MLP Week 2 Solve With Us

- API of the functions mentioned in the lectures
- Gist of working procedure of APIs, like encoders and transformations
-

Task 1

1. Import `SimpleImputer` from SkLearn library.
2. Take a matrix [[753, 1622, 3193], [np.nan, np.nan, 1966], [1200, 5, np.nan],[981, np.nan, 9211]]
3. Impute the missing values in the matrix using `SimpleImputer` with
    - Mean
    - Median
4. Print the imputed matrix using `fit_transform`. Do you see any change in results between imputing with mean and imputing with median?

Solution

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns #importing the usual libraries
```

```
from sklearn.impute import SimpleImputer
```

```
import numpy as np
#from sklearn.impute import SimpleImputer
imp_mean = SimpleImputer(strategy='mean') #imputed with mean
X=imp_mean.fit_transform([[753, 1622, 3193], [np.nan, np.nan, 1966], [1200, 5, np.nan],[98
print(X)
```

```
[[7.530e+02 1.622e+03 3.193e+03]
 [9.780e+02 8.135e+02 1.966e+03]
 [1.200e+03 5.000e+00 4.790e+03]
 [9.810e+02 8.135e+02 9.211e+03]]
```

```
import numpy as np
#from sklearn.impute import SimpleImputer
imp_mean = SimpleImputer(missing_values=np.nan, strategy='median') #imputed with median
```

```
X=imp_mean.fit_transform([[753, 1622, 3193], [np.nan, np.nan, 1966], [1200, 5, np.nan],[98
print(X)
```

```
[[7.530e+02 1.622e+03 3.193e+03]
 [9.810e+02 8.135e+02 1.966e+03]
 [1.200e+03 5.000e+00 3.193e+03]
 [9.810e+02 8.135e+02 9.211e+03]]
```

Task 2

1. Import `FunctionTransformer` from the SkLearn library.
2. Apply log base 10 to the elements of the following array : `[[0, 1], [2, 3],[10,100]]` and print it

```
from sklearn.preprocessing import FunctionTransformer
transformer = FunctionTransformer(np.log10, validate=False)
X = np.array([[0, 1], [2, 3],[10,100]])
print(X)
print(transformer.fit_transform(X))
```

```
[[  0   1]
 [  2   3]
 [ 10 100]]
[[      -inf 0.        ]
 [0.30103    0.47712125]
 [1.         2.        ]]
/usr/local/lib/python3.7/dist-packages/sklearn/preprocessing/_function_transformer.p
  return func(X, **(kw_args if kw_args else {}))
```

transformation cases create pipeline

Task 3

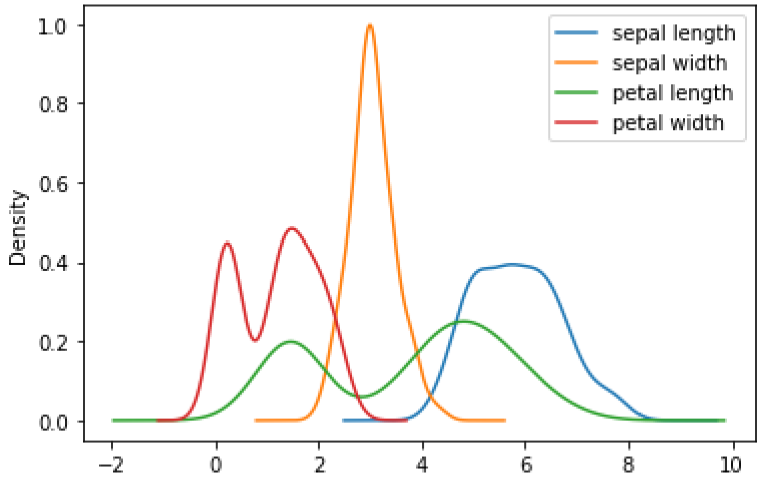
1. Read the CSV file from `https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data`, define the column headers as `'sepal length', 'sepal width', 'petal length', 'petal width','label'` and generate

- a Kernel Densiy Estimation (KDE) plot.
- a boxplot
- a violin plot
- generate the correlation plot between each pair of features.

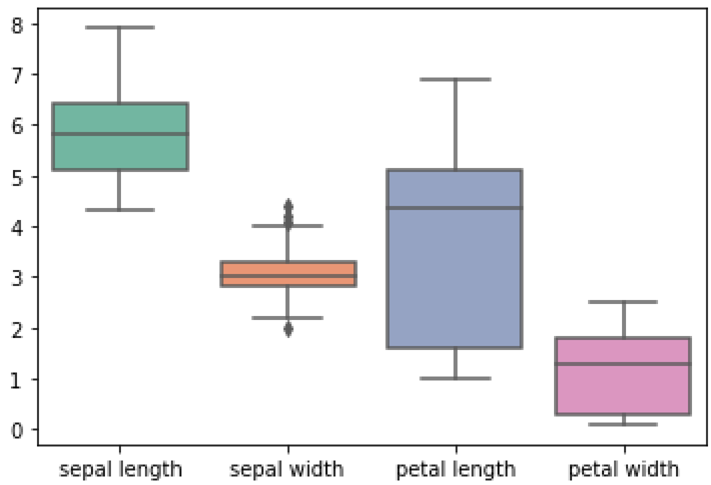
2. Generate a new feature matrix consisting of all polynomial combinations of the features with degree 2 (`For example, if an input sample is two dimensional and of the form [a,b] , the degree-2 polynomial features are` $[1, a, b, a^2, ab, b^2]$) and print the shapes of the feature matrix before and after the polynomial transformation.

Solution

