SMART BRIDGE EDUCATIONAL SERVICES

PROJECT UNDER MACHINE LEARNING

HIGH POTENTIAL EMPLOYEE IN A CORPORATE

Submitted/done by

R.Saraswathi Meena
Assistant Professor
Dept. of Applied Mathematics and Computational Science
Thiagarajar College Of Engineering
Thirupparankuntram, Madurai, Tamilnadu.

1. INTRODUCTION

The employee is the key element of the organization. The success or failure of an organization depends on the employee. Most of the organizations or companies have a formal performance evaluation system in which employee job performance is graded on a regular basis, usually once or twice a year.

2. LITERATURE SURVEY

2.1 Objective

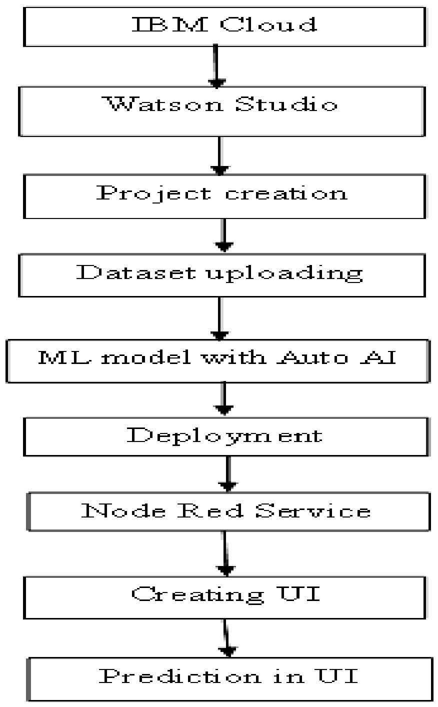
To predict if an employee is going to resign or not. Uncover the factors that lead to employee attrition and explore how each feature is co-related with attrition.

2.2 Proposed Solution

Using IBM Academic Initiative, IBM Cloud and its services like Auto AI and Node Red we have developed a machine learning model with 8 pipelines and selected one model with high accuracy of predicting Job Satisfaction and deployed the model inorder to make predictions. We have used Node Red service for creating a user interface for the user so that the user could enter the required inputs and can view the prediction in the user interface.

3. THEORITICAL ANALYSIS

3.1 Block diagram of Work Flow



Dataset:

We are using a dataset put up by IBM for our analysis. The dataset contain 35 features along with Attrition target variable. This is a fictional data set created by IBM data scientists. It can be downloaded from the following link.

Link- https://www.ibm.com/communities/analytics/watson-analytics-blog/hr-employee-attrition/ or https://www.kaggle.com/pavansubhasht/ibm-hr-analytics-attrition-dataset

Job Satisfaction

1 'Low'

2 'Medium'

3 'High'

4 'Very High'

3.2 IBM cloud:

There is around 6 steps to be followed before starting the project in Watson studio.

1. Creating IBM Academic Initiative Cloud Account
2. Creating Cloud object storage service after login of IBM cloud
3. Watson studio is a platform to work in Auto AI. So, Create a Watson studio platform for Machine Learning Service.
4. Download Watson Studio Desktop.

Implementation of Project:

- Creating a Project with the tittle “Prediction High Potential Employee” and mainly associate “Machine Learning –ve” service for the project with “Cloud Storage.”
- Uploading the dataset “.csv” and we can view it “Assets.”
- Creation of Auto AI Experiment for the column “Job Satisfaction” using Multiclass Regression with Accuracy and using associate “Machine Learning –ve” with 3 pipeline connection for the dataset.
- This AI model is ruined in the cloud Studio for 5-10 minutes.
- There are 6 algorithms used for the dataset and corresponding the output are sent for each model. Know, selecting the best algorithm for the column that is executed. This selected algorithm python code and model (with hybrid software is saved).
- The saved model is used for creating a space deployment and testing the model with some dataset in the space to find whether prediction is correct or wrong.

