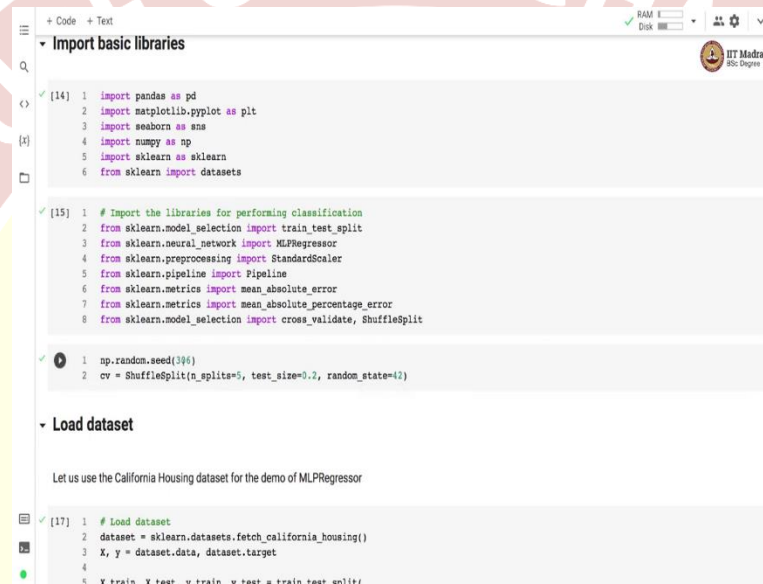


**IIT Madras**  
ONLINE DEGREE

**Machine Learning Practice**  
**Bachelor of Science**  
**Indian Institute of Technology, Madras**  
**Multilayer perceptron regressor on California Housing Dataset**

Namaste! Welcome to the next video of Machine Learning Practice Course. In this video, we will implement MLPRegressor on California housing dataset.

(Refer Time Slide: 00:25)



```
[14]: 1 import pandas as pd
      2 import matplotlib.pyplot as plt
      3 import seaborn as sns
      4 import numpy as np
      5 import sklearn as sklearn
      6 from sklearn import datasets

[15]: 1 # Import the libraries for performing classification
      2 from sklearn.model_selection import train_test_split
      3 from sklearn.neural_network import MLPRegressor
      4 from sklearn.preprocessing import StandardScaler
      5 from sklearn.pipeline import Pipeline
      6 from sklearn.metrics import mean_absolute_error
      7 from sklearn.metrics import mean_absolute_percentage_error
      8 from sklearn.model_selection import cross_validate, ShuffleSplit

1 np.random.seed(396)
2 cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=42)

- Load dataset

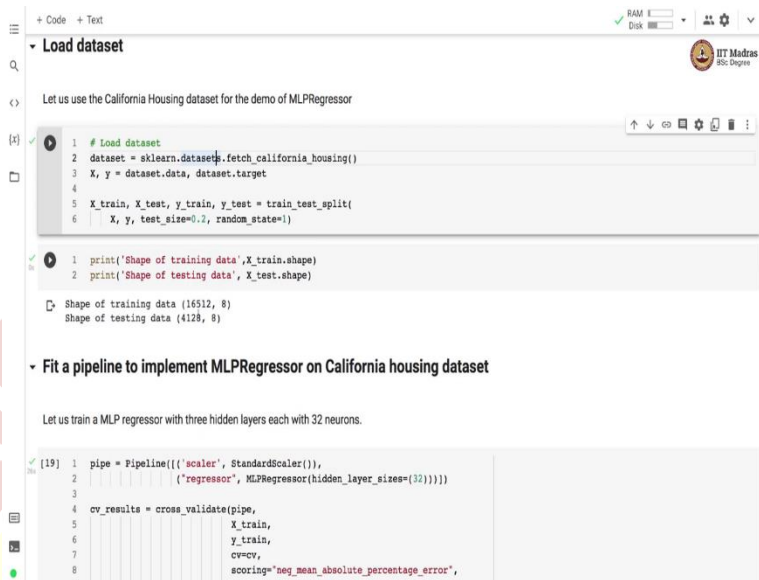
Let us use the California Housing dataset for the demo of MLPRegressor

[17]: 1 # Load dataset
      2 dataset = sklearn.datasets.fetch_california_housing()
      3 X, y = dataset.data, dataset.target
      4
      5 X_train, X_test, y_train, y_test = train_test_split(X,
```

We begin by importing basic libraries like pandas and Numpy, and then matplotlib and Seaborn for plotting. We import datasets from sklearn. The training test split is carried out by `train_test_split`. The regression is carried out by `MLPRegressor`, which is implemented in `sklearn.neural_network` module will perform feature processing like scaling using `StandardScaler` in `sklearn.preprocessing` module and the feature preprocessing and the regressor together are implemented in pipeline object, which is imported from `sklearn.pipeline` module.

We evaluate the performance of classifier using `mean_absolute_error` and `mean_absolute_percentage_error`. And both these metrics are imported from `sklearn.metrics` module. We use `cross_validate` for training the classifier and `ShuffleSplit` for model selection. We begin by defining a random seed. We use `ShuffleSplit` cross-validation with number of splits = 5 and we set a size 20% examples as test examples.