```
cols = ['sepal length', 'sepal width', 'petal length', 'petal width','label']
a=pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data',h
#a.info() #1.sepal length in cm, 2. sepal width in cm, 3. petal length in cm, 4. petal wic
ax=a.plot.kde() #the KDE plot
```




```
ay = sns.boxplot(data=a, orient="v", palette="Set2") #the box plot
```




```
ay = sns.violinplot(data=a, orient="v", palette="Set3", scale="width") #this is the violir
```

```
a.columns
```

    Index(['sepal length', 'sepal width', 'petal length', 'petal width', 'label'],
    dtype='object')

```
from sklearn.preprocessing import PolynomialFeatures
print('Number of features before transformation = ', a.shape)
#b = a.drop(['Iris-setosa'],axis=1 )
# Let us fit a polynomial of degree 2 to Iris_data
poly = PolynomialFeatures(degree=2)
poly_iris_data = poly.fit_transform(a[a.columns[:4]])
print('Number of features after transformation = ', poly_iris_data.shape)
```

    Number of features before transformation = (150, 5)
    Number of features after transformation = (150, 15)

$$1, a, b, c, d, a^2, b^2, c^2, d^2, ab, bc, cd, ac, bd, ad$$

```
#b=pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/wir
a.label.unique()
```

    array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)

