1 INTRODUCTION

1.1 Overview

This project discusses building a asystem for creating predictions that can be used in different scenarios. It focuses on predicting fraudulent transactions, which can reduce monetary loss and risk mitigation.

1.2 Purpose

This project aims at building a web App which automatically estimates if there is a fraud risk by taking the input values.

2 LITERATURE SURVEY

2.1 Existing problem

Prediction is done using machine learning algorithms by importing required libraries and function and this task in existing studies is not automated using IBM Auto AI.

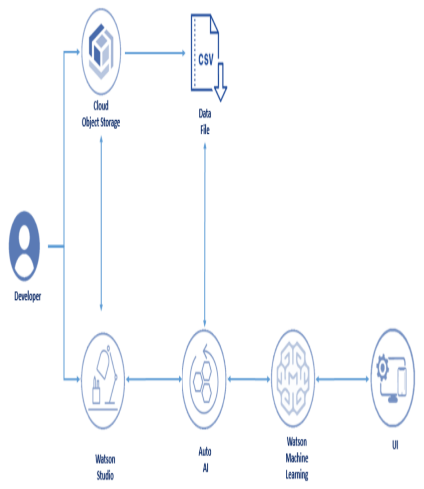
AI Model building process has been reduced from Days to Hours thanks to AutoAI. If you are a developer or a data scientist who wants to build the model quickly and deploy it for being production ready, then AutoAI is for you which will help in taking decisions faster and gives a detailed overview of the attribute relationships within the data.

2.2 Proposed solution

Using IBM AutoAI, we automate all of the tasks involved in building predictive models for different requirements. You create a model from a data set that includes the gender, married, dependents, education, self employed, applicant income, co-applicant income, loan amount, loan term, credit history, housing and locality.

3 THEORITICAL ANALYSIS

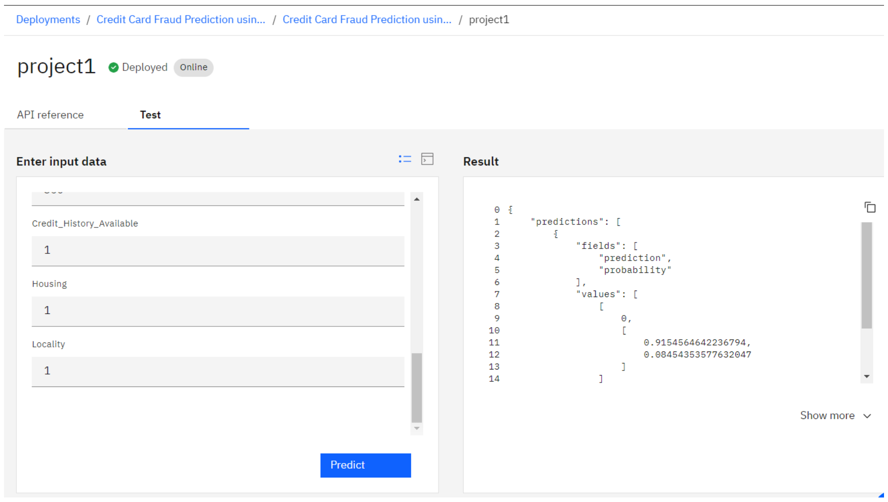
3.1 Block diagram



3.2 Hardware / Software designing

1. IBM Watson Studio
2. IBM Watson Machine Learning
3. Node-RED
4. IBM Cloud Object Storage

EXPERIMENTAL INVESTIGATIONS


Deployments / Credit Card Fraud Prediction usin... / Credit Card Fraud Prediction usin... / project1
project1
Deployed
Online
API reference
Test
Enter input data
Result
Show more
Predict

Credit_History_Available: 1

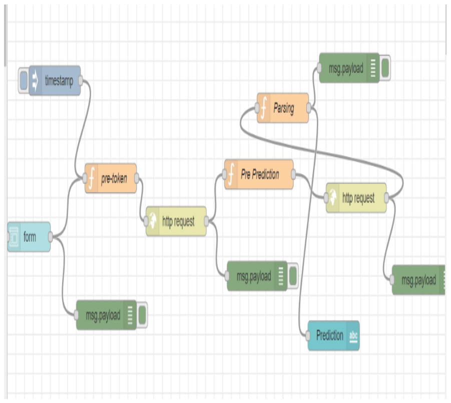
Housing: 1

Locality: 1

```
{
    "predictions": [
        {
            "fields": [
                "prediction",
                "probability"
            ],
            "values": [
                [
                    0,
                    [
                        0.9154564642236794,
                        0.08454353577632047
                    ]
                ]
```

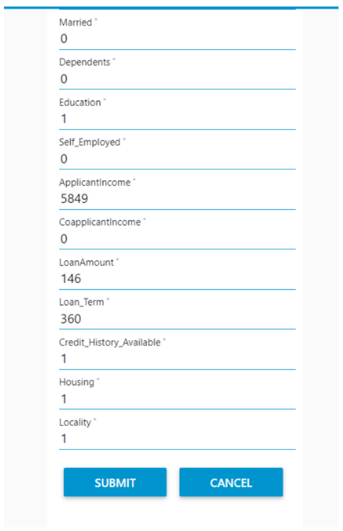
4

FLOWCHART



5

RESULT



Married *
0

Dependents *
0

Education *
1

Self_Employed *
0

ApplicantIncome *
5849

CoapplicantIncome *
0

LoanAmount *
146

Loan_Term *
360

Credit_History_Available *
1

Housing *
1

Locality *
1

6

ADVANTAGES & DISADVANTAGES

ADVANTAGES

1. Fast model selection. Select top-performing models in only minutes.

2. Start quickly. Get started with experimentation, evaluation and deployment.

3. AI lifecycle management. Enforce consistency and repeatability of end-to-end ML and AI development. 7

DISADVANTAGES

1. Maintenance
2. Doesn't process structured data directly
3. Increasing rate of data, with limited resources

APPLICATIONS

1. Healthcare.
2. Legal.
3. Retail.

8 4. Financial.

