

IIT Madras
ONLINE DEGREE

Machine Learning Practice

Indian Institute of Technology, Madras

Baseline Model

(Refer Slide Time: 0:10)

Table of contents

- Baseline models
- Imports
- LinearRegression classifier
- DummyRegressor
- permutation_test_score
- Model comparison
- Summary
- Section

Baseline models

In the last colab, we built a linear regression model. In this colab, we will build a couple of baseline models using DummyRegression and permutation_test_score. We will compare performance of our linear regression model with these two baselines.

Imports

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import matplotlib.pyplot as plt
4
5 from sklearn.datasets import fetch_california_housing
6 from sklearn.dummy import DummyRegressor
7
8 from sklearn.linear_model import LinearRegression
9
10 from sklearn.model_selection import cross_validate
11 from sklearn.model_selection import ShuffleSplit
12 from sklearn.model_selection import permutation_test_score
13 from sklearn.model_selection import train_test_split
14
15 from sklearn.pipeline import Pipeline
16 from sklearn.preprocessing import StandardScaler
```

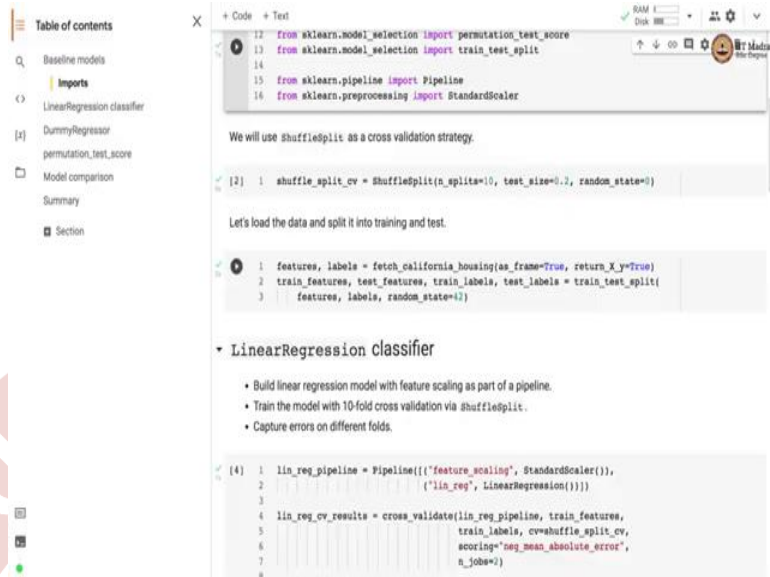
We will use ShuffleSplit as a cross validation strategy.

```
[2] 1 shuffle_split_cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)
```

LinearRegression classifier

Let's load the data and split it into training and test.

```
[3] 1 features, labels = fetch_california_housing(as_frame=True, return_X_y=True)
2 train_features, test_features, train_labels, test_labels = train_test_split(
3     features, labels, random_state=0)
```



The screenshot shows a Jupyter Notebook with a table of contents on the left and a code editor on the right. The code editor contains the following code:

```
12 from sklearn.model_selection import permutation_test_score
13 from sklearn.model_selection import train_test_split
14
15 from sklearn.pipeline import Pipeline
16 from sklearn.preprocessing import StandardScaler
```

Below the code, there is a text box that says: "We will use ShuffledSplit as a cross validation strategy."

```
[2] 1 shuffle_split_cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=0)
```

Below the code, there is a text box that says: "Let's load the data and split it into training and test."

