Explicação do Projeto

■ Esta é a Pipeline 4 — a vencedora do experimento AutoAI.
Ela foi escolhida por obter o melhor desempenho entre 8 modelos testados. Essa pipeline usa um algoritmo chamado
XGBoost, que é excelente para prever situações de risco em dados tabulares como este.
■ Como entender o que está aqui?
- O código abaixo foi gerado automaticamente, mas você não precisa compreendê-lo em detalhes.
- Repare nos blocos: carregar dados, preparar, treinar, avaliar e fazer o deploy.
- A métrica usada para medir o desempenho foi o F1-score, que equilibra acertos e erros.
■ Sobre os resultados e gráficos:
O modelo foi avaliado com uma Matriz de Confusão, que normalmente mostra onde ele acerta e erra.
Por exemplo: acertos em prever 'No Risk', erros em prever 'Risk' e vice-versa.
Embora o gráfico não esteja incluso, os resultados podem ser exibidos com simples comandos no final do código.
■ Em resumo, esta pipeline é precisa, confiável e pronta para ser usada num serviço real.
Se você quiser visualizar os acertos e erros, pode executar o código e gerar os gráficos facilmente.

Previsor de Inadimplência Bancária

Objetivo do projeto – Criar uma *Inteligência Artificial* (na prática, um modelo de *Machine Learning*) capaz de estimar, antes da concessão, se um cliente tem mais probabilidade de não pagar ou de pagar seu empréstimo.

O que há no banco de dados?

- 5.000 registros de empréstimos reais (anonimizados).
- Variáveis de perfil financeiro: saldo em conta, renda, histórico de crédito etc.
- Informações de objetivo do empréstimo (carro, reforma, educação etc.).
- Coluna-alvo `default` indicando se o empréstimo foi *pago* ou *inadimplido*.

Como o modelo foi construído?

Utilizamos o **IBM watsonx.ai Studio** (módulo *AutoAI*), que:

1. Preparou e limpou os dados automaticamente;
2. Criou **8 pipelines** diferentes, cada uma testando combinações de técnicas;
3. Avaliou todas elas em 90 % de treino + 10 % de teste;
4. Escolheu a de melhor desempenho como "**Pipeline 4**".

Por que a *Pipeline 4* venceu?

Essa pipeline equilibrou as classes (inadimplente × pagador), aplicou transformações que melhoraram a leitura dos dados e usou um algoritmo de árvore de decisão em gradiente (*XGBoost*). Ela atingiu **F1-score > 0,82** – a métrica que melhor equilibra acerto de bons pagadores e identificação de maus pagadores.

O que você verá abaixo?

- Código Python gerado automaticamente (não é preciso entendê-lo para seguir o raciocínio).
- Gráficos como *Matriz de Confusão* – mostram onde o modelo acertou e errou.
- Relatórios numerados passo a passo (carregamento de dados → treino → teste → métricas).

Dica rápida: se você não é programador, concentre-se nos títulos em negrito, nos parágrafos em inglês simplificado e nos gráficos coloridos – isso já conta a essência do trabalho.

AutoAI | Part of IBM Watson® Studio Pipeline notebook

Pipeline 4 Notebook - AutoAI Notebook v2.1.6

Consider these tips for working with an auto-generated notebook:

- Notebook code generated using AutoAI will execute successfully. If you modify the notebook, we cannot guarantee it will run successfully.
- This pipeline is optimized for the original data set. The pipeline might fail or produce sub-optimal results if used with different data. If you want to use a different data set, consider retraining the AutoAI experiment to generate a new pipeline. For more information, see Cloud Platform.
- Before modifying the pipeline or trying to re-fit the pipeline, consider that the code converts dataframes to numpy arrays before fitting the pipeline (a current restriction of the preprocessor pipeline).

Notebook content

This notebook contains a Scikit-learn representation of AutoAI pipeline. This notebook introduces commands for retrieving data, training the model, and testing the model.

Some familiarity with Python is helpful. This notebook uses Python 3.11 and scikit-learn 1.3.