CONCLUSION

The project focuses on predicting fraudulent transactions, which can reduce monetary loss and risk mitigation. **9**

FUTURE SCOPE

Scale to deep learning to analyze it deeply.

10 `

BIBILOGRAPHY

1. **Author:** Maganti Syamala, Assistant Professor, Department of Computer Science and Engineering, K L Deemed to be University, Green Fields, Vaddeswaram, Andhra Pradesh 522502.

2.**Project Titile:**

11 SPS-6811-Credit-Card-Fraud-Pre

diction-using-IBM-Auto-AI

APPENDIX

A. Source code

1. Set Up¶

```
try:
    import autoai_libs
except Exception as e:
    import subprocess
    out =
subprocess.check_output('pip
install autoai-libs'.split(' '))
    for line in out.splitlines():
        print(line)
    import autoai_libs
import sklearn
try:
    import xgboost
except:
    print('xgboost, if needed, will be
installed and imported later')
try:
    import lightgbm
except:
    print('lightgbm, if needed, will be
installed and imported later')
from sklearn.cluster import
FeatureAgglomeration
import numpy
from numpy import inf, nan, dtype,
mean
from
autoai_libs.sklearn.custom_scorer
s import CustomScorers
import sklearn.ensemble
from
autoai_libs.cognito.transforms.tra
nsform_utils import TExtras, FC
```

```
from
autoai_libs.transformers.exportabl
e import *
from
autoai_libs.utils.exportable_utils
import *
from sklearn.pipeline import
Pipeline
known_values_list=[]
# compose a decorator to assist
pipeline instantiation via import of
modules and installation of
packages
def decorator_retries(func):
    def install_import_retry(*args,
**kwargs):
        retries = 0
        successful = False
        failed_retries = 0
        while retries < 100 and
failed_retries < 10 and not
successful:
            retries += 1
            failed_retries += 1
            try:
                result = func(*args,
**kwargs)
                successful = True
            except Exception as e:
                estr = str(e)
                if estr.startswith('name ')
and estr.endswith(' is not defined'):
                    try:
                        import importlib
                        module_name =
estr.split("'")[1]
                        module =
importlib.import_module(module_n
ame)
```

```
globals().update({module_name:
module})
                        print('import
successful for ' + module_name)
                        failed_retries -= 1
                    except Exception as
import_failure:
                        print('import of ' +
module_name + ' failed with: ' +
str(import_failure))
                        import subprocess
                        if module_name ==
'lightgbm':
                            try:
                                print('attempting
pip install of ' + module_name)
                                process =
subprocess.Popen('pip install ' +
module_name, shell=True)
                                process.wait()
                            except Exception
as E:
                                print(E)
                                try:
                                    import sys

print('attempting conda install of ' +
module_name)
                                    process =
subprocess.Popen('conda install
--yes --prefix {sys.prefix} -c powerai
' + module_name, shell = True)
                                    process.wait()
                                except Exception
as lightgbm_installation_error:
                                    print('lightgbm
installation failed!' +
lightgbm_installation_error)
```

```
                    else:
                        print('attempting
pip install of ' + module_name)
                        process =
subprocess.Popen('pip install ' +
module_name, shell=True)
                        process.wait()
                    try:
                        print('re-attempting
import of ' + module_name)
                        module =
importlib.import_module(module_n
ame)

globals().update({module_name:
module})
                        print('import
successful for ' + module_name)
                        failed_retries -= 1
                    except Exception as
import_or_installation_failure:
                        print('failure
installing and/or importing ' +
module_name + ' error was: ' + str(

import_or_installation_failure))
                        raise
(ModuleNotFoundError('Missing
package in environment for ' +
module_name +
                                               '?
Try import and/or pip install
manually?'))
              elif type(e) is
AttributeError:
                  if 'module ' in estr and '
has no attribute ' in estr:
                    pieces = estr.split("'")
                    if len(pieces) == 5:
```

```
                    try:
                        import importlib

print('re-attempting import of ' +
pieces[3] + ' from ' + pieces[1])
                        module =
importlib.import_module('.' +
pieces[3], pieces[1])
                        failed_retries -= 1
                    except:
                        print('failed
attempt to import ' + pieces[3])
                        raise (e)
                else:
                    raise (e)
            else:
                raise (e)
        if successful:
            print('Pipeline successfully
instantiated')
        else:
            raise
(ModuleNotFoundError(
                'Remaining missing
imports/packages in environment?
Retry cell and/or try pip install
manually?'))
        return result
    return install_import_retry
```