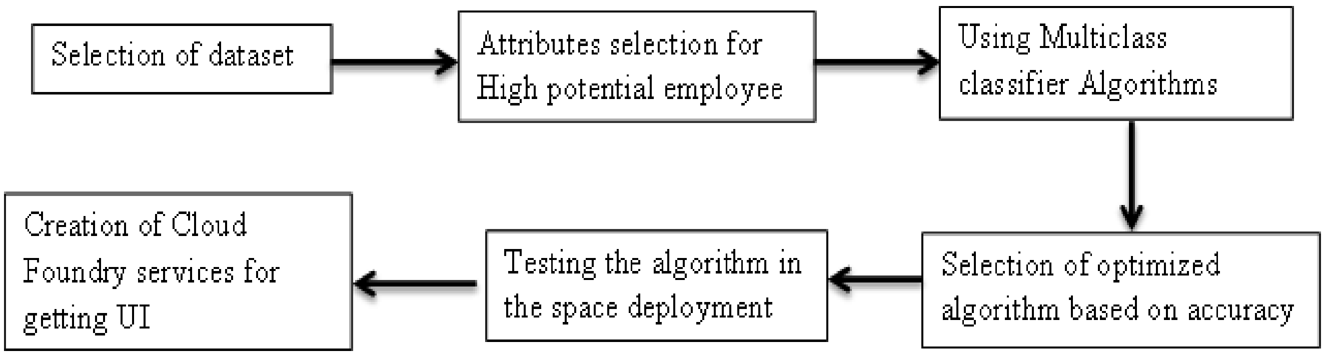
Deployment Space:

- After running the experiment we have to deploy our model inorder to make predictions.
- Save the best model you obtain.
- View the project and go to the deployments section and add deployment.

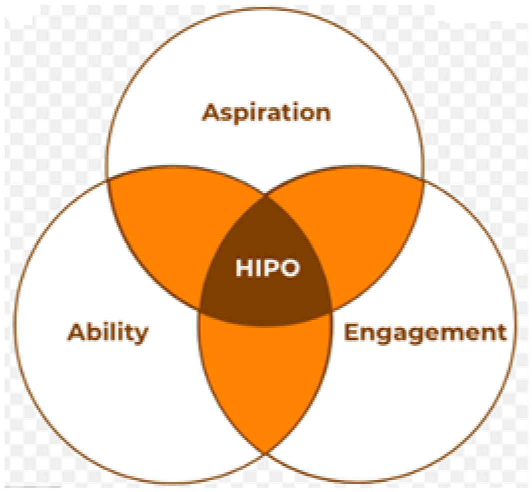
- Name the deployment and click create.
- In the test tab enter the input data and make some predictions.
- **Node UI**
  - A form node is included in the Flow.
  - Dashboard tab node, group node and form node is added and has been set up.
  - Form element “Job Satisfaction” of number type has been included.
  - Submit and cancel buttons are included by default.
- A Timestamp node has been included in the flow.
- A function node named “pre-token” is included in the flow
  - api key, headers and payload are set and the msg is returned.
- Both the form node and timestamp node are linked to the “Pre-token” node.
- Next a http request node is included in the flow.
  - Method is set to POST
  - URL is set to “https://eu-gb.ml.cloud.ibm.com”.
  - A parsed JSON object is returned
  - Link between http request and “Pre-token” is established.
- A function node “Prediction” is included in the flow
  - Token, header(Content type and Authorization) and payload (input data) is set and msg is returned.
  - Link between http request and “Prediction” is established.
- Next a http request node is included in the flow.
  - Method is set to POST
  - URL is set to the Machine learning deployment url.
  - A parsed JSON object is returned
  - Link between “Prediction” and http request is established.
- A function node “Parsing” is included in the flow.
  - Link between http request and “Parsing” is established.
- From “Parsing” a debug node and Edit text node is linked.
  - Edit text node is named as “Prediction” and it’s dashboard tab node, group node and text node has been added up.
- Now our Flow is ready and we can deploy it

- Deploy button is clicked and our flow has been deployed
- By clicking the link on the right side corner of the dashboard tab, will take us to the user interface that we have deployed.
- Enter the desired value to make prediction in the UI.

4. FLOW CHART


Selection of dataset
Attributes selection for High potential employee
Using Multiclass classifier Algorithms
Selection of optimized algorithm based on accuracy
Testing the algorithm in the space deployment
Creation of Cloud Foundry services for getting UI

We have to select a dataset for high potential employee prediction project. So, we have chosen best attribute to select a high potential employee. In that we are choosing 3 attributes (Job Satisfaction, Job Involvement, Salary). For that we are using multiclass classifier that has 7 algorithms for prediction. In that 7 algorithms for this dataset we have XGB classifier and Decision tree. Finally, the prediction value is accurate so, we created a UI model for the algorithm. Finally we can also compare the UI model prediction and algorithm's prediction value.