```
[3] 1 features, labels = fetch_california_housing(as_frame=True, return_X_y=True)
2 train_features, test_features, train_labels, test_labels = train_test_split(
3     features, labels, random_state=0)
```

Below the code, there is a section titled "LinearRegression Classifier" with a bulleted list:

- Build linear regression model with feature scaling as part of a pipeline.
- Train the model with 10-fold cross validation via shuffledsplit.
- Capture errors on different folds.

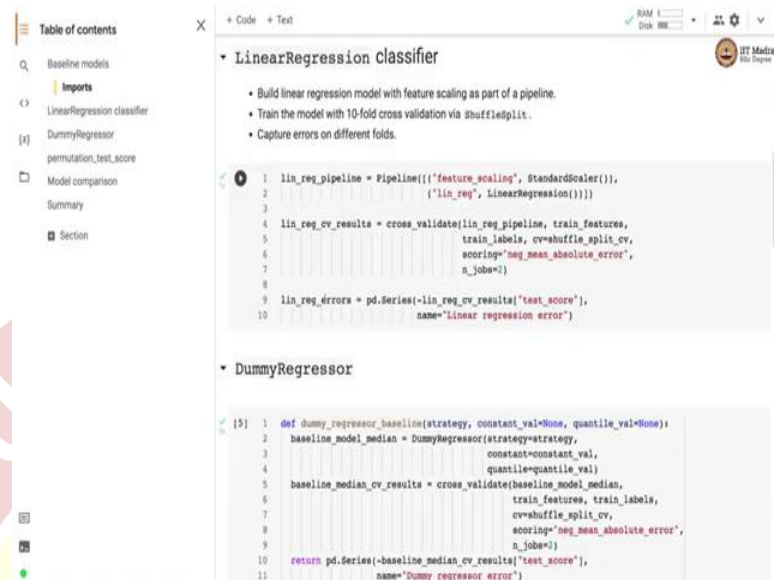
```
[4] 1 lin_reg_pipeline = Pipeline([("feature_scaling", StandardScaler()),
2                               ("lin_reg", LinearRegression())])
3
4 lin_reg_cv_results = cross_validate(lin_reg_pipeline, train_features,
5                                     train_labels, cv=shuffle_split_cv,
6                                     scoring="neg_mean_absolute_error",
7                                     n_jobs=-1)
```

Namaste welcome to the next video of the machine learning practice course. In this video, we will build a couple of baselines using dummy regression and permutation test score APIs. We will compare the performance of our linear regression model with these two base lines.

We begin by importing the necessary libraries. Here there are two additional libraries that we have imported, one is a dummy regressor that is implemented as part of sklearn.dummy module. And then we have permutation test code, that is imported from sklearn.model selection module.

Just like last colab we will use shuffle split as our cross-validation strategy and we come up with 10 folds, and in each fold, the test size is 0.2, that is 20 percent examples are set aside as test examples. We load the California housing data set and split it into training and test using train test split API.

(Refer Slide Time: 1:21)



The screenshot shows a Jupyter Notebook interface. On the left is a 'Table of contents' sidebar with links to 'Baseline models', 'Imports', 'LinearRegression classifier', 'DummyRegressor', 'permutation_test_score', 'Model comparison', 'Summary', and 'Section'. The main area is titled '+ Code + Text' and contains two sections of code. The first section, 'LinearRegression classifier', includes a bulleted list of tasks: 'Build linear regression model with feature scaling as part of a pipeline.', 'Train the model with 10-fold cross validation via shuffledsplit.', and 'Capture errors on different folds.' Below this is a code block with 10 lines of Python code that creates a pipeline with 'feature_scaling' and 'StandardScaler', performs cross-validation using 'cross_validate' with 'shuffledsplit_cv', and creates a 'pd.Series' for 'lin_reg_errors'. The second section, 'DummyRegressor', contains a code block with 11 lines of Python code that defines a 'dummy_regressor_baseline' function, which creates a 'DummyRegressor' model, performs cross-validation, and returns a 'pd.Series' for 'dummy_regressor_error'.

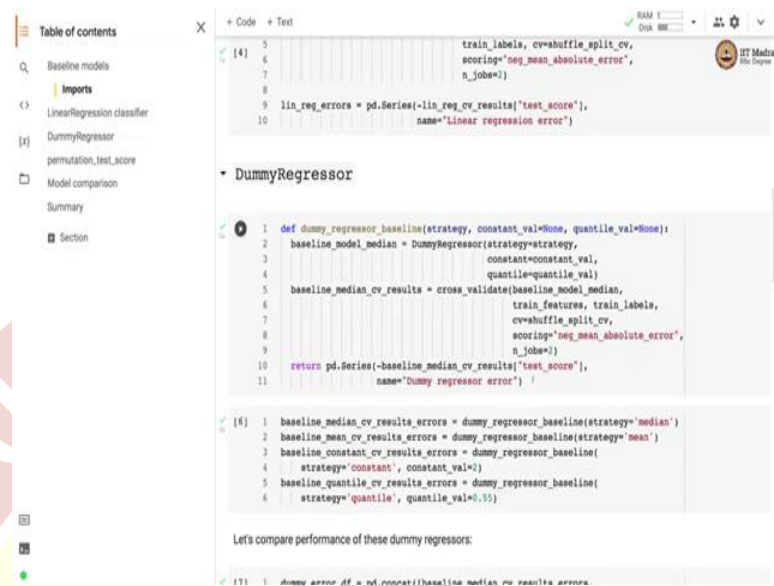
```
1 lin_reg_pipeline = Pipeline([('feature_scaling', StandardScaler()),
2                               ('lin_reg', LinearRegression())])
3
4 lin_reg_cv_results = cross_validate(lin_reg_pipeline, train_features,
5                                    train_labels, cv=shuffledsplit_cv,
6                                    scoring='neg_mean_absolute_error',
7                                    n_jobs=-1)
8
9 lin_reg_errors = pd.Series(-lin_reg_cv_results['test_score'],
10                            name='Linear regression error')
```

