Internship Title: RSIP Career Basic ML 251

Project ID : SPS_PRO_291

Project Title : Resale value preditcion Using Watson Auto AI

-ANAGHA PHANIRAJ

Resale Value Prediction Using Watson Auto AI

INTRODUCTION

Creating models manually is difficult. It starts with finding candidate algorithms that best fit the specific case. Do we know all the appropriate algorithms.Then we have to prepare the data by converting any non-numeric fields to numerical values. Do we need to do additional feature engineering. How do we tune all the hyper-parameters for each chosen algorithms.For that we should use watson Auto AI tool.

The AutoAI tool in Watson Studio automatically analyzes your data and generates candidate model pipelines customized for your predictive modeling problem. It categorizes data as needed, does features engineering and optimizes hyperparameters To get to the best possible model.

Here in this project, we are using watson auto AI to predict the resale value of cars.
In order to build this project we have to follow certain steps and we have to mantain the flow of all the steps.

LITERATURE SURVEY

In many developed countries, it is common to lease a car rather than buying it outright. A lease is a binding contract between a buyer and a seller (or a third party – usually a bank, insurance firm or other financial institutions) in which the buyer must pay fixed instalments for a pre-defined number of months/years to the seller/financer. After the lease period is over, the buyer has the possibility to buy the car at its residual value, i.e. its expected resale value.

Predicting the resale value of a car is not a simple task. It is trite knowledge that the value of used cars depends on a number of factors. The most important ones are usually the age of the car, its make (and model), the origin of the car (the original country of the manufacturer), its mileage (the number of kilometers it has run) and its horsepower. Due to rising fuel prices, fuel economy is also of prime importance.

Unfortunately, in practice, most people do not know exactly how much fuel their car consumes for each km driven. Other factors such as the type of fuel it uses, the interior style, the braking system, acceleration, the volume of its cylinders (measured in cc), safety index, its size, number of doors, paint colour, weight of the car, consumer reviews, prestigious awards won by the car manufacturer, its physical state, whether it is a sports car, whether it has cruise control, whether it is automatic or manual transmission, whether it belonged to an individual or a company and other options such as air conditioner, sound system, power steering, cosmic wheels, GPS navigator all may influence the price as well. Some special factors which buyers attach importance in Mauritius is the local of previous owners, whether the car had been involved in serious accidents and whether it is a lady-driven car.

The look and feel of the car certainly contributes a lot to the price. As we can see, the price depends on a large number of factors. Unfortunately, information about all these factors are not always available and the buyer must make the decision to purchase at a certain price based on few factors only. In this work, we have considered only a small subset of the factors mentioned above.

THEORITICAL ANALYSIS:

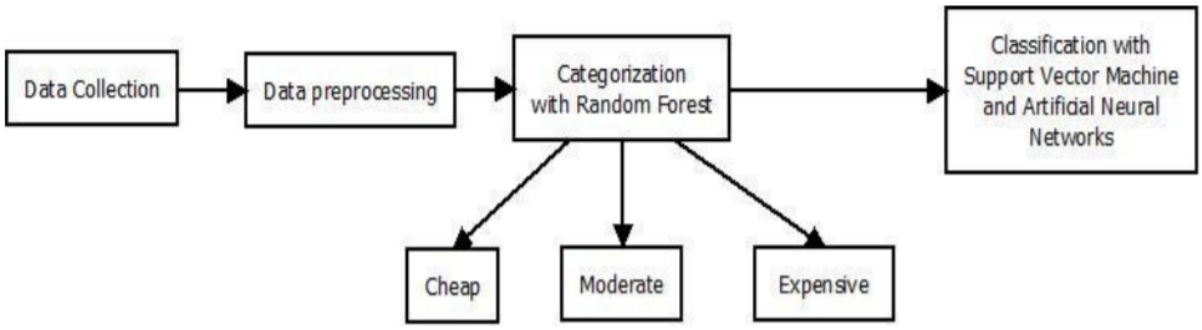

Figure 1. Block diagram of the overall classification process

Approach for car price prediction proposed in this report is composed of several steps, shown in Fig. 1.Data is collected from a local web portal for selling and buying cars , during winter season, as time interval itself has high impact on the price of the cars in some places. The following attributes were captured for each car: brand, model, car condition, fuel, year of manufacturing, power in kilowatts, transmission type, millage,