```
a
```

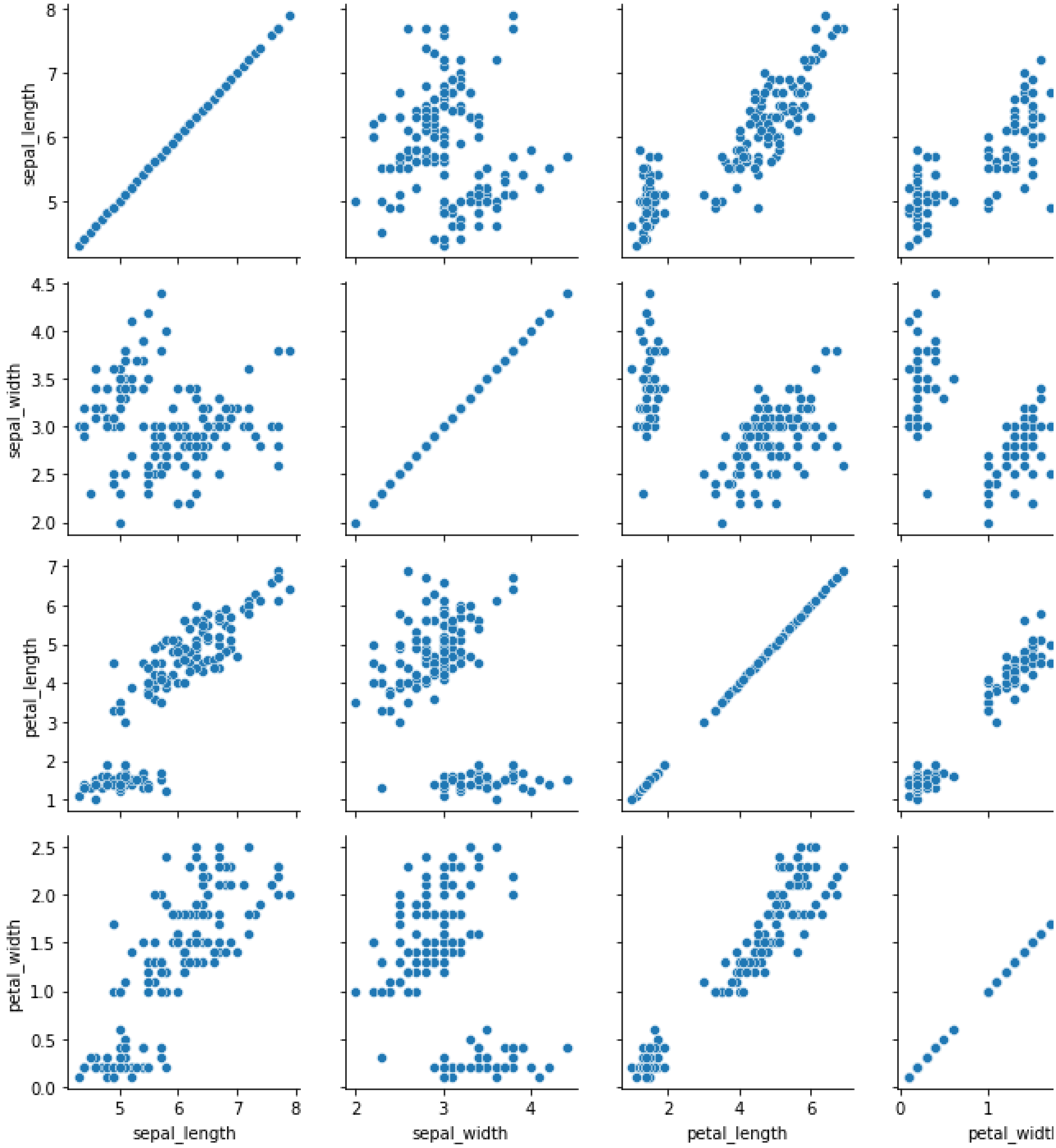
	sepal length	sepal width	petal length	petal width	label
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

```
iris = sns.load_dataset("iris")
g = sns.PairGrid(iris)
```

```
g.map(sns.scatterplot)
```

```
<seaborn.axisgrid.PairGrid at 0x7fcc9a973650>
```



Task 4

1. Import `OneHotEncoder` class from `sklearn.preprocessing` module.
2. Print shapes of the matrix before and after one-hot-encoding.
3. Print 45 to 55th row of the matrix after one-hot-encoding

Solution

```
from sklearn.preprocessing import OrdinalEncoder
```

```
from sklearn.preprocessing import OneHotEncoder


onehotencoder = OneHotEncoder(categories='auto')
print('Shape of the matrix before encoding', a.label.shape)
iris_labels = onehotencoder.fit_transform(a.label.values.reshape(-1,1))
print('Shape of the matrix after encoding', iris_labels.shape)
print (" labels:")
print(iris_labels.toarray()[45:55])
```

```
Shape of the matrix before encoding (150,)
Shape of the matrix after encoding (150, 3)
 labels:
[[1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]]
```

Task 5

1. Import the California Housing dataset and `SelectPercentile, mutual_info_regression`.
2. Select features according to 10 percentile of the highest scores
3. Print shapes of the feature matrix before and after feature selection.

```
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectPercentile, mutual_info_regression


from sklearn.feature_selection import SelectPercentile

# dowload data
X_california, y_california = fetch_california_housing(return_X_y=True)
# select a subset of data
X, y = X_california[:1000, :], y_california[:1000]
sp = SelectPercentile(mutual_info_regression, percentile = 10)
print(f'Shape of of feature matrix before feature selection:{X.shape}')
X_new = sp.fit_transform(X,y)
print(f'Shape of of feature matrix after feature selection:{X_new.shape}')
```