(Refer Time Slide: 01:48)



The screenshot shows a Jupyter Notebook with two sections. The first section, 'Load dataset', contains code to load the California Housing dataset using `sklearn.datasets.fetch_california_housing()` and split it into training and testing sets using `train_test_split()` with a 20% test size. The second section, 'Fit a pipeline to implement MLPRegressor on California housing dataset', contains code to create an `MLPRegressor` pipeline with a `StandardScaler` and three hidden layers of 32 neurons each. The code also includes cross-validation results.

```
1 # Load dataset
2 dataset = sklearn.datasets.fetch_california_housing()
3 X, y = dataset.data, dataset.target
4
5 X_train, X_test, y_train, y_test = train_test_split(
6     X, y, test_size=0.2, random_state=1)

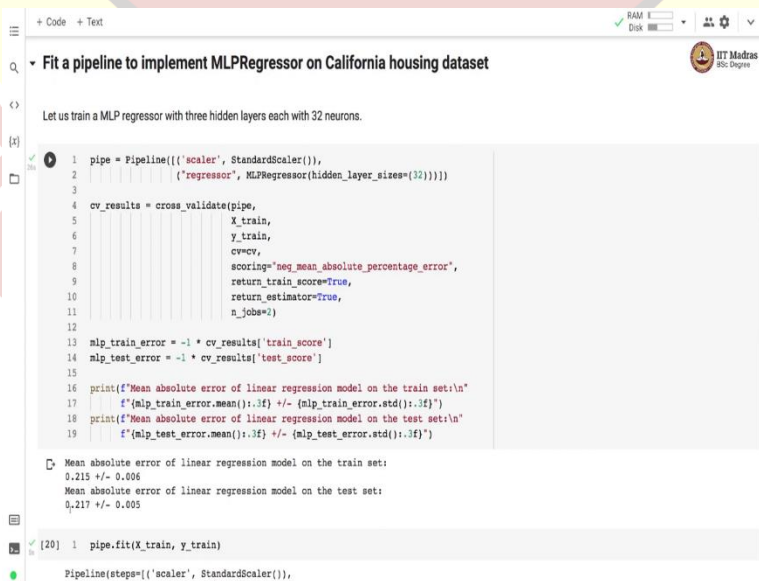
1 print('Shape of training data', X_train.shape)
2 print('Shape of testing data', X_test.shape)

Shape of training data (16512, 8)
Shape of testing data (4128, 8)

19 pipe = Pipeline([('scaler', StandardScaler()),
2                  ('regressor', MLPRegressor(hidden_layer_sizes=(32))))
3
4 cv_results = cross_validate(pipe,
5                             X_train,
6                             y_train,
7                             cv=cv,
8                             scoring='neg_mean_absolute_percentage_error',
9                             return_train_score=True,
10                            return_estimator=True,
11                            n_jobs=-1)
12
13 mlp_train_error = -1 * cv_results['train_score']
14 mlp_test_error = -1 * cv_results['test_score']
15
16 print(f'Mean absolute error of linear regression model on the train set:\n'
17       f'{mlp_train_error.mean():.3f} +/- (mlp_train_error.std():.3f)')
18 print(f'Mean absolute error of linear regression model on the test set:\n'
19       f'{mlp_test_error.mean():.3f} +/- (mlp_test_error.std():.3f)')
```

We begin by loading the California Housing dataset as we have loaded earlier, and California dataset is loaded using `fetch _California _housing` from `sklearn.datasets` module. We set a size 20% examples as test examples to train test split. After training test split we obtained training and test feature metrics and training and test label vectors. You can see that 16,512 examples are used for training and 4,128 examples are used as test examples. Each example in training and test is represented with 8 features.

(Refer Time Slide: 02:31)



