Credit Card Fraud Prediction Using IBM Auto AI

by

JAYANT PRATAP SINGH
BBD-NITM

As a part of

IBM Build-a-thon

Introduction

- ## Overview-

This project discusses building a system for creating predictions that can be used in different scenarios. It focuses on predicting fraudulent transactions, which can reduce monetary loss and risk mitigation.
This project aims at building a web App which automatically estimates if there is a fraud risk by taking the input values.
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.

- ## Purpose-

These are not the only challenges in the implementation of a real-world fraud detection

system, however. In real world examples, the massive stream of payment requests is quickly scanned by automatic tools that determine which transactions
to authorize. Machine learning algorithms are employed to analyse all the
authorized transactions and report the suspicious ones. These reports are investigated by professionals who contact the cardholders to confirm if the transaction was genuine or fraudulent. The investigators provide a feedback to the automated system
which is used to train and update the algorithm to eventually improve the fraud-detection performance over time.

Literature Survey

Fraud act as the unlawful or criminal deception intended to result in financial or personal benefit. It is a deliberate act that is against the law, rule or policy with an aim to attain unauthorized financial benefit.

Numerous literatures pertaining to anomaly or fraud detection in this domain have been published already and are available for public usage. A comprehensive survey conducted by Clifton Phua and his associates have revealed that techniques employed in this domain include data mining applications, automated fraud detection, adversarial detection. In another paper, Suman, Research Scholar, GJUS&T at Hisar HCE presented techniques like Supervised and Unsupervised Learning for credit card fraud detection. Even though these methods and algorithms fetched an unexpected success in some areas, they failed to provide a permanent and consistent solution to fraud detection.

A similar research domain was presented by Wen-Fang YU and Na Wang where they used Outlier mining, Outlier detection mining and Distance sum algorithms to accurately predict fraudulent transaction in an emulation experiment of credit card transaction data set of one certain commercial bank. Outlier mining is a field of data mining which is basically used in monetary and internet fields. It deals with detecting objects that are detached from the main system i.e. the transactions that aren't genuine. They have taken attributes of customer's behaviour and based on the value of those attributes they've calculated that distance between the observed value of that attribute and its predetermined value.

Unconventional techniques such as hybrid data mining/complex network classification algorithm is able to perceive illegal instances in an actual card transaction data set, based on network reconstruction algorithm that allows creating representations of the deviation of one instance from a reference group have proved efficient typically on medium sized online transaction.

There have also been efforts to progress from a completely new aspect. Attempts have been made to improve the alert-feedback interaction in case of fraudulent transaction.

In case of fraudulent transaction, the authorised system would be alerted and a feedback would be sent to deny the ongoing transaction.

Artificial Genetic Algorithm, one of the approaches that shed new light in this domain, countered fraud from a different direction.

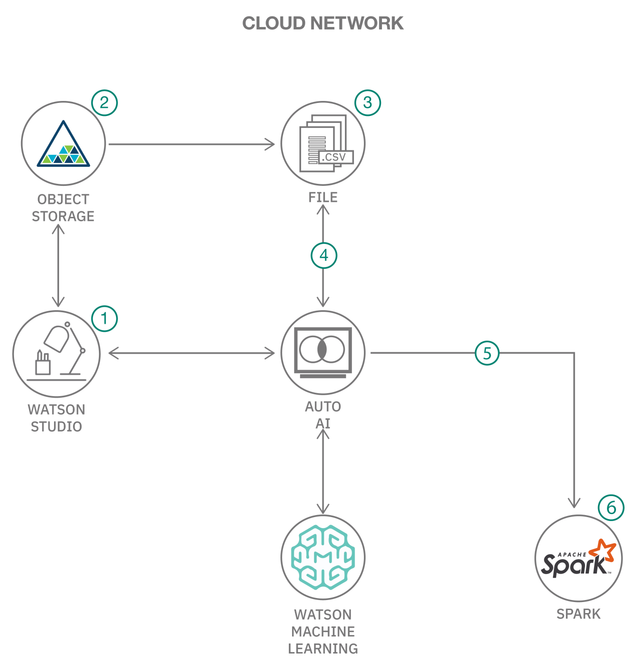
It proved accurate in finding out the fraudulent transactions and minimizing the number of false alerts. Even though, it was accompanied by classification problem with variable misclassification costs.