color, city, state, number of doors, four wheel drive (yes/no), damaged (yes/no), navigation (yes/no), leather seats (yes/no), alarm (yes/no), aluminum rims (yes/no), digital air condition (yes/no), parking sensors (yes/no), xenon lights (yes/no), remote unlock (yes/no), electric rear mirrors (yes/no), seat heat (yes/no), panorama roof (yes/no), cruise control (yes/no), abs (yes/no), esp (yes/no), asr (yes/no) and price expressed in BAM (Bosnian Mark). Since manual data collection is time consuming task, especially when there are numerous records to process, a “web scraper” as a part of this research is created to get this job done automatically and reduce the time for data gathering. Web scraping is well known technique to extract information from websites and save data into local file or database. Manual data extraction is time consuming and therefore web scrapers are used to do this job in a fraction of time. Web scrapers are programed for specific websites and can mimic regular users from website’s point of view.

EXPERIMENTAL INVESTIGATION:

Surprisingly, work on estimated the price of used cars is very recent but also very sparse. In her MSc thesis , Listiani showed that the regression mode build using support vector machines (SVM) can estimate the residual price of leased cars with higher accuracy than simple multiple regression or multivariate regression.
SVM is better able to deal with very high dimensional data (number of features used to predict the price) and can avoid both over-fitting and underfitting. In particular, she used a genetic algorithm to find the optimal parameters for SVM in less time.

The only drawback of this study is that the improvement of SVM regression over simple regression was not expressed in simple measures like mean deviation or variance. In another university thesis, Richardson working on the hypothesis that car manufacturers are more willing to produce vehicles which do not depreciate rapidly. In particular, by using a multiple regression analysis, he showed that hybrid cars (cars which use two different power sources to propel the car, i.e. they have both an internal combustion engine and an electric motor) are more able to keep their value than traditional vehicles.

This is likely due to more environmental concerns about the climate and because of its higher fuel efficiency. The importance of other factors like age, mileage, make and MPG (miles per gallon) were also considered in this study. He collected all his data from various websites.

FLOWCHART:

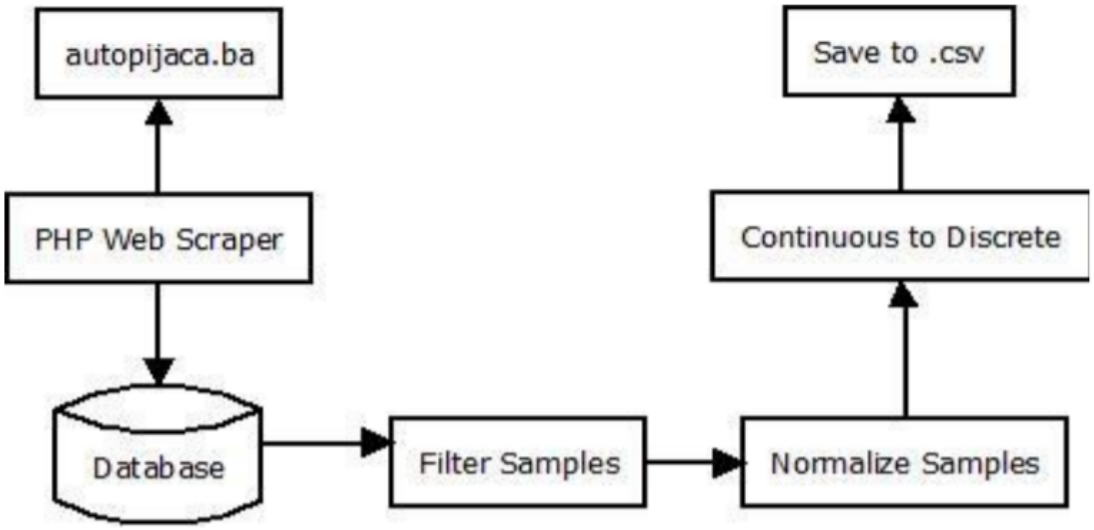

Figure 2. Data gathering and transformation workflow diagram

The whole dataset creation process is shown in the Fig. 2.After raw data has been collected and stored to local database, data preprocessing step was applied. Many of the attributes were sparse and they do not contain useful information for prediction. Hence, it is decided to remove them from the dataset. The attributes “state”, “city”, and “damaged” were completely removed.