2. Compose Pipeline

```
# metadata necessary to replicate
AutoAI scores with the pipeline
_input_metadata = {'separator': ',',
'excel_sheet': 0, 'target_label_name':
'Fraud_Risk', 'learning_type':
'classification', 'subsampling': None,
'pos_label': 1, 'pn': 'P8',
'cv_num_folds': 3, 'holdout_fraction':
0.1, 'optimization_metric':
'accuracy', 'random_state': 33,
'data_source': ''}

# define a function to compose the
pipeline, and invoke it
@decorator_retries
def compose_pipeline():
    import numpy
    from numpy import nan, dtype,
mean
    #
    # composing steps for toplevel
Pipeline
    #
    _input_metadata = {'separator': ',',
'excel_sheet': 0, 'target_label_name':
'Fraud_Risk', 'learning_type':
'classification', 'subsampling': None,
'pos_label': 1, 'pn': 'P8',
'cv_num_folds': 3, 'holdout_fraction':
0.1, 'optimization_metric':
'accuracy', 'random_state': 33,
'data_source': ''}
    steps = []
    #
    # composing steps for
preprocessor Pipeline
    #
    preprocessor__input_metadata =
None
    preprocessor_steps = []
    #
    # composing steps for
preprocessor_features
FeatureUnion
    #
```

```
preprocessor_features_transformer
_list = []
    #
    # composing steps for
preprocessor_features_categorical
Pipeline
    #

preprocessor_features_categorical
__input_metadata = None

preprocessor_features_categorical
_steps = []

preprocessor_features_categorical
_steps.append(('cat_column_select
or',
autoai_libs.transformers.exportabl
e.NumpyColumnSelector(columns=
[0, 1, 2, 3, 4, 8, 9, 10, 11])))

preprocessor_features_categorical
_steps.append(('cat_compress_stri
ngs',
autoai_libs.transformers.exportabl
e.CompressStrings(activate_flag=T
rue, compress_type='hash',
dtypes_list=['int_num', 'int_num',
'int_num', 'int_num', 'int_num',
'int_num', 'int_num', 'int_num',
'int_num'],
missing_values_reference_list=['', '-',
'?', nan], misslist_list=[[], [], [], [], [], [],
[], [], []])))

preprocessor_features_categorical
_steps.append(('cat_missing_repla
cer',
autoai_libs.transformers.exportabl
e.NumpyReplaceMissingValues(filli
ng_values=nan,
missing_values=[])))

preprocessor_features_categorical
_steps.append(('cat_unknown_repl
acer',
autoai_libs.transformers.exportabl
e.NumpyReplaceUnknownValues(fil
ling_values=nan,
filling_values_list=[nan, nan, nan,
nan, nan, nan, nan, nan, nan],
known_values_list=[[0, 1], [0, 1], [0,
1, 2, 3], [0, 1], [0, 1], [12, 14, 36, 60,
68, 84, 107, 109, 120, 159, 160, 178,
180, 197, 214, 215, 218, 240, 250,
281, 295, 296, 300, 306, 316, 323,
324, 328, 337, 338, 341, 343, 346,
360, 404, 418, 421, 459, 460, 465,
466, 476, 480], [0, 1], [0, 1], [1, 2, 3]],
missing_values_reference_list=['', '-',
'?', nan])))

preprocessor_features_categorical
_steps.append(('boolean2float_tran
sformer',
autoai_libs.transformers.exportabl
e.boolean2float(activate_flag=True
)))

preprocessor_features_categorical
_steps.append(('cat_imputer',
autoai_libs.transformers.exportabl
e.CatImputer(activate_flag=True,
missing_values=nan,
sklearn_version_family='20',
strategy='most_frequent')))

preprocessor_features_categorical
```

```
_steps.append(('cat_encoder',
autoai_libs.transformers.exportabl
e.CatEncoder(activate_flag=True,
categories='auto',
dtype=numpy.float64,
encoding='ordinal',
handle_unknown='error',
sklearn_version_family='20')))

preprocessor_features_categorical
_steps.append(('float32_transform
er',
autoai_libs.transformers.exportabl
e.float32_transform(activate_flag=
True)))
    # assembling
preprocessor_features_categorical_
Pipeline

preprocessor_features_categorical
_pipeline =
sklearn.pipeline.Pipeline(steps=pre
processor_features_categorical_st
eps)

preprocessor_features_transformer
_list.append(('categorical',
preprocessor_features_categorical
_pipeline))
    #
    # composing steps for
preprocessor_features_numeric
Pipeline
    #

preprocessor_features_numeric__in
put_metadata = None

preprocessor_features_numeric_st
```

```
eps = []

preprocessor_features_numeric_st
eps.append(('num_column_selecto
r',
autoai_libs.transformers.exportabl
e.NumpyColumnSelector(columns=
[5, 6, 7])))

preprocessor_features_numeric_st
eps.append(('num_floatstr2float_tr
ansformer',
autoai_libs.transformers.exportabl
e.FloatStr2Float(activate_flag=True
, dtypes_list=['int_num', 'int_num',
'int_num'],
missing_values_reference_list=[])))

preprocessor_features_numeric_st
eps.append(('num_missing_replace
r',
autoai_libs.transformers.exportabl
e.NumpyReplaceMissingValues(filli
ng_values=nan,
missing_values=[])))

preprocessor_features_numeric_st
eps.append(('num_imputer',
autoai_libs.transformers.exportabl
e.NumImputer(activate_flag=True,
missing_values=nan,
strategy='median')))

preprocessor_features_numeric_st
eps.append(('num_scaler',
autoai_libs.transformers.exportabl
e.OptStandardScaler(num_scaler_c
opy=None,
num_scaler_with_mean=None,
num_scaler_with_std=None,
use_scaler_flag=False)))

preprocessor_features_numeric_st
eps.append(('float32_transformer',
autoai_libs.transformers.exportabl
e.float32_transform(activate_flag=
True)))
    # assembling
preprocessor_features_numeric_
Pipeline

preprocessor_features_numeric_pi
peline =
sklearn.pipeline.Pipeline(steps=pre
processor_features_numeric_steps
)

preprocessor_features_transformer
_list.append(('numeric',
preprocessor_features_numeric_pi
peline))
    # assembling
preprocessor_features_
FeatureUnion
    preprocessor_features_pipeline =
sklearn.pipeline.FeatureUnion(trans
former_list=preprocessor_features_
transformer_list)

preprocessor_steps.append(('featu
res',