THEORETICAL ANALYSIS

The approach that this paper proposes, uses the latest machine learning algorithms to detect anomalous activities, called outliers.

The basic rough architecture diagram can be represented with the following figure:

When looked at in detail on a larger scale along with real life elements, the full architecture diagram can be represented as follows:



First of all, we obtained our dataset from Kaggle, a data analysis website which provides datasets.

Inside this dataset, there are 31 columns out of which 28 are named as v1-v28 to protect sensitive data.

The other columns represent Time, Amount and Class. Time shows the time gap between the first transaction and the following one. Amount is the amount of money transacted. Class 0 represents a valid transaction and 1 represents a fraudulent one.

We plot different graphs to check for inconsistencies in the dataset and to visually comprehend it:

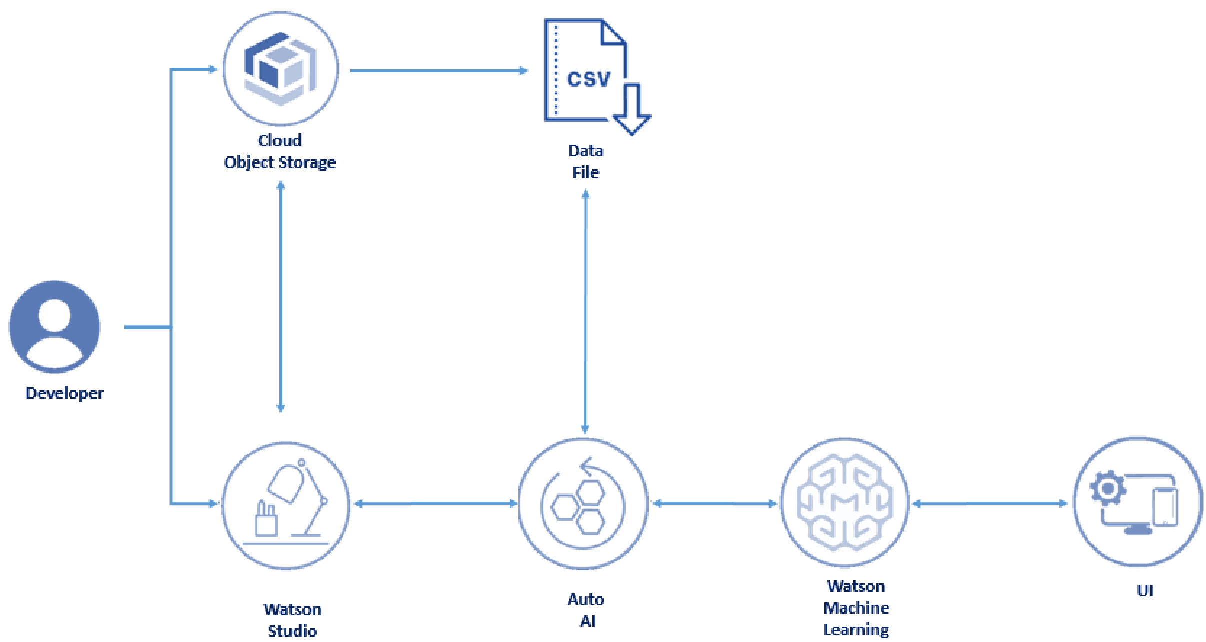
THEORITICAL ANALYSIS

3.1 Block diagram

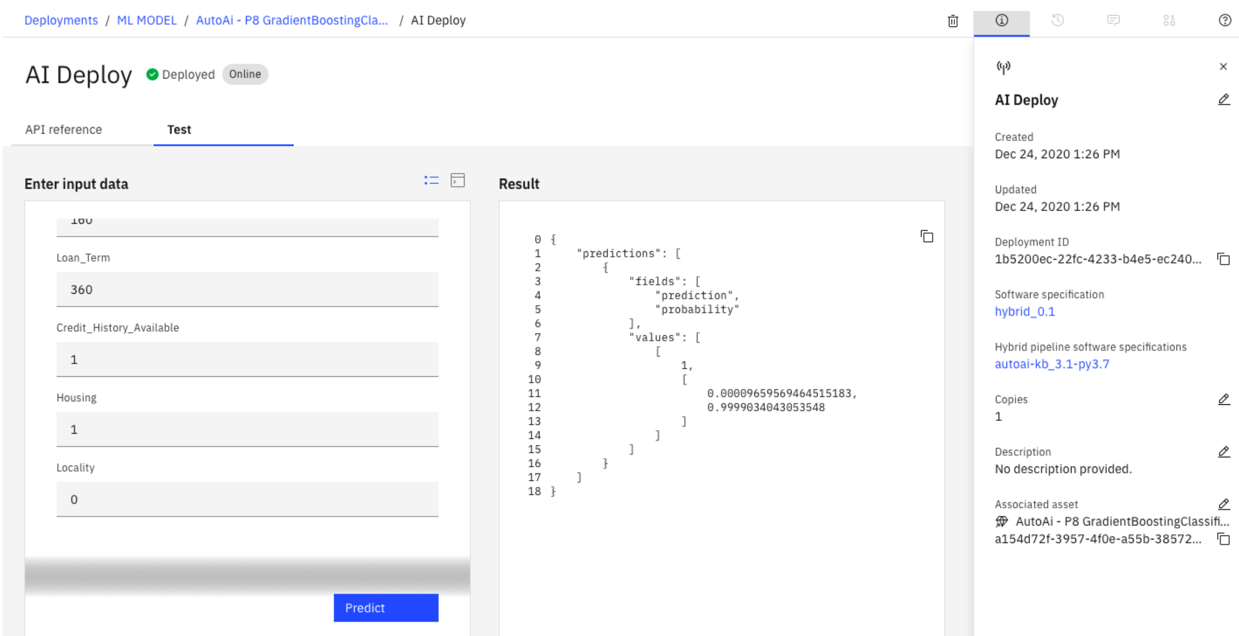
3.2 Hardware / Software designing 1. IBM Watson Studio