brand	model	fuel	power in kilowatts	year of man	miles	leather	cruise control	price
volkswagen	golf2	Diesel	45-55	l7	l4	no	no	0-1500
volkswagen	golf2	Gasoline	0-45	l7	l4	no	no	0-1500
ford	escort	Gasoline	45-55	l7	l1	no	no	0-1500
ford	fiesta	Gasoline	55-65	l4	l2	no	no	0-1500
mercedes-benz	190	Gasoline	45-55	l7	l4	no	no	0-1500
volkswagen	jetta	Diesel	0-45	l7	l5	no	no	0-1500
ford	focus	Gasoline	55-65	l6	l4	no	no	0-1500
fiat	punto	Diesel	65-75	l5	l4	no	no	0-1500
volkswagen	golf2	Gasoline	65-75	l7	l4	no	no	0-1500

The collected raw data set contains 1105 samples. Since data is collected using web scraper, there are many samples that have only few attributes. In order to clean these samples, PHP script that is reading scraped data from database, perform cleaning and saves the cleaned samples into CSV file.

The CSV file is later used to load data into WEKA, software for building machine learning models. After cleanup process, the data set has been reduced to 797 samples. In particular, all brands that have less than 10 samples and where the price is higher than 60 000 BAM were removed due to the skew class problem.

The color of the cars was normalized into fixed set of 15 different colors. Continuous attributes such as “millage”, “year of manufacturing”, “power in kilowatts” and “price” are converted into categorical values using predefined cluster intervals. The millage is converted into five distinct categories, the year of manufacturing has been converted into seven categories and the power in kilowatts is converted into eleven categories.

The price attribute has been categorized into 15 distinct categories based on price range. These categories are shown in Table 2 and similar principle was applied to other attributes. This data transformation process converted regression prediction machine learning problem into classification problem.

Using NodeRed UI:

The **node red** dashboard is an add-on module that lets you create live dashboards. ... The Node_red dashboard files are stored in a folder called **node**-**red**-dashboard in the node_modules folder. The install adds a new category to the **node** palette and a collection of **UI** (**User Interface**) **nodes** or widgets

Node-**RED** is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways. It provides a browser-based editor that makes it easy to wire together flows using the wide range of **nodes** in the palette that can be deployed to its runtime in a single-click.

Node-**RED** is a programming tool for wiring together hardware devices, APIs and online services. Primarily, it is a visual tool designed for **the** Internet of Things, but it can also be used for other **applications** to very quickly assemble flows of various services. ... js **application**.