A High Potential employee is a proven high performer with three distinguishing attributes that allow them to rise and succeed in more senior, critical positions:

1. Aspiration- To rise to senior roles

2. Ability- To be effective in more responsible and senior roles

3.Engagement- To commit to the organization and remain in challenging roles

5. RESULTS

Our end-to-end solution, specially designed for high-potential identification of employees, helps organizations validate results through predictive modeling and correlation exercises. We, don’t believe in the one-size-fits-all approach, and thus our subject matter experts, researchers, and account executives handhold you at every step of the process to ensure a successful partnership.

6. ADVANTAGE AND DISADVANTAGE

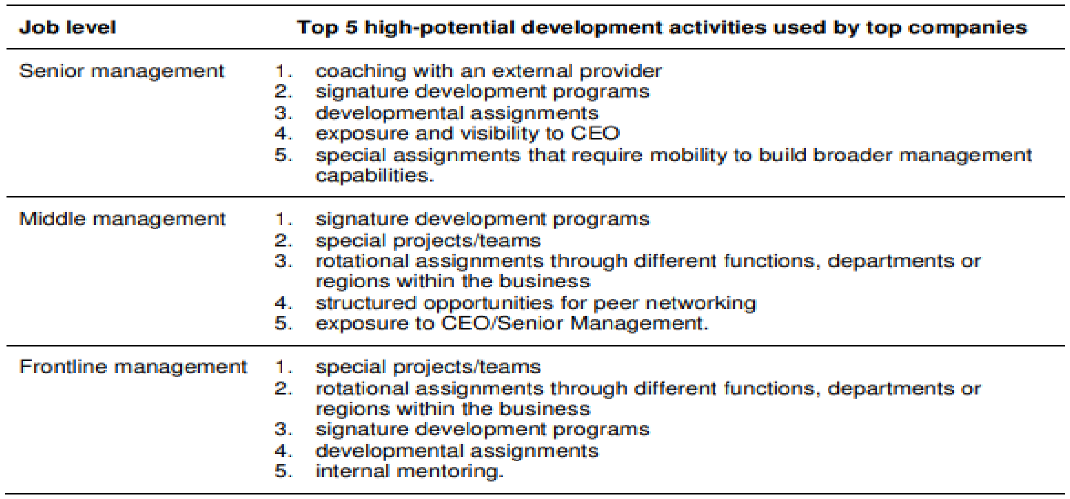
Advantage:

- ✓ Get high performance as the sole criteria for consideration off the agenda! Do it now
- ✓ Create a plan to review every candidate on the program and reassess each candidate
- ✓ Use the new analysis from your talent audit to create new understanding about those people on the program today, and who should actually be on the program.

Disadvantage:

- ✓ **They could start believing their own press.** Praising them as high potentials may result in overconfident, unbearable employees.
- ✓ **The grapevine:** If you're telling 10 percent of the workforce they're high potentials, what are you telling the other 90 percent? Word spreads quickly in any organization, so once employees know who's considered high potential, the rest of the team could quickly become disillusioned, unmotivated and unproductive.
- ✓ **The risk of creating a caste system.** Once staff know who is considered high potential, it can cause tension between "the haves" and "the have-nots."
- ✓ **They take on too much, too soon.** High potentials want to prove they can handle additional responsibilities, and often they fail because they've taken on projects and responsibilities that they're not equipped to handle.

7. APPLICATION



Job level	Top 5 high-potential development activities used by top companies
Senior management	1. coaching with an external provider 2. signature development programs 3. developmental assignments 4. exposure and visibility to CEO 5. special assignments that require mobility to build broader management capabilities.
Middle management	1. signature development programs 2. special projects/teams 3. rotational assignments through different functions, departments or regions within the business 4. structured opportunities for peer networking 5. exposure to CEO/Senior Management.
Frontline management	1. special projects/teams 2. rotational assignments through different functions, departments or regions within the business 3. signature development programs 4. developmental assignments 5. internal mentoring.

8. CONCLUSION

This project gave a brief learning in IBM studio and Auto AI especially without typing and finding error in the code. The predicted value is as same as the value in testing. There is lot of scope of learning different analytics oriented project.