preprocessor_features_pipeline))

preprocessor_steps.append(('perm
uter',
autoai_libs.transformers.exportabl
e.NumpyPermuteArray(axis=0,
permutation_indices=[0, 1, 2, 3, 4, 8,
9, 10, 11, 5, 6, 7])))
    # assembling preprocessor_
Pipeline
    preprocessor_pipeline =
sklearn.pipeline.Pipeline(steps=pre
processor_steps)
    steps.append(('preprocessor',
preprocessor_pipeline))
    #
    # composing steps for cognito
Pipeline
    #
    cognito__input_metadata = None
    cognito_steps = []
    cognito_steps.append(('0',
autoai_libs.cognito.transforms.tran
sform_utils.TAM(tans_class=sklear
n.decomposition.pca.PCA(copy=Tr
ue, iterated_power='auto',
n_components=None,
random_state=None,
svd_solver='auto', tol=0.0,
whiten=False), name='pca',
tgraph=None, apply_all=True,
col_names=['Gender', 'Married',
'Dependents', 'Education',
'Self_Employed', 'ApplicantIncome',
'CoapplicantIncome', 'LoanAmount',
'Loan_Term',
'Credit_History_Available', 'Housing',
'Locality'],
col_dtypes=[dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32')],
col_as_json_objects=None)))
```

```
    cognito_steps.append(('1',
autoai_libs.cognito.transforms.tran
sform_utils.FS1(cols_ids_must_kee
p=range(0, 12),
additional_col_count_to_keep=12,
ptype='classification')))
    cognito_steps.append(('2',
autoai_libs.cognito.transforms.tran
sform_utils.TA1(fun=numpy.tan,
name='tan', datatypes=['float'],
feat_constraints=[autoai_libs.utils.f
c_methods.is_not_categorical],
tgraph=None, apply_all=True,
col_names=['Gender', 'Married',
'Dependents', 'Education',
'Self_Employed', 'ApplicantIncome',
'CoapplicantIncome', 'LoanAmount',
'Loan_Term',
'Credit_History_Available', 'Housing',
'Locality', 'pca_0', 'pca_1', 'pca_2',
'pca_3', 'pca_4', 'pca_5', 'pca_6',
'pca_7', 'pca_8', 'pca_9', 'pca_10',
'pca_11'],
col_dtypes=[dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32')],
col_as_json_objects=None)))
    cognito_steps.append(('3',
autoai_libs.cognito.transforms.tran
sform_utils.FS1(cols_ids_must_kee
p=range(0, 12),
additional_col_count_to_keep=12,
ptype='classification')))
    # assembling cognito_ Pipeline
    cognito_pipeline =
sklearn.pipeline.Pipeline(steps=cog
nito_steps)
    steps.append(('cognito',
cognito_pipeline))
    steps.append(('estimator',
sklearn.ensemble.forest.RandomF
orestClassifier(bootstrap=True,
class_weight='balanced',
criterion='entropy', max_depth=3,
max_features=0.99950521077198
8, max_leaf_nodes=None,
min_impurity_decrease=0.0,
min_impurity_split=None,
min_samples_leaf=1,
min_samples_split=4,
min_weight_fraction_leaf=0.0,
n_estimators=54, n_jobs=4,
oob_score=True, random_state=33,
verbose=0, warm_start=False)))
    # assembling  Pipeline
    pipeline =
sklearn.pipeline.Pipeline(steps=ste
ps)
    return pipeline
pipeline = compose_pipeline()
# metadata necessary to replicate
AutoAI scores with the pipeline
_input_metadata = {'separator': ',',
'excel_sheet': 0, 'target_label_name':
'Fraud_Risk', 'learning_type':
'classification', 'subsampling': None,
'pos_label': 1, 'pn': 'P8',
'cv_num_folds': 3, 'holdout_fraction':
```

```
0.1, 'optimization_metric':
'accuracy', 'random_state': 33,
'data_source': ''}

# define a function to compose the
pipeline, and invoke it
@decorator_retries
def compose_pipeline():
    import numpy
    from numpy import nan, dtype,
mean
    #
    # composing steps for toplevel
Pipeline
    #
    _input_metadata = {'separator': ',',
'excel_sheet': 0, 'target_label_name':
'Fraud_Risk', 'learning_type':
'classification', 'subsampling': None,
'pos_label': 1, 'pn': 'P8',
'cv_num_folds': 3, 'holdout_fraction':
0.1, 'optimization_metric':
'accuracy', 'random_state': 33,
'data_source': ''}
    steps = []
    #
    # composing steps for
preprocessor Pipeline
    #
    preprocessor__input_metadata =
None
    preprocessor_steps = []
    #
    # composing steps for
preprocessor_features
FeatureUnion
    #

preprocessor_features_transformer
```

```
_list = []
    #
    # composing steps for
preprocessor_features_categorical
Pipeline
    #

preprocessor_features_categorical
__input_metadata = None

preprocessor_features_categorical
_steps = []

preprocessor_features_categorical
_steps.append(('cat_column_select
or',
autoai_libs.transformers.exportabl
e.NumpyColumnSelector(columns=
[0, 1, 2, 3, 4, 8, 9, 10, 11])))

preprocessor_features_categorical
_steps.append(('cat_compress_stri
ngs',
autoai_libs.transformers.exportabl
e.CompressStrings(activate_flag=T
rue, compress_type='hash',
dtypes_list=['int_num', 'int_num',
'int_num', 'int_num', 'int_num',
'int_num', 'int_num', 'int_num',
'int_num'],
missing_values_reference_list=['', '-',
'?', nan], misslist_list=[[], [], [], [], [], [],
[], [], []])))

preprocessor_features_categorical
_steps.append(('cat_missing_repla
cer',
autoai_libs.transformers.exportabl
e.NumpyReplaceMissingValues(filli
ng_values=nan,
missing_values=[])))

preprocessor_features_categorical
_steps.append(('cat_unknown_repl
acer',
autoai_libs.transformers.exportabl
e.NumpyReplaceUnknownValues(fil
ling_values=nan,
filling_values_list=[nan, nan, nan,
nan, nan, nan, nan, nan, nan],
known_values_list=[[0, 1], [0, 1], [0,
1, 2, 3], [0, 1], [0, 1], [12, 14, 36, 60,
68, 84, 107, 109, 120, 159, 160, 178,
180, 197, 214, 215, 218, 240, 250,
281, 295, 296, 300, 306, 316, 323,
324, 328, 337, 338, 341, 343, 346,
360, 404, 418, 421, 459, 460, 465,
466, 476, 480], [0, 1], [0, 1], [1, 2, 3]],
missing_values_reference_list=['', '-',
'?', nan])))

preprocessor_features_categorical
_steps.append(('boolean2float_tran
sformer',
autoai_libs.transformers.exportabl