```
Shape of of feature matrix before feature selection:(1000, 8)
Shape of of feature matrix after feature selection:(1000, 1)
```

Task 6

- Generate a numeric pipeline using

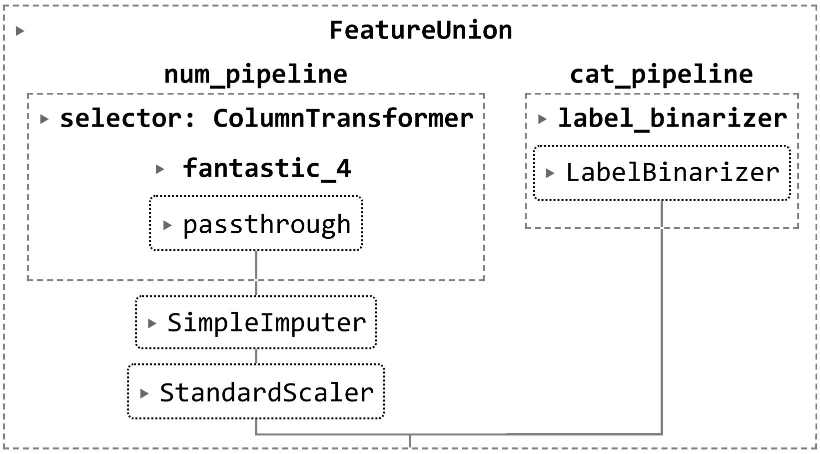
1. a `columnTransformer` named `fantastic_4` with a block `passthrough` inside it.
2. a `SimpleImputer` using `mean` strategy and
3. a StandardScaler operator named `std_scaler`.

- Generate a categorical pipeline applying `LabelBinarizer` on the 5th feature.
- Combine these two pipelines using `FeatureUnion` and display the full pipeline diagram.

```
from sklearn.preprocessing import StandardScaler, LabelBinarizer
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer


num_pipeline = Pipeline([('selector',ColumnTransformer([('fantastic_4',
                                                         'passthrough',
                                                         slice(0,4))])),#selector is the c
                         ('imputer', SimpleImputer(strategy="median")),
                         ('std_scaler', StandardScaler()),
                        ])
cat_pipeline = ColumnTransformer([('label_binarizer', LabelBinarizer(),[4])])#on the 5th f
full_pipeline = FeatureUnion(transformer_list=
                             [("num_pipeline", num_pipeline),("cat_pipeline", cat_pipeline


from sklearn import set_config
set_config(display='diagram')
# displays HTML representation in a jupyter context
full_pipeline
```



Task 7

Make a pipeline containing `SimpleImputer`, `PCA` and `LinearRegression` estimator steps.

- Print the length of steps
- In PCA step, set no. of principal components to 4.
- Access the individual steps of the pipeline.
- Print no of components of PCA step via pipeline object.

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
#estimators = [
#  ('simpleImputer', SimpleImputer()),
#  ('standardScaler', StandardScaler()),
#]
#pipe = Pipeline(steps=estimators)


#from sklearn.pipeline import make_pipeline
#pipe = make_pipeline(SimpleImputer(),StandardScaler())


from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.decomposition import PCA
estimators = [('simpleImputer', SimpleImputer()),('pca', PCA(n_components=4)),('regressor'
pipe = Pipeline(steps=estimators)


print(len(pipe.steps)) #print number of steps in this pipeline
```

```
3
```

```
pipe.steps[0]
```

```
('simpleImputer', SimpleImputer())
```

```
print(pipe.named_steps.pca.n_components)
```

```
4
```

Task 8

1. Import the California housing dataset, Recursive Feature Elimination (RFE) from appropriate modules.
2. Perform wrapper based feature selection using RFE method
3. Print the support attribute and rankings of features.

```
from sklearn.datasets import fetch_california_housing
```

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

X,y = fetch_california_housing(return_X_y = True)

estimator = LinearRegression()
selector = RFE(estimator, n_features_to_select=5, step = 1)
selector = selector.fit(X,y)
print(selector.support_)
print(selector.ranking_)
```

```
[1 2 1 1 4 3 1 1]
```

Task 9

1. Import `SequentialFeatureSelector, KNeighborsClassifier` with 3 features to select and 3 neighbors and load the iris dataset.
2. Perform wrapper based feature selection using SFS method
3. Find the shape of the transformed matrix.

```
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris

X,y = load_iris(return_X_y = True)
knn = KNeighborsClassifier(n_neighbors = 3)
sfs = SequentialFeatureSelector(knn, n_features_to_select = 3)
sfs.fit(X,y)
SequentialFeatureSelector(estimator=KNeighborsClassifier(n_neighbors=3),
                          n_features_to_select=3)
#sfs.get_support()
sfs.transform(X).shape
```

```
(150, 3)
```