9. BIBILOGRAPHY

1. https://cloud.ibm.com/
2. https://hbr.org/2010/06/are-you-a-high-potential
3. https://blog.shrm.org/workforce/should-you-tell-employees-theyre-high-potential

Appendix:

```
#!/usr/bin/env python
# coding: utf-8

#
########################################################################################
##
# #Licensed Materials - Property of IBM
# #(C) Copyright IBM Corp. 2020
# #US Government Users Restricted Rights - Use, duplication disclosure restricted
# #by GSA ADP Schedule Contract with IBM Corp.
#
#################################################################################
##
#
# The auto-generated notebooks are subject to the International License Agreement for Non-
Warranted Programs (or equivalent) and License Information document for Watson Studio Auto-
generated Notebook (License Terms), such agreements located in the link below.
# Specifically, the Source Components and Sample Materials clause included in the License
Information document for
# Watson Studio Auto-generated Notebook applies to the auto-generated notebooks.
# By downloading, copying, accessing, or otherwise using the materials, you agree to the License
Terms.
# http://www14.software.ibm.com/cgi-bin/weblap/lap.pl?li_formnum=L-AMCU-
BHU2B7&title=IBM%20Watson%20Studio%20Auto-generated%20Notebook%20V2.1

# ### IBM AutoAI Auto-Generated Notebook v1.14.1
#
# **Note:** Notebook code generated using AutoAI will execute successfully. If code is
modified or reordered,
# there is no guarantee it will successfully execute. This pipeline is optimized for the original
dataset.
# The pipeline may fail or produce sub-optimium results if used with different data. For different
data,
# please consider returning to AutoAI Experiments to generate a new pipeline. Please read our
documentation
# for more information:
# (Cloud Platform) https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/autoai-
notebook.html .
# (Cloud Pak For Data)
https://www.ibm.com/support/knowledgecenter/SSQNUZ_3.0.0/wsj/analyze-data/autoai-
notebook.html .
#
#
# Before modifying the pipeline or trying to re-fit the pipeline, consider:
# The notebook converts dataframes to numpy arrays before fitting the pipeline
# (a current restriction of the preprocessor pipeline). The known_values_list is passed by
reference
```

```
# and populated with categorical values during fit of the preprocessing pipeline. Delete its
members before re-fitting.

# ### Representing Pipeline_2
#

# ### 1. Set Up
# If lightgbm or xgboost installation fails, please follow:
#  - [lightgbm docs](https://lightgbm.readthedocs.io/en/latest/Installation-Guide.html)
#  - [xgboost docs](https://xgboost.readthedocs.io/en/latest/build.html)

# In[ ]:




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_scorers import CustomScorers
import sklearn.ensemble
from autoai_libs.cognito.transforms.transform_utils import TExtras, FC
from autoai_libs.transformers.exportable import *
from autoai_libs.utils.exportable_utils import *
from sklearn.pipeline import Pipeline
known_values_list=[]


# In[ ]:


# 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_name)
                        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_name)
               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

# In[ ]:



# metadata necessary to replicate AutoAI scores with the pipeline
_input_metadata = {'separator': ',', 'excel_sheet': 0, 'target_label_name': 'JobSatisfaction', 
'learning_type': 'classification', 'subsampling': None, 'pos_label': '1', 'pn': 'P2', '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': 'JobSatisfaction', 
'learning_type': 'classification', 'subsampling': None, 'pos_label': '1', 'pn': 'P2', '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_selector',
autoai_libs.transformers.exportable.NumpyColumnSelector(columns=[0])))
    preprocessor_features_categorical_steps.append(('cat_compress_strings',
autoai_libs.transformers.exportable.CompressStrings(activate_flag=True, compress_type='hash',
dtypes_list=['int_num'], missing_values_reference_list=['', '-', '?', nan], misslist_list=[[]])))
    preprocessor_features_categorical_steps.append(('cat_missing_replacer',
autoai_libs.transformers.exportable.NumpyReplaceMissingValues(filling_values=nan,
missing_values=[])))
    preprocessor_features_categorical_steps.append(('cat_unknown_replacer',
autoai_libs.transformers.exportable.NumpyReplaceUnknownValues(filling_values=nan,
filling_values_list=[nan], known_values_list=[[1, 2, 3, 4]], missing_values_reference_list=['', '-',
'?', nan])))
    preprocessor_features_categorical_steps.append(('boolean2float_transformer',
autoai_libs.transformers.exportable.boolean2float(activate_flag=True)))
    preprocessor_features_categorical_steps.append(('cat_imputer',
autoai_libs.transformers.exportable.CatImputer(activate_flag=True, missing_values=nan,
sklearn_version_family='23', strategy='most_frequent')))