e.boolean2float(activate_flag=True
)))

preprocessor_features_categorical
_steps.append(('cat_imputer',
autoai_libs.transformers.exportabl
e.CatImputer(activate_flag=True,
missing_values=nan,
sklearn_version_family='20',
strategy='most_frequent')))

preprocessor_features_categorical
_steps.append(('cat_encoder',
autoai_libs.transformers.exportabl
e.CatEncoder(activate_flag=True,
categories='auto',
dtype=numpy.float64,
encoding='ordinal',
handle_unknown='error',
sklearn_version_family='20')))

preprocessor_features_categorical
_steps.append(('float32_transform
er',
autoai_libs.transformers.exportabl
e.float32_transform(activate_flag=
True)))
    # assembling
preprocessor_features_categorical_
Pipeline

preprocessor_features_categorical
_pipeline =
sklearn.pipeline.Pipeline(steps=pre
processor_features_categorical_st
eps)

preprocessor_features_transformer
_list.append(('categorical',
preprocessor_features_categorical
_pipeline))
    #
    # composing steps for
preprocessor_features_numeric
Pipeline
    #

preprocessor_features_numeric__in
put_metadata = None

preprocessor_features_numeric_st
eps = []
```

```
preprocessor_features_numeric_st
eps.append(('num_column_selecto
r',
autoai_libs.transformers.exportabl
e.NumpyColumnSelector(columns=
[5, 6, 7])))

preprocessor_features_numeric_st
eps.append(('num_floatstr2float_tr
ansformer',
autoai_libs.transformers.exportabl
e.FloatStr2Float(activate_flag=True
, dtypes_list=['int_num', 'int_num',
'int_num'],
missing_values_reference_list=[])))

preprocessor_features_numeric_st
eps.append(('num_missing_replace
r',
autoai_libs.transformers.exportabl
e.NumpyReplaceMissingValues(filli
ng_values=nan,
missing_values=[])))

preprocessor_features_numeric_st
eps.append(('num_imputer',
autoai_libs.transformers.exportabl
e.NumImputer(activate_flag=True,
missing_values=nan,
strategy='median')))

preprocessor_features_numeric_st
eps.append(('num_scaler',
autoai_libs.transformers.exportabl
e.OptStandardScaler(num_scaler_c
opy=None,
num_scaler_with_mean=None,
num_scaler_with_std=None,
use_scaler_flag=False)))

preprocessor_features_numeric_st
eps.append(('float32_transformer',
autoai_libs.transformers.exportabl
e.float32_transform(activate_flag=
True)))
    # assembling
preprocessor_features_numeric_
Pipeline

preprocessor_features_numeric_pi
peline =
sklearn.pipeline.Pipeline(steps=pre
processor_features_numeric_steps
)

preprocessor_features_transformer
_list.append(('numeric',
preprocessor_features_numeric_pi
peline))
    # assembling
preprocessor_features_
FeatureUnion
    preprocessor_features_pipeline =
sklearn.pipeline.FeatureUnion(trans
former_list=preprocessor_features_
transformer_list)

preprocessor_steps.append(('featu
res',
preprocessor_features_pipeline))

preprocessor_steps.append(('perm
uter',
autoai_libs.transformers.exportabl
e.NumpyPermuteArray(axis=0,
permutation_indices=[0, 1, 2, 3, 4, 8,
9, 10, 11, 5, 6, 7])))
```

```
    # assembling preprocessor_
Pipeline
    preprocessor_pipeline =
sklearn.pipeline.Pipeline(steps=pre
processor_steps)
    steps.append(('preprocessor',
preprocessor_pipeline))
    #
    # composing steps for cognito
Pipeline
    #
    cognito__input_metadata = None
    cognito_steps = []
    cognito_steps.append(('0',
autoai_libs.cognito.transforms.tran
sform_utils.TAM(tans_class=sklear
n.decomposition.pca.PCA(copy=Tr
ue, iterated_power='auto',
n_components=None,
random_state=None,
svd_solver='auto', tol=0.0,
whiten=False), name='pca',
tgraph=None, apply_all=True,
col_names=['Gender', 'Married',
'Dependents', 'Education',
'Self_Employed', 'ApplicantIncome',
'CoapplicantIncome', 'LoanAmount',
'Loan_Term',
'Credit_History_Available', 'Housing',
'Locality'],
col_dtypes=[dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32')],
col_as_json_objects=None)))
    cognito_steps.append(('1',
autoai_libs.cognito.transforms.tran
sform_utils.FS1(cols_ids_must_kee
p=range(0, 12),
additional_col_count_to_keep=12,
ptype='classification')))
    cognito_steps.append(('2',
autoai_libs.cognito.transforms.tran
sform_utils.TA1(fun=numpy.tan,
name='tan', datatypes=['float'],
feat_constraints=[autoai_libs.utils.f
c_methods.is_not_categorical],
tgraph=None, apply_all=True,
col_names=['Gender', 'Married',
'Dependents', 'Education',
'Self_Employed', 'ApplicantIncome',
'CoapplicantIncome', 'LoanAmount',
'Loan_Term',
'Credit_History_Available', 'Housing',
'Locality', 'pca_0', 'pca_1', 'pca_2',
'pca_3', 'pca_4', 'pca_5', 'pca_6',
'pca_7', 'pca_8', 'pca_9', 'pca_10',
'pca_11'],
col_dtypes=[dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32'), dtype('float32'),
dtype('float32')],
col_as_json_objects=None)))
    cognito_steps.append(('3',
autoai_libs.cognito.transforms.tran
sform_utils.FS1(cols_ids_must_kee
p=range(0, 12),
additional_col_count_to_keep=12,
ptype='classification')))
    # assembling cognito_ Pipeline
    cognito_pipeline =
sklearn.pipeline.Pipeline(steps=cog
nito_steps)
    steps.append(('cognito',
cognito_pipeline))
    steps.append(('estimator',
sklearn.ensemble.forest.RandomF
orestClassifier(bootstrap=True,
class_weight='balanced',
criterion='entropy', max_depth=3,
max_features=0.99950521077198
8, max_leaf_nodes=None,
min_impurity_decrease=0.0,
min_impurity_split=None,
min_samples_leaf=1,
min_samples_split=4,
min_weight_fraction_leaf=0.0,
n_estimators=54, n_jobs=4,
oob_score=True, random_state=33,
verbose=0, warm_start=False)))
    # assembling  Pipeline
    pipeline =
sklearn.pipeline.Pipeline(steps=ste
ps)
    return pipeline
pipeline = compose_pipeline()
```