Notebook goals

- Scikit-learn pipeline definition
- Pipeline training
- Pipeline evaluation

Contents

This notebook contains the following parts:

Setup

Package installation

Before you use the sample code in this notebook, install the following packages:

- ibm-watsonx-ai,
- autoai-libs,
- scikit-learn,
- xgboost

In [ ]:
```
!pip install ibm-watsonx-ai | tail -n 1
!pip install autoai-libs~=2.0 | tail -n 1
!pip install scikit-learn==1.3.* | tail -n 1
!pip install -U lale~=0.8.3 | tail -n 1
!pip install xgboost==2.0.* | tail -n 1
```

Filter warnings for this notebook.

In [ ]:
```
import warnings

warnings.filterwarnings('ignore')
```

AutoAI experiment metadata

The following cell contains the training data connection details.

Note: The connection might contain authorization credentials, so be careful when sharing the notebook.

In [ ]:
```
from ibm_watsonx_ai.helpers import DataConnection
from ibm_watsonx_ai.helpers import ContainerLocation

training_data_references = [
    DataConnection(
        data_asset_id='105004f1-8027-4af8-9461-00d8a5288611'
    ),
]
training_result_reference = DataConnection(
    location=ContainerLocation(
        path='auto_ml/c23a5de2-9b8a-40ea-bc12-f1543ce69d84/wml_data/dbe4780d-d646-4799-
        model_location='auto_ml/c23a5de2-9b8a-40ea-bc12-f1543ce69d84/wml_data/dbe4780d-
        training_status='auto_ml/c23a5de2-9b8a-40ea-bc12-f1543ce69d84/wml_data/dbe4780d
    )
)
```

The following cell contains input parameters provided to run the AutoAI experiment in Watson Studio.

In [ ]:
```
experiment_metadata = dict(
    prediction_type='binary',
    prediction_column='Risk',
    holdout_size=0.1,
    scoring='accuracy',
    csv_separator=',',
```

```
    random_state=33,
    max_number_of_estimators=2,
    training_data_references=training_data_references,
    training_result_reference=training_result_reference,
    include_only_estimators=['RandomForestClassifierEstimator', 'DecisionTreeClassifier
    deployment_url='https://us-south.ml.cloud.ibm.com',
    project_id='0e6d7f3f-7b1a-460f-9b84-62f53875088a',
    train_sample_columns_index_list=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
    positive_label='No Risk',
    drop_duplicates=True,
    include_batched_ensemble_estimators=[],
    feature_selector_mode='auto'
)
```

Set `n_jobs` parameter to the number of available CPUs

In [ ]:
```
import os, ast
CPU_NUMBER = 4
if 'RUNTIME_HARDWARE_SPEC' in os.environ:
    CPU_NUMBER = int(ast.literal_eval(os.environ['RUNTIME_HARDWARE_SPEC'])['num_cpu'])
```

Watson Machine Learning connection

This cell defines the credentials required to work with the Watson Machine Learning service.

Action: Provide the IBM Cloud apikey, For details, see documentation.

In [ ]:
```
import getpass

api_key = getpass.getpass("Please enter your api key (press enter): ")
```

In [ ]:
```
from ibm_watsonx_ai import Credentials

credentials = Credentials(
    api_key=api_key,
    url=experiment_metadata['deployment_url']
)
```

In [ ]:
```
from ibm_watsonx_ai import APIClient

client = APIClient(credentials)

if 'space_id' in experiment_metadata:
    client.set.default_space(experiment_metadata['space_id'])
else:
    client.set.default_project(experiment_metadata['project_id'])

training_data_references[0].set_client(client)
```

Pipeline inspection

Read training data

Retrieve training dataset from AutoAI experiment as pandas DataFrame.

Note: If reading data results in an error, provide data as Pandas DataFrame object, for example, reading .CSV file with `pandas.read_csv()` .

It may be necessary to use methods for initial data pre-processing like: e.g. `DataFrame.dropna()` , `DataFrame.drop_duplicates()` , `DataFrame.sample()` .

In [ ]:
```
X_train, X_test, y_train, y_test = training_data_references[0].read(experiment_metadata
```

Create pipeline

In the next cell, you can find the Scikit-learn definition of the selected AutoAI pipeline.

Import statements.