    preprocessor_features_categorical_steps.append(('cat_encoder',
autoai_libs.transformers.exportable.CatEncoder(activate_flag=True, categories='auto',
dtype=numpy.float64, encoding='ordinal', handle_unknown='error',
sklearn_version_family='23')))
    preprocessor_features_categorical_steps.append(('float32_transformer',
autoai_libs.transformers.exportable.float32_transform(activate_flag=True)))
    # assembling preprocessor_features_categorical_ Pipeline
    preprocessor_features_categorical_pipeline =
sklearn.pipeline.Pipeline(steps=preprocessor_features_categorical_steps)
    preprocessor_features_transformer_list.append(('categorical',
preprocessor_features_categorical_pipeline))
    #
    # composing steps for preprocessor_features_numeric Pipeline
    #
    preprocessor_features_numeric__input_metadata = None
    preprocessor_features_numeric_steps = []
    preprocessor_features_numeric_steps.append(('num_column_selector',
autoai_libs.transformers.exportable.NumpyColumnSelector(columns=[1])))
    preprocessor_features_numeric_steps.append(('num_floatstr2float_transformer',
autoai_libs.transformers.exportable.FloatStr2Float(activate_flag=True, dtypes_list=['int_num'],
missing_values_reference_list=[])))
    preprocessor_features_numeric_steps.append(('num_missing_replacer',
autoai_libs.transformers.exportable.NumpyReplaceMissingValues(filling_values=nan,
missing_values=[])))
    preprocessor_features_numeric_steps.append(('num_imputer',
autoai_libs.transformers.exportable.NumImputer(activate_flag=True, missing_values=nan,
strategy='median')))
    preprocessor_features_numeric_steps.append(('num_scaler',
autoai_libs.transformers.exportable.OptStandardScaler(num_scaler_copy=None,
num_scaler_with_mean=None, num_scaler_with_std=None, use_scaler_flag=False)))
    preprocessor_features_numeric_steps.append(('float32_transformer',
autoai_libs.transformers.exportable.float32_transform(activate_flag=True)))
    # assembling preprocessor_features_numeric_ Pipeline
    preprocessor_features_numeric_pipeline =
sklearn.pipeline.Pipeline(steps=preprocessor_features_numeric_steps)
    preprocessor_features_transformer_list.append(('numeric',
preprocessor_features_numeric_pipeline))
    # assembling preprocessor_features_ FeatureUnion
    preprocessor_features_pipeline =
sklearn.pipeline.FeatureUnion(transformer_list=preprocessor_features_transformer_list)
    preprocessor_steps.append(('features', preprocessor_features_pipeline))
    preprocessor_steps.append(('permuter',
autoai_libs.transformers.exportable.NumpyPermuteArray(axis=0, permutation_indices=[0, 1])))
    # assembling preprocessor_ Pipeline
    preprocessor_pipeline = sklearn.pipeline.Pipeline(steps=preprocessor_steps)
    steps.append(('preprocessor', preprocessor_pipeline))
    steps.append(('estimator', xgboost.sklearn.XGBClassifier(base_score=0.5, booster='gbtree',
colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
gamma=5.7641840673449006e-08, learning_rate=0.020000072486408667, max_delta_step=0,
max_depth=2, min_child_weight=20, missing=None, n_estimators=500, n_jobs=1,
nthread=None, objective='multi:softprob', random_state=33, reg_alpha=0.9999999381174887,
reg_lambda=0.9999998988979512, scale_pos_weight=1, seed=None, silent=True,
subsample=0.010000049274900193, verbosity=0, tree_method='hist')))
    # assembling  Pipeline
    pipeline = sklearn.pipeline.Pipeline(steps=steps)
    return pipeline
```

```
pipeline = compose_pipeline()


# ### 3. Extract needed parameter values from AutoAI run metadata

# In[ ]:




# Metadata used in retrieving data and computing metrics.  Customize as necessary for your
environment.
#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

# In[ ]:




# @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.softlayer.net',
    'IBM_AUTH_ENDPOINT': 'https://iam.bluemix.net/oidc/token/',
    'APIKEY': 'UoU3_8O3s0al-AL_NedHgG6epu14bIC9IDXM87IwUf0y',
    'BUCKET': 'project-donotdelete-pr-pf5ryhyca5xwir',
    'FILE': 'ibm sarawathi1.csv',
    'SERVICE_NAME': 's3',
    'ASSET_ID': '1',
    }


# In[ ]:


#  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!')
                get_ipython().system('pip install ibm-cos-sdk')
                print('accessing data via Cloud Object Storage')
            try:
```

```
            client = ibm_boto3.client(service_name=credentials['SERVICE_NAME'],
                                 ibm_api_key_id=credentials['APIKEY'],
                                 ibm_auth_endpoint=credentials['IBM_AUTH_ENDPOINT'],
                                 config=Config(signature_version='oauth'),
                                 endpoint_url=credentials['ENDPOINT'])
          except Exception as cos_exception:
            print('unable to create client for cloud object storage')
          try:
            readable =
client.get_object(Bucket=credentials['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')
```

