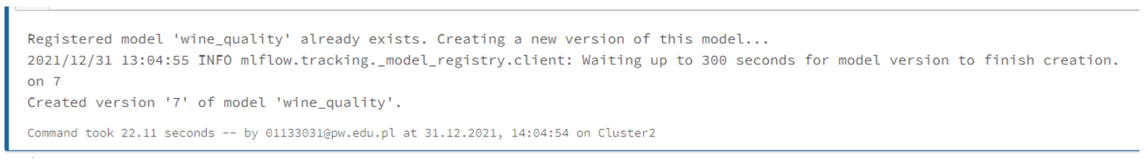


```
AUC of Best Run: 0.9207316203221526

Command took 0.46 seconds -- by 01133031@pw.edu.pl at 31.12.2021, 14:04:03 on Cluster2
```

Figure 13: AUC

Then. We can update the model in Model Registry with previous name "wine_quality"

```python
new_model_version = mlflow.register_model(f"runs:/{best_run.run_id}/model", model_name)

time.sleep(15)
```



```
Registered model 'wine_quality' already exists. Creating a new version of this model...
2021/12/31 13:04:55 INFO mlflow.tracking._model_registry.client: Waiting up to 300 seconds for model version to finish creation.
on 7
Created version '7' of model 'wine_quality'.

Command took 22.11 seconds -- by 01133031@pw.edu.pl at 31.12.2021, 14:04:54 on Cluster2
```

Figure 14: Model registration as new version

4 Conclusion

In this example we have covered the parallel hyperparameter sweep to train machine learning models o the dataset, then explore the results of the hyperparameter sweep with MLflow. The results showed that the hyperparameter tuning improved the AUC result, so it improved in predictions. Also the usage of SparkTrails to conduct the parallel computations enhanced speedups (there where 96 total succeeded trials computed with parallelism = 10). The whole process of trails was register and tracked in MLflow.