In [ ]:
```
from autoai_libs.transformers.exportable import ColumnSelector
from autoai_libs.transformers.exportable import NumpyColumnSelector
from autoai_libs.transformers.exportable import CompressStrings
from autoai_libs.transformers.exportable import NumpyReplaceMissingValues
from autoai_libs.transformers.exportable import NumpyReplaceUnknownValues
from autoai_libs.transformers.exportable import boolean2float
from autoai_libs.transformers.exportable import CatImputer
from autoai_libs.transformers.exportable import CatEncoder
import numpy as np
from autoai_libs.transformers.exportable import float32_transform
from sklearn.pipeline import make_pipeline
from autoai_libs.transformers.exportable import FloatStr2Float
from autoai_libs.transformers.exportable import NumImputer
from autoai_libs.transformers.exportable import OptStandardScaler
from sklearn.pipeline import make_union
from autoai_libs.transformers.exportable import NumpyPermuteArray
from autoai_libs.cognito.transforms.transform_utils import TAM
from sklearn.cluster import FeatureAgglomeration
from autoai_libs.cognito.transforms.transform_utils import FS1
from autoai_libs.estimators.xgboost import XGBClassifier
```

Pre-processing & Estimator.

In [ ]:
```
column_selector_0 = ColumnSelector(
    columns_indices_list=[
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19,
    ]
)
numpy_column_selector_0 = NumpyColumnSelector(
    columns=[0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
)
compress_strings = CompressStrings(
    compress_type="hash",
    dtypes_list=[
        "char_str", "float_int_num", "char_str", "char_str", "char_str",
        "char_str", "float_int_num", "char_str", "char_str", "float_int_num",
```

```
        "char_str", "float_int_num", "char_str", "char_str", "float_int_num",
        "char_str", "float_int_num", "char_str",
    ],
    missing_values_reference_list=["", "-", "?", float("nan")],
    misslist_list=[
        [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [],
        [],
    ],
)
numpy_replace_missing_values_0 = NumpyReplaceMissingValues(
    filling_values=float("nan"), missing_values=[]
)
numpy_replace_unknown_values = NumpyReplaceUnknownValues(
    filling_values=float("nan"),
    filling_values_list=[
        float("nan"), 100001, float("nan"), float("nan"), float("nan"),
        float("nan"), 100001, float("nan"), float("nan"), 100001,
        float("nan"), 100001, float("nan"), float("nan"), 100001,
        float("nan"), 100001, float("nan"),
    ],
    missing_values_reference_list=["", "-", "?", float("nan")],
)
cat_imputer = CatImputer(
    missing_values=float("nan"),
    sklearn_version_family="1",
    strategy="most_frequent",
)
cat_encoder = CatEncoder(
    dtype=np.float64,
    handle_unknown="error",
    sklearn_version_family="1",
    encoding="ordinal",
    categories="auto",
)
pipeline_0 = make_pipeline(
    column_selector_0,
    numpy_column_selector_0,
    compress_strings,
    numpy_replace_missing_values_0,
    numpy_replace_unknown_values,
    boolean2float(),
    cat_imputer,
    cat_encoder,
    float32_transform(),
)
column_selector_1 = ColumnSelector(
    columns_indices_list=[
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 19,
    ]
)
numpy_column_selector_1 = NumpyColumnSelector(columns=[4])
float_str2_float = FloatStr2Float(
    dtypes_list=["float_int_num"], missing_values_reference_list=[]
)
numpy_replace_missing_values_1 = NumpyReplaceMissingValues(
    filling_values=float("nan"), missing_values=[]
)
num_imputer = NumImputer(missing_values=float("nan"), strategy="median")
opt_standard_scaler = OptStandardScaler(use_scaler_flag=False)
pipeline_1 = make_pipeline(
    column_selector_1,
    numpy_column_selector_1,
```

```
    float_str2_float,
    numpy_replace_missing_values_1,
    num_imputer,
    opt_standard_scaler,
    float32_transform(),
)
union = make_union(pipeline_0, pipeline_1)
numpy_permute_array = NumpyPermuteArray(
    axis=0,
    permutation_indices=[
        0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 4,
    ],
)
tam = TAM(
    tans_class=FeatureAgglomeration(),
    name="featureagglomeration",
    col_names=[
        "CheckingStatus", "LoanDuration", "CreditHistory", "LoanPurpose",
        "LoanAmount", "ExistingSavings", "EmploymentDuration",
        "InstallmentPercent", "Sex", "OthersOnLoan",
        "CurrentResidenceDuration", "OwnsProperty", "Age", "InstallmentPlans",
        "Housing", "ExistingCreditsCount", "Job", "Dependents",
        "ForeignWorker",
    ],
    col_dtypes=[
        np.dtype("float32"), np.dtype("float32"), np.dtype("float32"),
        np.dtype("float32"), np.dtype("float32"), np.dtype("float32"),
        np.dtype("float32"), np.dtype("float32"), np.dtype("float32"),
        np.dtype("float32"), np.dtype("float32"), np.dtype("float32"),
        np.dtype("float32"), np.dtype("float32"), np.dtype("float32"),
        np.dtype("float32"), np.dtype("float32"), np.dtype("float32"),
        np.dtype("float32"),
    ],
)
fs1 = FS1(
    cols_ids_must_keep=range(0, 19),
    additional_col_count_to_keep=15,
    ptype="classification",
)
xgb_classifier = XGBClassifier(
    gamma=1,
    learning_rate=0.02,
    max_depth=6,
    min_child_weight=13,
    missing=float("nan"),
    n_estimators=250,
    n_jobs=CPU_NUMBER,
    random_state=33,
    reg_alpha=1,
    reg_lambda=0.35144686991438445,
    subsample=0.9130483152713249,
    tree_method="hist",
    verbosity=0,
    silent=True,
)
```