3. Extract needed parameter values from AutoAI run metadata¶

```
#data_source='replace_with_path_and_csv_filename'
target_label_name = _input_metadata['target_label_name']
learning_type = _input_metadata['learning_type']
optimization_metric = _input_metadata['optimization_metric']
random_state = _input_metadata['random_state']
cv_num_folds = _input_metadata['cv_num_folds']
holdout_fraction = _input_metadata['holdout_fraction']
if 'data_provenance' in _input_metadata:
    data_provenance = _input_metadata['data_provenance']
else:
    data_provenance = None
if 'pos_label' in _input_metadata and learning_type == 'classification':
    pos_label = _input_metadata['pos_label']
else:
    pos_label = None
```

4. Create dataframe from dataset in Cloud Object Storage¶

```
# @hidden_cell
# The following code contains the credentials for a file in your IBM Cloud Object Storage.
# You might want to remove those
credentials before you share your
notebook.
credentials_0 = {
    'ENDPOINT':
'https://s3.eu-geo.objectstorage.so
ftlayer.net',
    'IBM_AUTH_ENDPOINT':
'https://iam.bluemix.net/oidc/token
/',
    'APIKEY':
'ZXf0vscPTf9ay8Z1wuyxjyboO3gcl
3S7RVDtV78r5ZyU',
    'BUCKET':
'creditcardfraudpredictionusingibm-
donotdelete-pr-zfkfnodem0wmcu',
    'FILE': 'fraud_dataset.csv',
    'SERVICE_NAME': 's3',
    'ASSET_ID': '1',
    }
#  Read the data as a dataframe
import pandas as pd

csv_encodings=['UTF-8','Latin-1'] #
supplement list of encodings as
necessary for your data
df = None
readable = None  # if automatic
detection fails, you can supply a
filename here

# First, obtain a readable object
# Cloud Object Storage data access
# Assumes COS credentials are in a
dictionary named 'credentials_0'

credentials = df =
globals().get('credentials_0')
if readable is None and credentials
```

```
is not None :
    try:
        import types
        import pandas as pd
        import io
    except Exception as
import_exception:
        print('Error with importing
packages - check if you installed
them on your environment')
    try:
        if
credentials['SERVICE_NAME'] ==
's3':
            try:
                from botocore.client
import Config
                import ibm_boto3
            except Exception as
import_exception:
                print('Installing required
packages!')
                !pip install ibm-cos-sdk
                print('accessing data via
Cloud Object Storage')
            try:
                client =
ibm_boto3.client(service_name=cr
edentials['SERVICE_NAME'],

ibm_api_key_id=credentials['APIKE
Y'],

ibm_auth_endpoint=credentials['IB
M_AUTH_ENDPOINT'],

config=Config(signature_version='o
auth'),
endpoint_url=credentials['ENDPOIN
T'])
      except Exception as
cos_exception:
        print('unable to create
client for cloud object storage')
      try:
        readable =
client.get_object(Bucket=credential
s['BUCKET'],Key=credentials['FILE'])
['Body']
        # add missing __iter__
method, so pandas accepts
readable as file-like object
        if not hasattr(readable,
"__iter__"): readable.__iter__ =
types.MethodType( __iter__,
readable )
      except Exception as
cos_access_exception:
        print('unable to access
data object in cloud object storage
with credentials supplied')
    elif
credentials['SERVICE_NAME'] ==
'fs':
      print('accessing data via File
System')
      try:
        if
credentials['FILE'].endswith('xlsx')
or credentials['FILE'].endswith('xls'):
          df =
pd.read_excel(credentials['FILE'])
        else:
          df =
pd.read_csv(credentials['FILE'], sep
= _input_metadata['separator'])
      except Exception as
```

```
FS_access_exception:
                print('unable to access
data object in File System with path
supplied')
    except Exception as
data_access_exception:
        print('unable to access data
object with credentials supplied')

# IBM Cloud Pak for Data data
access
project_filename =
globals().get('project_filename')
if readable is None and
'credentials_0' in globals() and
'ASSET_ID' in credentials_0:
    project_filename =
credentials_0['ASSET_ID']
if project_filename != None and
project_filename != '1':
    print('attempting project_lib
access to ' + str(project_filename))
    try:
        from project_lib import Project
        project = Project.access()
        storage_credentials =
project.get_storage_metadata()
        readable =
project.get_file(project_filename)
    except Exception as
project_exception:
        print('unable to access data
using the project_lib interface and
filename supplied')

# Use data_provenance as filename
if other access mechanisms are
unsuccessful
if readable is None and
```

```
type(data_provenance) is str:
    print('attempting to access local
file using path and name ' +
data_provenance)
    readable = data_provenance

# Second, use pd.read_csv to read
object, iterating over list of
csv_encodings until successful
if readable is not None:
    for encoding in csv_encodings:
        try:
            if
credentials['FILE'].endswith('xlsx')
or credentials['FILE'].endswith('xls'):
                buffer =
io.BytesIO(readable.read())
                buffer.seek(0)
                df = pd.read_excel(buffer,
encoding=encoding,sheet_name=_i
nput_metadata['excel_sheet'])
            else:
                df = pd.read_csv(readable,
encoding = encoding, sep =
_input_metadata['separator'])
            print('successfully loaded
dataframe using encoding = ' +
str(encoding))
            break
        except Exception as
exception_dataread:
            print('unable to read csv
using encoding ' + str(encoding))
            print('handled error was ' +
str(exception_dataread))
    if df is None:
        print('unable to read file/object
as a dataframe using supplied
csv_encodings ' +
str(csv_encodings))
        print(f'Please use \'insert to
code\' on data panel to load
dataframe.')
        raise(ValueError('unable to
read file/object as a dataframe
using supplied csv_encodings ' +
str(csv_encodings)))

if isinstance(df,pd.DataFrame):
    print('Data loaded succesfully')
    if
_input_metadata.get('subsampling'
) is not None:
        df =
df.sample(frac=_input_metadata['s
ubsampling'],
random_state=_input_metadata['ra
ndom_state']) if
_input_metadata['subsampling'] <=
1.0 else
df.sample(n=_input_metadata['sub
sampling'],
random_state=_input_metadata['ra
ndom_state'])
else:
    print('Data cannot be loaded with
credentials supplied, please
provide DataFrame with training
data.')
```