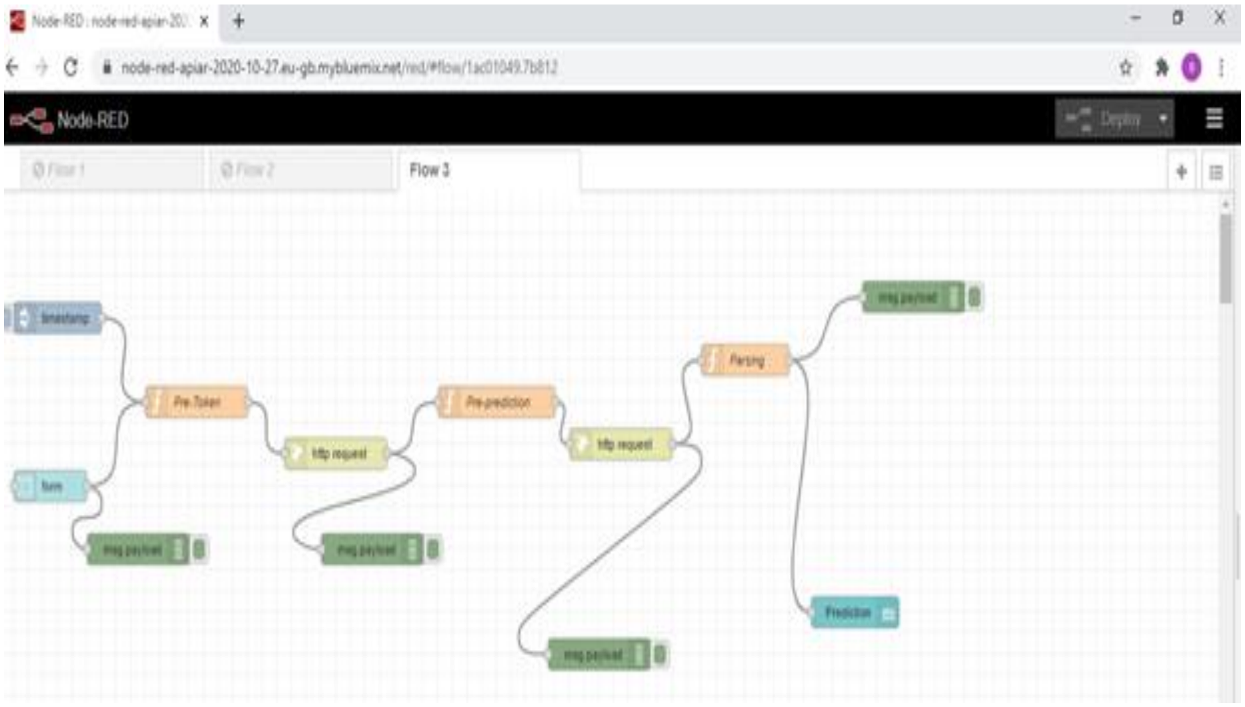
```
# Use data_provenance as filename if other access mechanisms are unsuccessful
if readable is None and type(data_provenance) is str:
    print('attempting to access local file using path and name ' + data_provenance)
    readable = data_provenance

# 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=_input_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['subsampling'],
random_state=_input_metadata['random_state']) if _input_metadata['subsampling'] <= 1.0 else
df.sample(n=_input_metadata['subsampling'], random_state=_input_metadata['random_state'])
else:
    print('Data cannot be loaded with credentials supplied, please provide DataFrame with training
data.')
```