The screenshot shows the third section of the Jupyter Notebook, which continues the code from the previous section. It includes the same pipeline creation and cross-validation code, but with additional code to calculate the mean absolute error (MAE) for the training and testing sets. The output shows the MAE for the training set as 0.215 +/- 0.006 and for the testing set as 0.217 +/- 0.005.

```
1 pipe = Pipeline([('scaler', StandardScaler()),
2                  ('regressor', MLPRegressor(hidden_layer_sizes=(32))))
3
4 cv_results = cross_validate(pipe,
5                             X_train,
6                             y_train,
7                             cv=cv,
8                             scoring='neg_mean_absolute_percentage_error',
9                             return_train_score=True,
10                            return_estimator=True,
11                            n_jobs=-1)
12
13 mlp_train_error = -1 * cv_results['train_score']
14 mlp_test_error = -1 * cv_results['test_score']
15
16 print(f'Mean absolute error of linear regression model on the train set:\n'
17       f'{mlp_train_error.mean():.3f} +/- (mlp_train_error.std():.3f)')
18 print(f'Mean absolute error of linear regression model on the test set:\n'
19       f'{mlp_test_error.mean():.3f} +/- (mlp_test_error.std():.3f)')

Mean absolute error of linear regression model on the train set:
0.215 +/- 0.006
Mean absolute error of linear regression model on the test set:
0.217 +/- 0.005

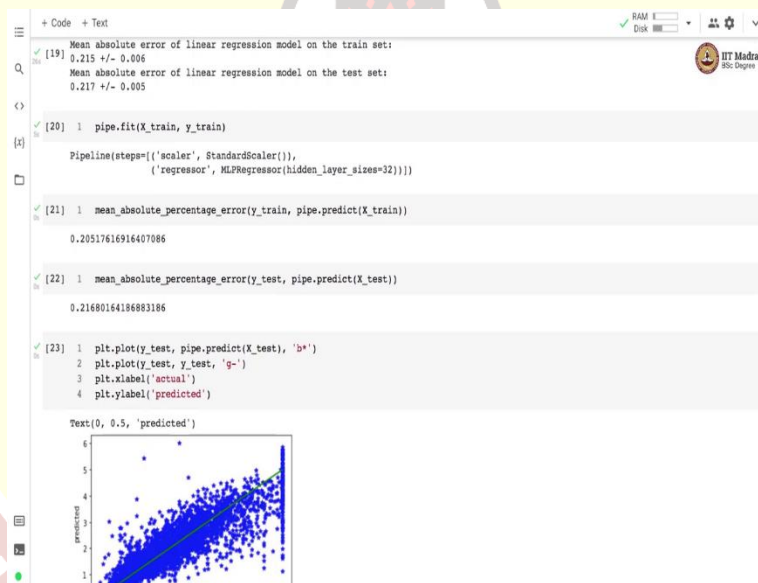
20 pipe.fit(X_train, y_train)

Pipeline(steps=[('scaler', StandardScaler()),
                 ('regressor', MLPRegressor(hidden_layer_sizes=(32)))])
```

We define a pipeline object with feature preprocessing which is StandardScaler and then using regressor with MLPRegressor with hidden layers set to 32. So, we are defining MLPRegressor with 1 hidden layer and this hidden layer has 32 different neurons, we train the pipeline with cross-validation by supplying the training feature metrics and training labels. We use ShuffleSplit cross-validation strategy in the CV parameter and we use negative mean \_absolute \_percentage \_error for scoring. We have set return train \_score and little estimators to true.

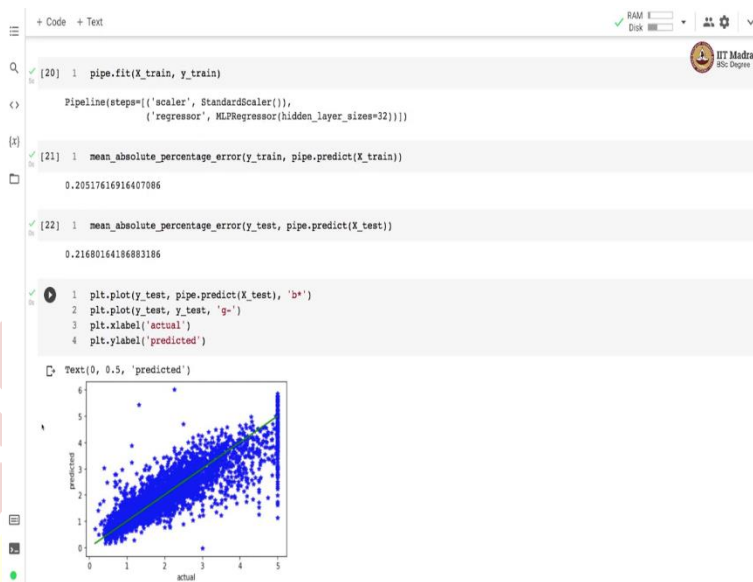
After training the model, we obtained mean \_absolute \_error on training set of 0.215 with a small standard deviation, the mean \_absolute \_percentage \_error on the test set was 0.217. So, the mean \_absolute \_percentage \_error is very similar on training and test set which indicates that the model is well trained and model is not underfitting or overfitting.

**(Refer Time Slide: 03:49)**



Then we obtain the mean \_absolute \_percentage \_error on training and test set. So, you can see that on the training the mean \_absolute \_percentage \_error is 0.205 And on test set it is 0.216.

(Refer Time Slide: 04:07)



We plotted the actual values and the predicted values. So, you can see that the actual value is represented with the green line which is at 45 degree and that is the line that shows the what should ideal prediction be. And you can see that the prediction is around this green line on either side. So, there is some error and which is which is what you see with mean \_absolute \_percentage \_error.

So, this is the predicted values are represented with blue stars. So, there are some miss classification or there are there are many wrong predictions for this value 5, and you have seen this earlier that the actual value of 5. This is more like a truncated value for all the houses with value > 5 million and that explains why our regressor is not working well, in this particular region and the rest of the region the regressor is performing satisfactorily.

So, in this video, we trained MLPRegressor on California housing dataset and demonstrated how to use MLPRegressor in regression problems.