2. IBM Watson Machine Learning 3. Node-RED

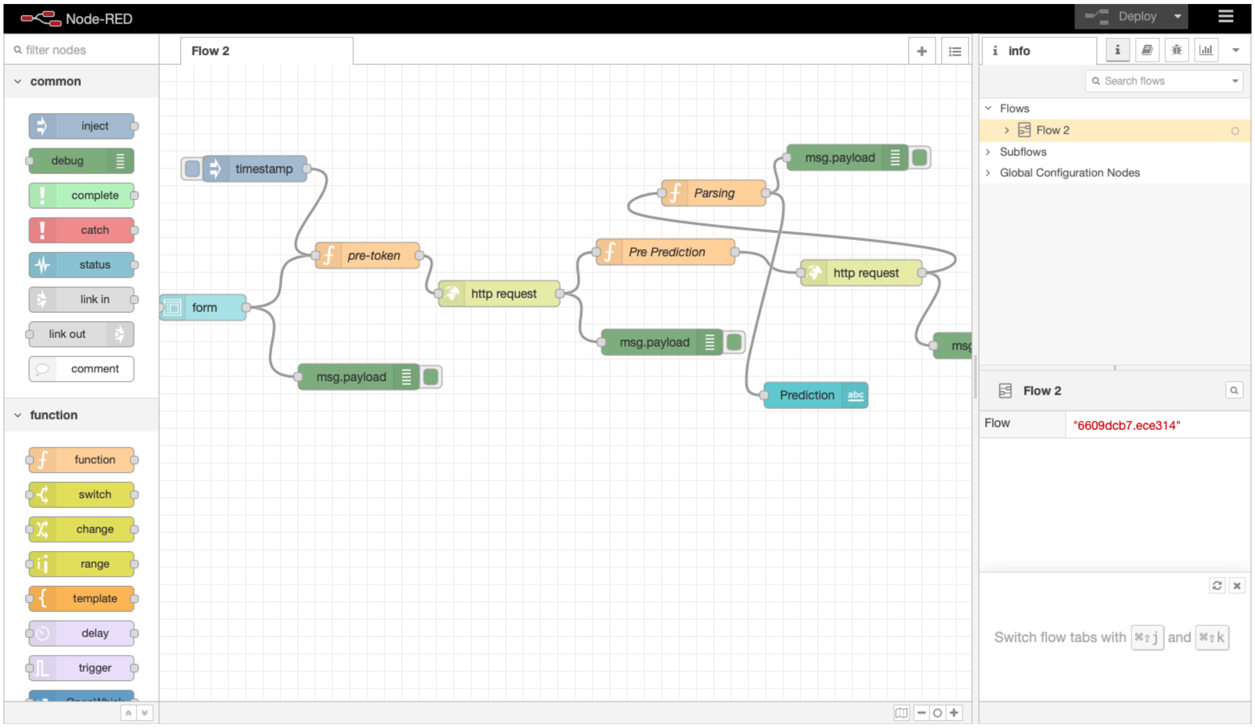
4. IBM Cloud Object Storage



EXPERIMENTAL INVESTIGATIONS



Flowchart



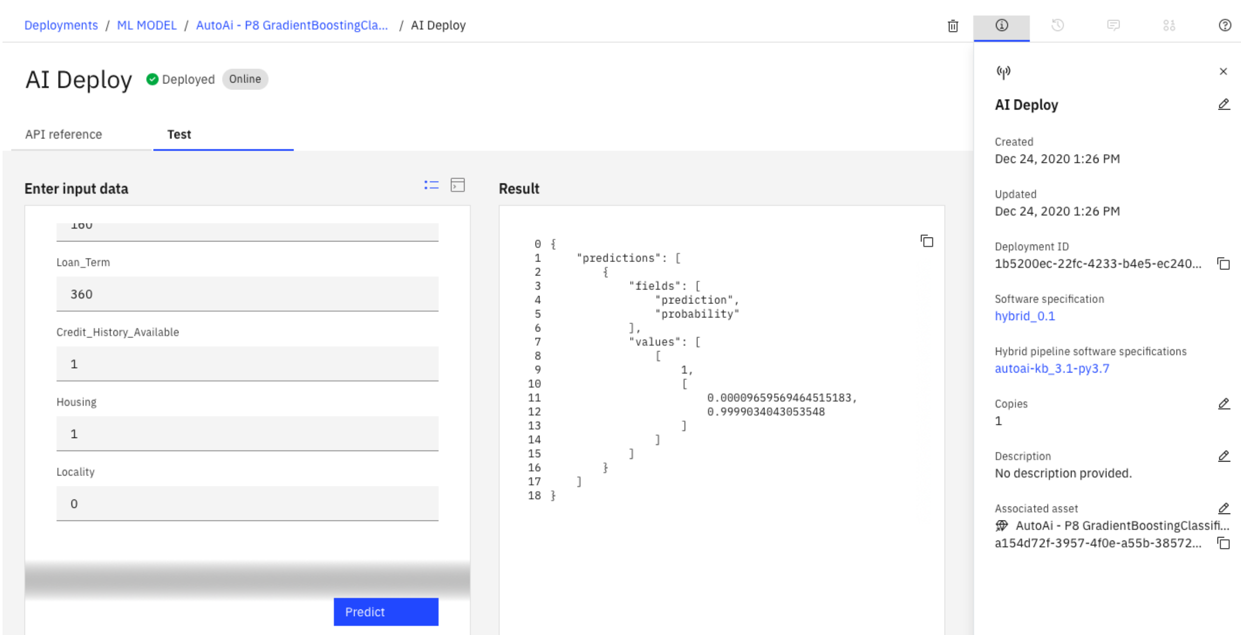
RESULT

The code prints out the number of false positives it detected and compares it with the actual values. This is used to calculate the accuracy score and precision of the algorithms.

The fraction of data we used for faster testing is 10% of the entire dataset. The complete dataset is also used at the end and both the results are printed.

These results along with the classification report for each algorithm is given in the output as follows, where class 0 means the transaction was determined to be valid and 1 means it was determined as a fraud transaction.
This result matched against the class values to check for false positives.

Results when 10% of the dataset is used:



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.