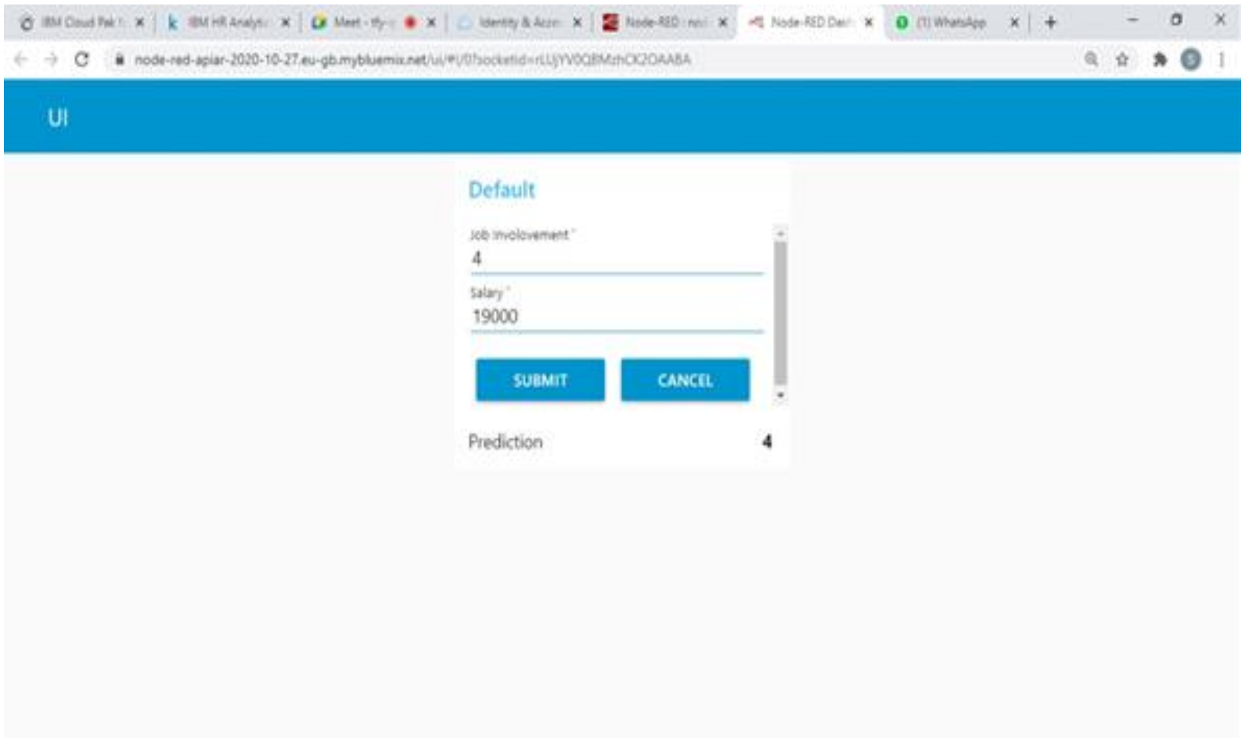
```
# ### 5. Preprocess Data

# In[ ]:


# 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)


# In[ ]:


# extract X and y
df_X = df.drop(columns=[target])
df_y = df[target]


# In[ ]:


# 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(pipeline.steps[preprocessor_index])
if preprocessor_index >= 0:
    preprocessing_pipeline = Pipeline(memory=pipeline.memory, steps=preprocessing_steps)
    pipeline = Pipeline(steps=pipeline.steps[preprocessor_index+1:])
```

```
# In[ ]:


# 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],pipeline.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)


# In[ ]:


# predict on the holdout data
y_pred = pipeline.predict(X_holdout)


# In[ ]:


# 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_label']
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)


# In[ ]:


# 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_folds, 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:scorer}, return_train_score=True)
import numpy as np
np.mean(cv_results['test_' + optimization_metric])


# In[ ]:


cv_results
```

OUTPUT- NODE RED Flow and User interface





Job Involvement *
4
Salary *
19000

Prediction 4