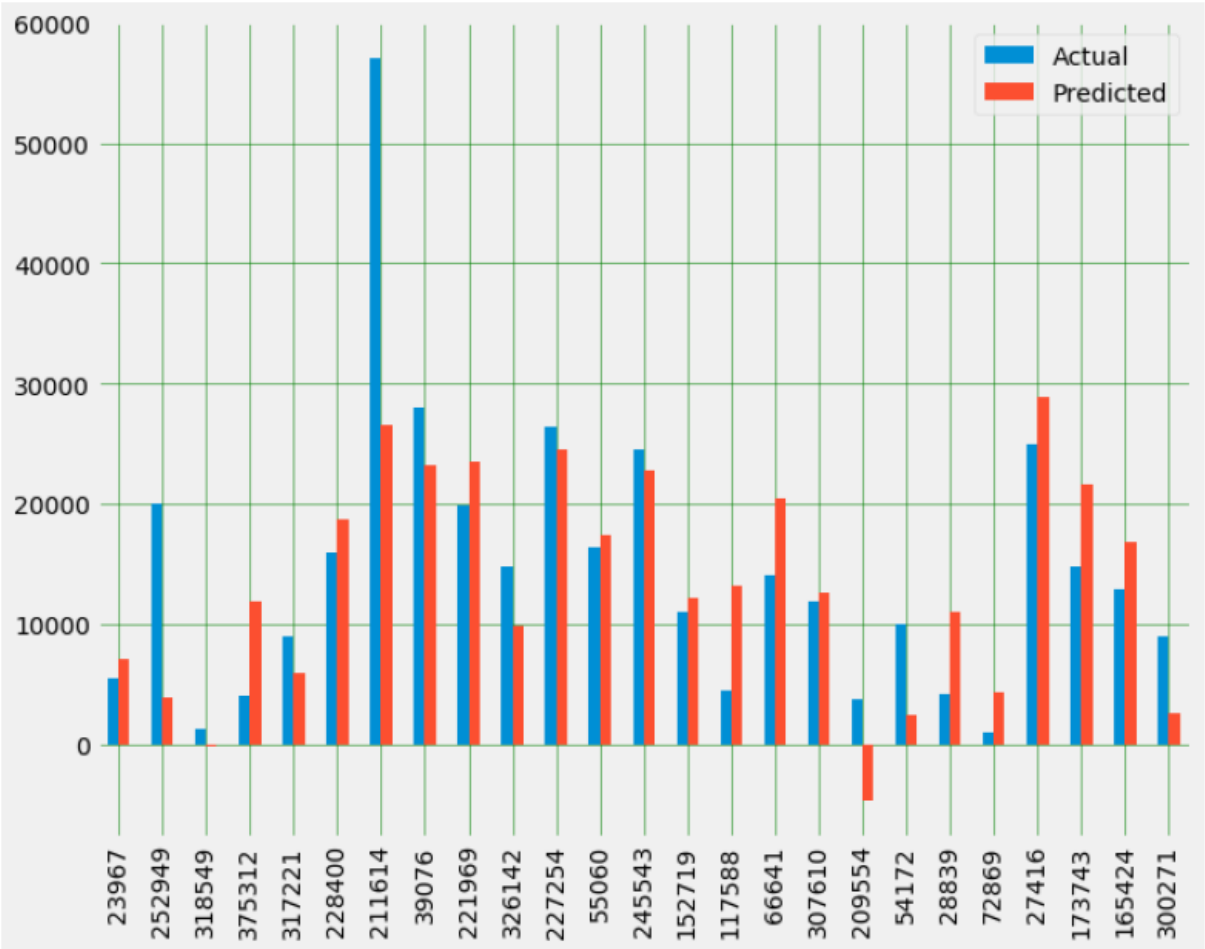

Figure 14: Performance of Linear Regression

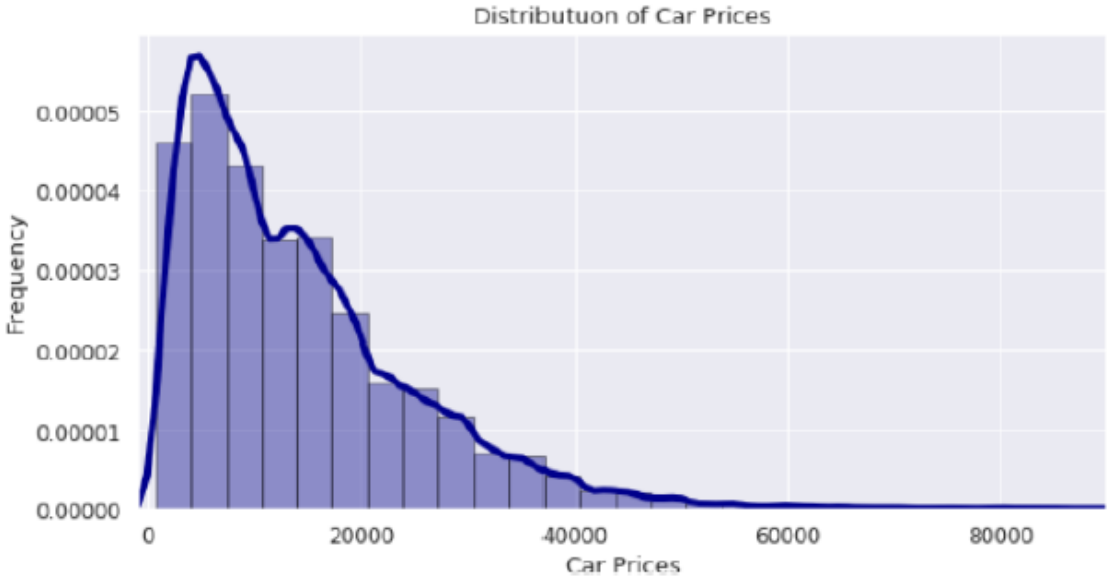

Figure 1: Density Plot of Car Prices

RESULT:

Thus, by using Gradient Boosting Regression Algorithm and training the model by entering the values of mileage, city, state, make, model and vehicle age, we can obtain the resale value.

ADVANTAGES:

Here are some advantages;

- Processes unstructured data
- Fills human limitations
- Acts as a decision support system, doesn't replace humans
- Improves performance + abilities by giving best available data
- Improve and transform customer service
- Handle enormous quantities of data
- Sustainable Competitive Advantage

DISADVANTAGES:

Here are some disadvantages;

- Only in English (Limits areas of use)
- Seen as disruptive technology
- Maintenance
- Doesn't process structured data directly
- Increasing rate of data, with limited resources

APPLICATIONS:

Here are five different types of applications where Watson is making an impact.

1. Healthcare

The medical field is the sector that is likely being impacted the most by Watson. For starters, Watson has taken residence at three of the top cancers hospitals in the US -- Memorial Sloan Kettering Cancer Center, University of Texas MD Anderson Cancer Center, and the Mayo Clinic -- where it helps with cancer research and patient care. In terms of cancer research, Watson is speeding up DNA analysis in cancer patients to help make their treatment more effective.

For physicians, Watson is helping with diagnoses. A dermatology app called schEMA allows doctors to input patient data and, using natural-language processing (NLP), helps identify potential symptoms and treatments. Additionally, Watson uses vision recognition to help doctors read scans such as X-rays and MRIs to better narrow the focus on a potential ailment.

It's used in veterinary medicine as well.

2. Finance

In the financial sector, Watson use is typically geared toward its question and answer capabilities. By not only answering questions, but also analyzing them as well, Watson can help give financial guidance and help manage financial risk.

In Australia, the company ANZ Global Wealth is focused on the latter. The company uses the Watson Engagement Advisor Tool, an NLP SaaS offering, to observe and field customer questions. Similarly, DBS bank in Singapore uses Watson to ensure the proper

advice and experience for customers of its wealth management business.

3. Legal

When it comes to the law, most of us likely have more questions than answers on any topic. However, startups such as ROSS Intelligence Inc. are using Watson to make it easier to get answers to your burning legal questions.

According to ROSS's website, users can ask questions in plain English and the app uses NLP to understand the questions and then sifts through the entirety of a database to return a cited answer with relevant legislation. ROSS also monitors potential changes to relevant laws and alerts you when changes occur.

Singapore's Inland Revenue Authority is another organization using Watson to help answer legal questions, deploying Watson to field questions about tax.

4. Retail

Modern retail experiences are all about personalization. Natural Selection is an app created by Sellpoints that uses Watson's NLP capabilities to present products to customers at the most appropriate point in the buying cycle. This can help reduce the overall number of clicks until conversion for an online retailer.

Watson is also being used in online travel purchases. Travel company WayBlazer has created a Discovery Engine that uses Watson to take in and analyze data to better link additional offers and customize preferences for individual consumers.

5. Fantasy Football

This may seem like an odd place for IBM Watson to make a mark, but the fact is that there's a ton of data that can be leveraged in fantasy sports. Edge Up Sports has a Watson-powered platform that analyzes NFL data to help fantasy football fans make better choices during the season. It analyzes data such as news about the teams and tweets from individual players. It specifically relies on Watson APIs and IBM Watson Personality Insights to streamline the research process for fantasy sports.

CONCLUSION:

In this report, four different techniques have been used to forecast the price of used cars . The mean error with linear regression was about Rs51, 000 while on the other side it was about Rs27, 000 for Nissan cars and about Rs45, 000 for Toyota cars. J48 and NaiveBayes accuracy dangled between 60-70% for different combinations of parameters. The main weakness of decision trees and naïve bayes is their inability to handle output classes with numeric values. Hence, the price attribute had to be classified into classes which contained a range of prices but this evidently introduced further grounds for inaccuracies. The main limitation of this study is the low number of records that have been used. As future work, we intend to collect more data and to use more advanced techniques like artificial neural networks, fuzzy logic and genetic algorithms to predict car prices.