5. Preprocess Data¶

```
# Drop rows whose target is not
defined
target = target_label_name # your
target name here
if learning_type == 'regression':
    df[target] =
pd.to_numeric(df[target],
errors='coerce')
df.dropna('rows', how='any',
subset=[target], inplace=True)
# extract X and y
df_X = df.drop(columns=[target])
df_y = df[target]
# Detach preprocessing pipeline
(which needs to see all training
data)
preprocessor_index = -1
preprocessing_steps = []
for i, step in
enumerate(pipeline.steps):

preprocessing_steps.append(step)
    if step[0]=='preprocessor':
        preprocessor_index = i
        break
#if len(pipeline.steps) >
preprocessor_index+1 and
pipeline.steps[preprocessor_index +
1][0] == 'cognito':
    #preprocessor_index += 1

#preprocessing_steps.append(pipel
ine.steps[preprocessor_index])
if preprocessor_index >= 0:
    preprocessing_pipeline =
Pipeline(memory=pipeline.memory,
steps=preprocessing_steps)
    pipeline =
Pipeline(steps=pipeline.steps[prepr
ocessor_index+1:])
# Preprocess X
# preprocessor should see all data
for cross_validate on the remaining
steps to match autoai scores
known_values_list.clear()  #
known_values_list is filled in by the
```

```
preprocessing_pipeline if needed
preprocessing_pipeline.fit(df_X.values, df_y.values)
X_prep = preprocessing_pipeline.transform(df_X.values)
```