DISADVANTAGES

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

. CONCLUSION

Credit card fraud is without a doubt an act of criminal dishonesty. This article has listed out the most common methods of fraud along with their detection methods and reviewed recent findings in this field. This paper has also explained in detail, how machine learning can be applied to get better results in fraud detection along with the algorithm, pseudocode, explanation its implementation and experimentation results.

While the algorithm does reach over 99.6% accuracy, its precision remains only at 28% when a tenth of the data set is taken into consideration. However, when the entire dataset is fed into the algorithm, the precision rises to 33%. This high percentage of accuracy is to be expected due to the huge imbalance between the number of valid and number of genuine transactions.

Since the entire dataset consists of only two days' transaction records, its only a fraction of data that can be made available if this project were to be used on a commercial scale. Being based on machine learning algorithms, the program will only increase its efficiency over time as more data is put into it.

FUTURE ENHANCEMENTS

While we couldn’t reach out goal of 100% accuracy in fraud detection, we did end up creating a system that can, with enough time and data, get very close to that goal. As with any such project, there is some room for improvement here.

The very nature of this project allows for multiple algorithms to be integrated together as modules and their results can be combined to increase the accuracy of the final result.

This model can further be improved with the addition of more algorithms into it. However, the output of these algorithms needs to be in the same format as the others. Once that condition is satisfied, the modules are easy to add as done in the code. This provides a great degree of modularity and versatility to the project.

More room for improvement can be found in the dataset. As demonstrated before, the precision of the algorithms increases when the size of dataset is increased. Hence, more data will surely make the model more accurate in detecting frauds and reduce the number of false positives. However, this requires official support from the banks themselves.

BIBLIOGRAPHY

[1] “Survey Paper on Credit Card Fraud Detection by Suman” , Research Scholar, GJUS&T Hisar HCE, Sonepat published by International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 3 Issue 3, March 2014

[2] “Research on Credit Card Fraud Detection Model Based on Distance Sum – by Wen-Fang YU and Na Wang” published by 2009 International Joint Conference on Artificial Intelligence

[3] “Credit Card Fraud Detection through Parenclitic Network Analysis- By Massimiliano Zanin, Miguel Romance, Regino Criado, and SantiagoMoral” published by Hindawi Complexity Volume 2018, Article ID 5764370, 9 pages

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 importFeatureAgglomeration
import numpy
from numpy import inf, nan, dtype, mean
fromautoai_libs.sklearn.custom_scorer s import CustomScorers
import sklearn.ensemble
fromautoai_libs.cognito.transforms.tra nsform_utils import
TExtras, FC

from
autoai_libs.transformers.exportabl e import *
fromautoai_libs.utils.exportable_utilsimport *
from sklearn.pipeline importPipeline
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 = Falsefailed_retries = 0
while retries < 100 and
failed_retries < 10 and notsuccessful:
retries += 1failed_retries += 1try:
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 importlibmodule_name =
estr.split("'")[1]
module =
importlib.import_module(module_n ame)

globals().update({module_name: module})
```

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

as E:
except Exception
print(E)try:

import sysprint('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 asimport_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 = Nonecognito_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 = Nonecognito_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_a
nd_csv_filename'
target_label_name =_input_metadata['target_label_nam e']
learning_type =_input_metadata['learning_type'] optimization_metric
=_input_metadata['optimization_met ric']
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_metadataand 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 pdimport io
except Exception asimport_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 asimport_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 ascos_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 ascos_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 asdata_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 andproject_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 elsedf.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 = -1preprocessing_steps = []
for i, step inenumerate(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
scoresknown_values_list.clear() # known_values_list is filled in by the

preprocessing_pipeline if needed
preprocessing_pipeline.fit(df_X.val ues,
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.multiclassimport 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_selectionimport 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' inscorer._factory_args():
kwargs[prop] = True if scorer._sign == -1:
kwargs['greater_is_better'] =False
from sklearn.metrics importmake_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 datay_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 importget_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 orstr(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' inexception_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_selectionimport cross_validate
from sklearn.model_selectionimport 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 npnp.mean(cv_results['test_' +optimization_metric])
```

```
cv_results
```