Lesson 06

CSC357 Advanced Topics—Machine Learning

20 January 2020

Create a lesson with Jupyter. Work with your team on this exercise.

- Experiment with this code. You might...
  - Look on the Internet to find the documentation for the NumPy and scikit-learn classes and functions that this code uses.
  - Try varying the values of some parameters.
  - Add code that plots some of the data, prints the contents of data structures with labels, or adds some descriptive statistics.
- Annotate this program with comments that describe what the programmer has accomplished with each block of code.
- Create a Jupyter notebook with some or all of this code and your comments.

```
import numpy as np
import pandas as pd
from sklearn.base import BaseEstimator
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import MinMaxScaler

def make_row(mean, std, columns):
    return list(map((lambda x: int(x * 100) / 100),
                    np.random.normal(2.0, 3.0, columns)))

def make_table( mean, std, rows, columns ):
    return [make_row(mean, std, columns) for i in range(rows)]
```

```
def make_column_labels( columns ):
    return [ chr(65 + i) for i in range(columns)]

def make_dataset( mean, std, rows, columns ):
    table = make_table( mean, std, rows, columns )
    labels = make_column_labels( columns )
    return pd.DataFrame( table, columns=labels )

ds = make_dataset( 2.0, 3.0, 8, 4 )

column_labels = ds.columns.values

print( ”\n” )
print( column_labels )

print( ”\n” )
print( ds )

#print( ”\n” )
#print( ds.head(2) )

#print( ”\n” )
#print( ds.info() )

#print( ”\n” )
#print( ds.describe() )

d_series = ds[”D”]
print( ”\n” )
print( d_series )

ds_dropped = ds.drop( ”D”, axis=1)

print( ”\n” )
print( ds_dropped )

ds_dropped[”D”] = d_series

print( ”\n” )
print( ds_dropped )

f = lambda offset: FunctionTransformer(
        (lambda x: x + offset), validate=True)

#print( ”\n” )
```

```
#print( f.transform(ds) )

column_transformer = ColumnTransformer( [("a", f(1), ["A"]),
                        ("b", f(2), ["B"]),
                        ("c", f(1), ["C"]),
                        ("d", f(2), ["D"])] )

ds_transformed = pd.DataFrame( column_transformer.fit_transform(ds),
                               columns=column_labels)

print( "\n")
print( ds_transformed )

min_max_scaler = MinMaxScaler()

ds_transformed = pd.DataFrame( min_max_scaler.fit_transform( ds ),
                               columns=column_labels )

print( "\n" )
print( ds_transformed )

pipeline = make_pipeline( column_transformer, min_max_scaler )

ds_transformed = pd.DataFrame( pipeline.fit_transform( ds ),
                               columns=column_labels)

print( "\n" )
print( ds_transformed )

missing_values = pd.DataFrame( [[1, np.nan], [np.nan, 1]],
                               index=["ROW_0", "ROW_1"],
                               columns=["COL_0", "COL_1"] )

print( "\n" )
print( missing_values )

row_labels = missing_values.index.values
column_labels = missing_values.columns.values

imputer = SimpleImputer( strategy="constant", fill_value=99 )
missing_values_transformed = pd.DataFrame(
        imputer.fit_transform( missing_values),
                               index=row_labels,
                               columns=column_labels)

print( "\n" )
print( missing_values_transformed )

x_min = 2.0
x_max = 4.0
number_of_points = 12
x_values = np.linspace( x_min, x_max, number_of_points )

slope = 2.0
intercept = 1.0
standard_deviation = 0.5
y_sample = lambda i: (slope * i + intercept +
                      np.random.normal( 0.0, standard_deviation ))
y_values = np.array( [y_sample(x) for x in x_values])

values = pd.DataFrame( {”X”: x_values, ”Y”: y_values} )

print( ”\n” )
print( values )

train_set, test_set = train_test_split( values, test_size=0.25)

print( ”\n” )
print( train_set )

print( ”\n” )
print( test_set )

class LinearEstimator( BaseEstimator ):
    def __init__(self, m=1.0, b=0.0):
        self.m = m
        self.b = b

    def fit(self, x_values, y_values ):
        return self

    def predict(self, x_values ):
        return [self.m * x + self.b for x in x_values]


estimator = LinearEstimator( m = 2.0, b = 1.0 )

estimator.fit( values[”X”], values[”Y”])

predicted_y_values = estimator.predict( train_set[”X”] )

s = train_set.copy()
```

```
s["PREDICTED_Y"] = predicted_y_values

print( "\n" )
print( s )

parameters = { 'm': [1,2], 'b': [0,1]}

grid_search = GridSearchCV( estimator, parameters,
                            scoring="neg_mean_squared_error", cv=2, iid=False )

grid_search.fit( train_set["X"], train_set["Y"])

print( "\n" )
#print( grid_search.cv_results_)
print( grid_search.cv_results_["params"])
print( grid_search.cv_results_["mean_test_score"])
print( grid_search.cv_results_["rank_test_score"])
```