6. Split data into Training and Holdout sets¶

In [ ]:

```
# determine learning_type and perform holdout split (stratify conditionally)
if learning_type is None:
    # When the problem type is not available in the metadata, use the sklearn type_of_target to determine whether to stratify the holdout split
    # Caution:  This can mis-classify regression targets that can be expressed as integers as multiclass, in which case manually override the learning_type
    from sklearn.utils.multiclass import type_of_target
    if type_of_target(df_y.values) in ['multiclass', 'binary']:
        learning_type = 'classification'
    else:
        learning_type = 'regression'
    print('learning_type determined by type_of_target as:',learning_type)
else:
    print('learning_type specified as:',learning_type)

from sklearn.model_selection import train_test_split
```

```
if learning_type == 'classification':
    X, X_holdout, y, y_holdout =
train_test_split(X_prep, df_y.values,
test_size=holdout_fraction,
random_state=random_state,
stratify=df_y.values)
else:
    X, X_holdout, y, y_holdout =
train_test_split(X_prep, df_y.values,
test_size=holdout_fraction,
random_state=random_state)
```

7. Generate features via Feature Engineering pipeline¶

In [ ]:

```
#Detach Feature Engineering
pipeline if next, fit it, and transform
the training data
fe_pipeline = None
if pipeline.steps[0][0] == 'cognito':
    try:
        fe_pipeline =
Pipeline(steps=[pipeline.steps[0]])
        X = fe_pipeline.fit_transform(X,
y)
        X_holdout =
fe_pipeline.transform(X_holdout)
        pipeline.steps =
pipeline.steps[1:]
    except IndexError:
        try:
            print('Trying to compose
pipeline with some of cognito
steps')
            fe_pipeline = Pipeline(steps
=
list([pipeline.steps[0][1].steps[0],pip
eline.steps[0][1].steps[1]]))
```

```
         X =
fe_pipeline.fit_transform(X, y)
         X_holdout =
fe_pipeline.transform(X_holdout)
         pipeline.steps =
pipeline.steps[1:]
      except IndexError:
         print('Composing pipeline
without cognito steps!')
         pipeline.steps =
pipeline.steps[1:]
```

8. Additional setup: Define a function that returns a scorer for the target's positive label

In [ ]:

```
# create a function to produce a
scorer for a given positive label
def make_pos_label_scorer(scorer,
pos_label):
   kwargs = {'pos_label':pos_label}
   for prop in ['needs_proba',
'needs_threshold']:
      if prop+'=True' in
scorer._factory_args():
         kwargs[prop] = True
   if scorer._sign == -1:
      kwargs['greater_is_better'] =
False
   from sklearn.metrics import
make_scorer

scorer=make_scorer(scorer._score
_func, **kwargs)
   return scorer
```

9. Fit pipeline, predict on

Holdout set, calculate score, perform cross-validation¶

In [ ]:

```
# fit the remainder of the pipeline on
the training data
pipeline.fit(X,y)
# predict on the holdout data
y_pred =
pipeline.predict(X_holdout)
# compute score for the
optimization metric
# scorer may need pos_label, but
not all scorers take pos_label
parameter
from sklearn.metrics import
get_scorer
scorer =
get_scorer(optimization_metric)
score = None
#score = scorer(pipeline, X_holdout,
y_holdout)  # this would suffice for
simple cases
pos_label = None  # if you want to
supply the pos_label, specify it here
if pos_label is None and 'pos_label'
in _input_metadata:

pos_label=_input_metadata['pos_la
bel']
try:
    score = scorer(pipeline,
X_holdout, y_holdout)
except Exception as e1:
    if learning_type is "classification"
and (pos_label is None or
str(pos_label)==''):
        print('You may have to provide
a value for pos_label in order for a
score to be calculated.')
        raise(e1)
    else:
        exception_string=str(e1)
        if 'pos_label' in
exception_string:
            try:
                scorer =
make_pos_label_scorer(scorer,
pos_label=pos_label)
                score = scorer(pipeline,
X_holdout, y_holdout)
                print('Retry was
successful with pos_label supplied
to scorer')
            except Exception as e2:
                print('Initial attempt to use
scorer failed.  Exception was:')
                print(e1)
                print('')
                print('Retry with pos_label
failed.  Exception was:')
                print(e2)
        else:
            raise(e1)

if score is not None:
    print(score)
# cross_validate pipeline using
training data
from sklearn.model_selection
import cross_validate
from sklearn.model_selection
import StratifiedKFold, KFold
if learning_type == 'classification':
    fold_generator =
StratifiedKFold(n_splits=cv_num_fo
lds, random_state=random_state)
else:
    fold_generator =
KFold(n_splits=cv_num_folds,
random_state=random_state)
cv_results =
cross_validate(pipeline, X, y,
cv=fold_generator,
scoring={optimization_metric:score
r}, return_train_score=True)
import numpy as np
np.mean(cv_results['test_' +
optimization_metric])
cv_results
```