BIBILOGRAPHY:

- International Journal of Information & Computation Technology. ISSN 0974-2239 Volume 4, Number 7 (2014), pp. 753-764 © International Research Publications House http://www. irphouse.com
- https://www.nicholasrenotte.com/how-to-predict-car-prices-using-watson-studio-machine-learning/

Appendix:

Source code

This is the auto ai notebook that is generated for our auto ai experiment

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 v1.12.2

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 from run: Pipeline_4 from run ceda1c2d-1c07-4036-ba62-9e15bfecbaff

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_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=[]
# 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

```
# metadata necessary to replicate AutoAI scores with the pipeline
_input_metadata = {'target_label_name': 'price', 'learning_type':
'regression', 'run_uid': 'ceda1c2d-1c07-4036-ba62-9e15bfecbaff', 'pn': 'P4',
'cv_num_folds': 3, 'holdout_fraction': 0.1, 'optimization_metric':
'neg_root_mean_squared_error', 'pos_label': None, '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 = {'target_label_name': 'price', 'learning_type':
'regression', 'run_uid': 'ceda1c2d-1c07-4036-ba62-9e15bfecbaff', 'pn': 'P4',
'cv_num_folds': 3, 'holdout_fraction': 0.1, 'optimization_metric':
'neg_root_mean_squared_error', 'pos_label': None, '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, 1, 2, 3,
4, 5, 7, 8, 9, 10, 11])))
    preprocessor_features_categorical_steps.append(('cat_compress_strings',
autoai_libs.transformers.exportable.CompressStrings(activate_flag=True,
compress_type='hash', dtypes_list=['float_int_num', 'char_str', 'char_str',
'char_str', 'char_str', 'char_str', 'char_str', 'char_str', 'char_str',
'char_str', 'char_str'], missing_values_reference_list=['', '-', '?', nan],
misslist_list=[[nan], [nan], [nan], [nan], [nan], [nan], [nan], [nan], [nan],
[nan], [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='20', 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='20')))
    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=[6])))

preprocessor_features_numeric_steps.append(('num_floatstr2float_transformer',
autoai_libs.transformers.exportable.FloatStr2Float(activate_flag=True,
dtypes_list=['float_int_num'], missing_values_reference_list=[nan])))
    preprocessor_features_numeric_steps.append(('num_missing_replacer',
autoai_libs.transformers.exportable.NumpyReplaceMissingValues(filling_values=na
n, missing_values=[nan])))
    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_transform
er_list)
    preprocessor_steps.append(('features', preprocessor_features_pipeline))
    preprocessor_steps.append(('permuter',
autoai_libs.transformers.exportable.NumpyPermuteArray(axis=0,
permutation_indices=[0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 6])))
    # assembling preprocessor_ Pipeline
    preprocessor_pipeline =
```

```
sklearn.pipeline.Pipeline(steps=preprocessor_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.transform_utils.TA1(fun=numpy.sin, name='sin',
datatypes=['float'],
feat_constraints=[autoai_libs.utils.fc_methods.is_not_categorical],
tgraph=None, apply_all=True, col_names=['year', 'manufacturer', 'model',
'condition', 'cylinders', 'fuel', 'odometer', 'transmission', 'drive', 'size',
'type', 'paint_color'], 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.transform_utils.FS1(cols_ids_must_keep=range(0,
12), additional_col_count_to_keep=12, ptype='regression')))
    cognito_steps.append(('2',
autoai_libs.cognito.transforms.transform_utils.TA1(fun=numpy.square,
name='square', datatypes=['numeric'],
feat_constraints=[autoai_libs.utils.fc_methods.is_not_categorical],
tgraph=None, apply_all=True, col_names=['year', 'manufacturer', 'model',
'condition', 'cylinders', 'fuel', 'odometer', 'transmission', 'drive', 'size',
'type', 'paint_color', 'sin(year)', 'sin(manufacturer)', 'sin(model)',
'sin(odometer)'], 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')], col_as_json_objects=None)))
    cognito_steps.append(('3',
autoai_libs.cognito.transforms.transform_utils.FS1(cols_ids_must_keep=range(0,
12), additional_col_count_to_keep=12, ptype='regression')))
    cognito_steps.append(('4',
```

```
autoai_libs.cognito.transforms.transform_utils.TGen(fun=autoai_libs.cognito.tr
ansforms.transform_extras.NXOR, name='nxor', arg_count=2,
datatypes_list=[['numeric'], ['numeric']],
feat_constraints_list=[[autoai_libs.utils.fc_methods.is_not_categorical],
[autoai_libs.utils.fc_methods.is_not_categorical]], tgraph=None,
apply_all=True, col_names=['year', 'manufacturer', 'model', 'condition',
'cylinders', 'fuel', 'odometer', 'transmission', 'drive', 'size', 'type',
'paint_color', 'sin(year)', 'sin(manufacturer)', 'sin(model)',
'sin(odometer)', 'square(year)', 'square(manufacturer)', 'square(model)',
'square(odometer)', 'square(sin(year))', 'square(sin(manufacturer))',
'square(sin(model))', 'square(sin(odometer))'], 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(('5',
autoai_libs.cognito.transforms.transform_utils.FS1(cols_ids_must_keep=range(0,
12), additional_col_count_to_keep=12, ptype='regression')))
    # assembling cognito_ Pipeline
    cognito_pipeline = sklearn.pipeline.Pipeline(steps=cognito_steps)
    steps.append(('cognito', cognito_pipeline))
    steps.append(('estimator', xgboost.sklearn.XGBRegressor(base_score=0.5,
booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
gamma=0.9866156450335044, importance_type='gain', learning_rate=0.02,
max_delta_step=0, max_depth=4, min_child_weight=20, missing=None,
n_estimators=1421, n_jobs=2, nthread=None, objective='reg:linear',
random_state=33, reg_alpha=0.610707914564802, reg_lambda=0.10803689309982949,
scale_pos_weight=1, seed=None, silent=True, subsample=0.9925140154963886,
verbosity=0)))
    # assembling  Pipeline
    pipeline = sklearn.pipeline.Pipeline(steps=steps)
    return pipeline
pipeline = compose_pipeline()
```