```
[5]: 1 def dummy_regressor_baseline(strategy, constant_val=None, quantile_val=None):
2     baseline_model_median = DummyRegressor(strategy=strategy,
3                                             constant=constant_val,
4                                             quantile=quantile_val)
5     baseline_median_cv_results = cross_validate(baseline_model_median,
6                                                train_features, train_labels,
7                                                cv=shuffledsplit_cv,
8                                                scoring='neg_mean_absolute_error',
9                                                n_jobs=-1)
10     return pd.Series(-baseline_median_cv_results['test_score'],
11                      name='Dummy regressor error')
```

We will first build our linear regression classifier based on what we studied in the last colab. We set up a pipeline containing the feature scaling that helps us to scale all the features on the same scale. And then we use the linear regression estimator for training the linear regression model with the transform feature set obtained via standard scaling.

Then we perform the training of this particular estimator to cross-validation. We use shuffle split cv as a cross-validation strategy and we obtain the scores based on negative mean absolute error. We convert the scores obtained to cross-validation into errors by multiplying them by minus 1. So, we now have test scores stored in linear regression error series.

(Refer Slide Time: 2:29)



```
train_labels, cv=shuffle_split_cv,
scoring='neg_mean_absolute_error',
n_jobs=1)

lin_reg_errors = pd.Series(-lin_reg_cv_results['test_score'],
                           name='Linear regression error')

# DummyRegressor

def dummy_regressor_baseline(strategy, constant_val=None, quantile_val=None):
    baseline_model_median = DummyRegressor(strategy=strategy,
                                           constant=constant_val,
                                           quantile=quantile_val)
    baseline_median_cv_results = cross_validate(baseline_model_median,
                                              train_features, train_labels,
                                              cv=shuffle_split_cv,
                                              scoring='neg_mean_absolute_error',
                                              n_jobs=1)
    return pd.Series(-baseline_median_cv_results['test_score'],
                    name='Dummy regressor error')

baseline_median_cv_results_errors = dummy_regressor_baseline(strategy='median')
baseline_mean_cv_results_errors = dummy_regressor_baseline(strategy='mean')
baseline_constant_cv_results_errors = dummy_regressor_baseline(
    strategy='constant', constant_val=2)
baseline_quantile_cv_results_errors = dummy_regressor_baseline(
    strategy='quantile', quantile_val=0.55)

Let's compare performance of these dummy regressors:

dummy_errors_df = pd.concat([baseline_median_cv_results_errors,
```

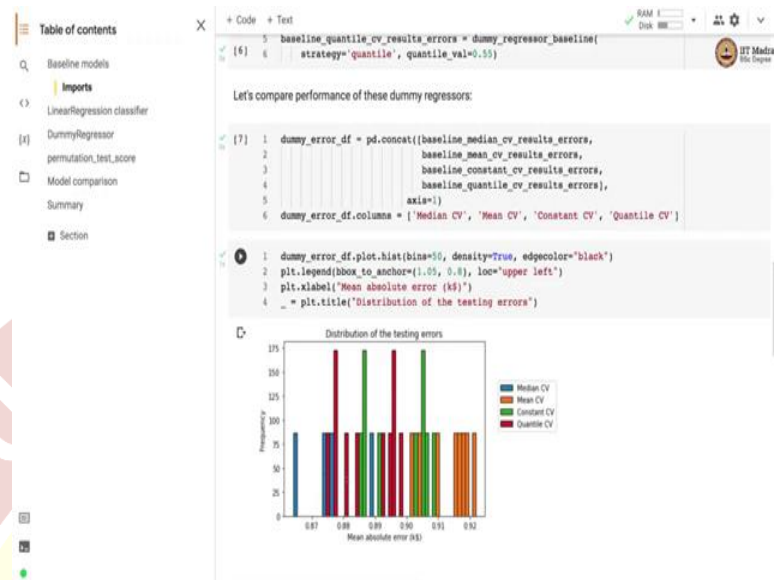
So, dummy regressor can be built using different strategies. And for that, we define a function, where we take strategies and input constant value and quantile values as additional inputs, that are default that is set to default arguments of none. The constant value is used in the constant strategy, while the quantile value is used in the quantile strategy.

In constant, we are going to use the value specified by the user, in the constant argument. Whereas, in the case of quantile, we generally use that quantile as a predicted value, as specified by the user.

So, here we instantiate the dummy regressor object with strategy constant and quantile values as specified by the user. Then we build then we train this dummy regressor using cross-validation-based training. And we finally convert the scores obtained in cross-validation to errors by multiplying them by -1 and storing them in the pandas series.

So, here we train four different dummy regressor baselines using strategies, that is median, mean, constant, and constant will predict always the value 2. And in quantile, we will predict the 0.55, 0.55 that is fifty-fifth quantile for of the. And in the quantile strategy, you have specified the quantile 0.55 as the value as the predicted value.

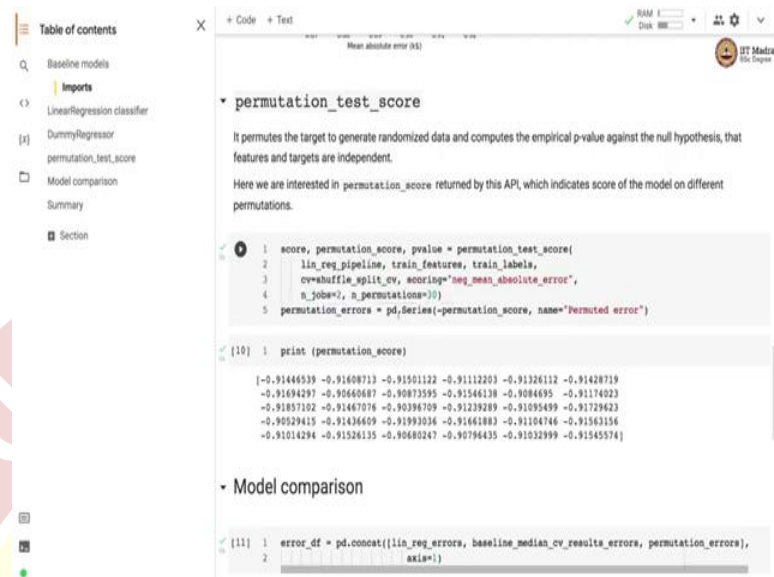
(Refer Slide Time: 4:45)



Let us compare the performance of these dummy regressors. We concatenate all the errors series, that we obtain to different strategies and we are plotting them. So, on X-axis, we have mean absolute error and on Y-axis, we have frequency. So, this is some kind of a histogram of the errors. And we have four different types of errors obtained through median strategy, mean strategy, constant strategy, and quantile strategy.

So, you can see that the mean strategy has the worst error. Whereas, median strategy seems to have better error. So, median's median strategy and quantile strategy, because we use quantile of 0.55, which is close to median and median is quantile 0.5. So, median and this quantile strategy, that we have better performance than other strategies.

(Refer Slide Time: 5:59)



The screenshot shows a Jupyter Notebook interface. On the left is a 'Table of contents' sidebar with links to 'Baseline models', 'Imports', 'LinearRegression classifier', 'DummyRegressor', 'permutation_test_score', 'Model comparison', 'Summary', and 'Section'. The main area is titled '+ Code + Text' and shows the following code and output:

```
1 score, permutation_score, pvalue = permutation_test_score(  
2     lin_reg_pipeline, train_features, train_labels,  
3     cv=shuffle_split_cv, scoring="neg_mean_absolute_error",  
4     n_jobs=-1, n_permutations=30)  
5 permutation_errors = pd.Series(-permutation_score, name="Permuted error")
```

The output of the code is a single value: -0.91446539.

```
1 print(permutation_score)
```

The output is a list of 30 values, representing the permutation scores for each of the 30 permutations:

```
[-0.91446539 -0.91608713 -0.91501122 -0.91112203 -0.91326112 -0.91428719  
-0.91694297 -0.90640687 -0.90873595 -0.91546138 -0.9084695 -0.91174023  
-0.91857102 -0.91467076 -0.90396709 -0.91239289 -0.91095499 -0.91729623  
-0.90529415 -0.91436409 -0.91993036 -0.91661883 -0.91104746 -0.91563156  
-0.91014294 -0.91526135 -0.90680247 -0.90796435 -0.91032999 -0.91545574]
```

Below the code, there is a section titled 'Model comparison' with the following code:

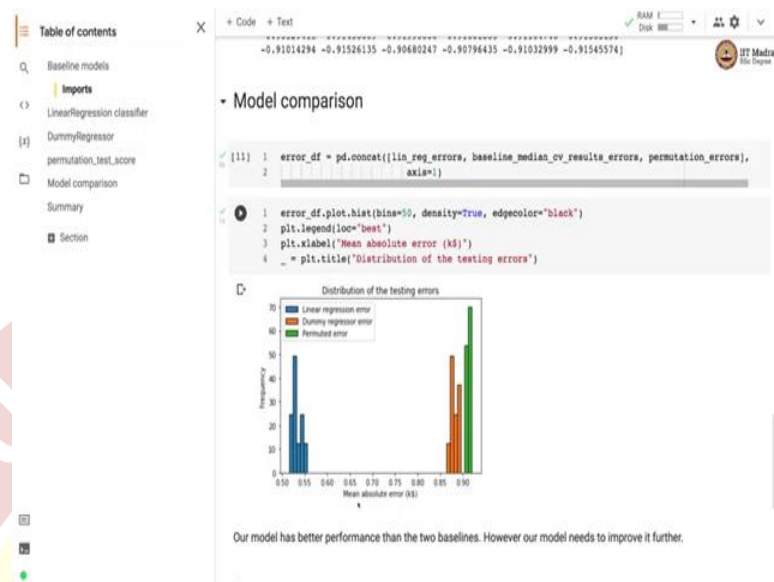
```
1 error_df = pd.concat([lin_reg_errors, baseline_median_cv_results_errors, permutation_errors],  
2                       axis=1)
```

Now, we build the second baseline model based on the permutation test score. Permutation test score permutes the target to generate randomized data and computes the empirical p-value against the null hypothesis, that features and targets are independent. Here we are interested in the permutation score written by this API, which indicates the score of models on different permutations.

So, permutation test score takes estimator training, training feature matrix, training labels, thus cross-validation strategy, scoring, and number of permutations as arguments. Here we have set number of permutations to 30.

So, here what happens is that there are 30 different permutations that are computed and for each permutation, we perform cross-validation-based training and we obtain the permutation score. So, you multiply this permutation score by minus 1 in order to obtain permutation error. And we store that in a panda series object.

(Refer Slide Time: 7:08)

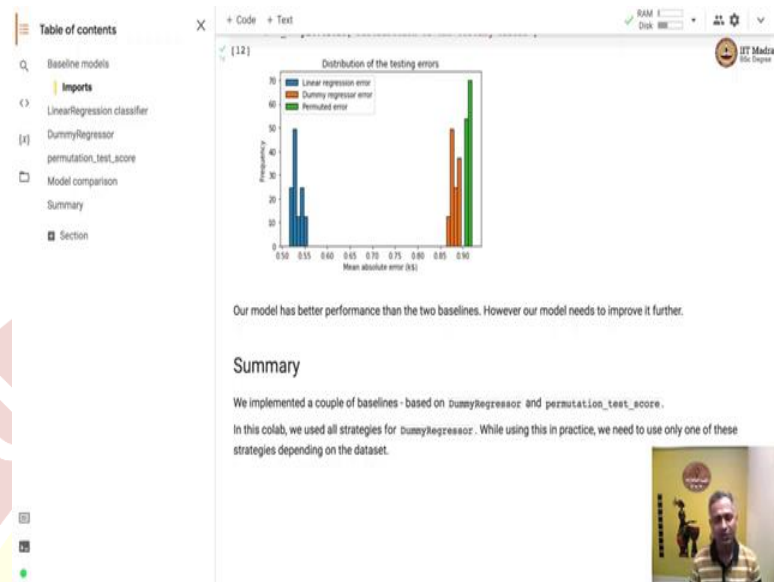


Now, we have three different linear regression models, one which is based on linear regression classifier, the second is based on dummy regressor and the third is based on the permutation test score. So, we use the median cv strategy in the dummy regressor for this comparison. So, we concatenate all these errors in the pandas' data frame and plot histograms.

So, you can see that our model performs definitely better than the two baselines, our mean square error is much smaller compared to the baseline models. The permutation error has worse performance among these three models. Whereas, dummy regressor performs slightly better than the permutation test score.

Going forward we will be using dummy regressor more as our baseline. Our model performance is also far from being ideal, but it is definitely better than the other two baselines because we have got much lesser mean absolute error than these two baselines.

(Refer Slide Time: 8:24)



So, in summary, we implemented a couple of baselines based on dummy regressor and permutation test score. Now, we have, now we have a good idea about how to build a baseline for any of the linear regression models. So, we can use two strategies, either based on dummy regressor or use a permutation test score. In this colab, we used all strategies for dummy regressor. In practice however we just need to use the strategy, that makes sense for that particular data set, that is it from this demonstration.