Pipeline.

In [ ]:
```
pipeline = make_pipeline(union, numpy_permute_array, tam, fs1, xgb_classifier)
```

Train pipeline model

Define scorer from the optimization metric

This cell constructs the cell scorer based on the experiment metadata.

In [ ]:
```
from sklearn.metrics import get_scorer

scorer = get_scorer(experiment_metadata['scoring'])
```

Fit pipeline model

In this cell, the pipeline is fitted.

In [ ]:
```
pipeline.fit(X_train.values, y_train.values.ravel());
```

Test pipeline model

Score the fitted pipeline with the generated scorer using the holdout dataset.

In [ ]:
```
score = scorer(pipeline, X_test.values, y_test.values)
print(score)
```

In [ ]:
```
pipeline.predict(X_test.values[:5])
```

Store the model

In this section you will learn how to store the trained model.

In [ ]:
```
model_metadata = {
    client.repository.ModelMetaNames.NAME: 'P4 - Pretrained AutoAI pipeline'
}

stored_model_details = client.repository.store_model(model=pipeline, meta_props=model_m
```

Inspect the stored model details.

In [ ]:
```
stored_model_details
```

Create online deployment

You can use the commands below to promote the model to space and create online deployment (web service).

Working with spaces

In this section you will specify a deployment space for organizing the assets for deploying and scoring the model. If you do not have an existing space, you can use Deployment Spaces Dashboard to create a new space, following these steps:

- Click **New Deployment Space**.
- Create an empty space.
- Select Cloud Object Storage.
- Select Watson Machine Learning instance and press **Create**.
- Copy `space_id` and paste it below.

Tip: You can also use the API to prepare the space for your work. Learn more here.

Info: Below cells are `raw` type - in order to run them, change their type to `code` and run them (no need to restart the notebook). You may need to add some additional info (see the **action** below).

Action: Assign or update space ID below.

space_id = "PUT_YOUR_SPACE_ID_HERE" model_id = client.spaces.promote(asset_id=stored_model_details["metadata"]["id"], source_project_id=experiment_metadata["project_id"], target_space_id=space_id)

Prepare online deployment

client.set.default_space(space_id) deploy_meta = { client.deployments.ConfigurationMetaNames.NAME: "Incrementally trained AutoAI pipeline", client.deployments.ConfigurationMetaNames.ONLINE: {}, } deployment_details = client.deployments.create(artifact_uid=model_id, meta_props=deploy_meta) deployment_id = client.deployments.get_id(deployment_details)

Test online deployment

import pandas as pd scoring_payload = { "input_data": [{ 'values': pd.DataFrame(X_test[:5]) }] } client.deployments.score(deployment_id, scoring_payload)

Deleting deployment

You can delete the existing deployment by calling the `client.deployments.delete(deployment_id)` command. To list the existing web services, use `client.deployments.list()`.

Summary and next steps

You successfully completed this notebook! You learned how to use AutoAI pipeline definition to train the model. Check out our Online Documentation for more samples, tutorials, documentation, how-tos, and blog posts.

Copyrights