3. Extract needed parameter values from AutoAI run metadata

```
# 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

```
# @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': '2jTpq2xw_ozmTaWsjWQoTgxfeRdOnzflk1RVpvZ5omJN',
    'BUCKET':
'resalevaluepredictionusingwatsona-donotdelete-pr-3mkk84ytjdgxd1',
    'FILE': 'cars_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
        import os
    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:
                cos_client =
ibm_boto3.resource(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:

cos_client.meta.client.download_file(Bucket=credentials['BUCKET'],
Filename=credentials['FILE'], Key=credentials['FILE'])
            except Exception as cos_access_exception:
                print('unable to access data object in cloud object storage
with credentials supplied')
            try:
                for encoding in csv_encodings:
                    df = pd.read_csv(credentials['FILE'], encoding = encoding,
sep = None, engine = 'python')
                    os.remove(credentials['FILE'])
                    print('Data loaded from cloud object storage with encoding
' + encoding)
                    break
            except Exception as cos_object_read_exception:
                print('unable to access data object from cos object with
encoding ' + encoding)
        elif credentials['SERVICE_NAME'] == 'fs':
            print('accessing data via File System')
            try:
                df = pd.read_csv(credentials['FILE'], sep = None, engine =
'python')
            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:
            df = pd.read_csv(readable, encoding=encoding, sep = None, engine =
'python')
            print('successfully loaded dataframe using encoding = ' +
str(encoding))
            break
        except Exception as exception_csv:
            print('unable to read csv using encoding ' + str(encoding))
            print('handled error was ' + str(exception_csv))
    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')
```

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(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:])
# 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

```
# 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

```
#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

```
# 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

```
# 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_label']
try:
    score = scorer(pipeline, X_holdout, y_holdout)
except Exception as e1:
    if 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_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])
```