Bafna Gold
Date: Page:

3/3/24

LAB - 02

End-to-End ML Project:

Framing the problem:

For this project, we will use the California Census Data to build a model of the housing prices in the state. This Data includes features such as:

- Population
- Median Income
- Median housing price for each block group in California

This model will be able to predict the median housing price for any district

Select a Performance Measure:

The model is a regression model. A typical performance measure for regression problem is Root Mean Squared Error (RMSE). It has a higher weight for large errors.

$$RMSE(X, h) = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})^2}$$

where m = no. of instances

$x^{(i)}$ = vector containing all input feature values for i^{th} instance

$y^{(i)}$ = label or the desired output of input $x^{(i)}$

X = matrix containing all features values excluding the labels/targets

$h(\cdot)$ = system's prediction function, also called hypothesis.

MAE: Mean Absolute Error is used in the case where we have many outliers.

$$MAE(X,h) = \frac{1}{m}\sum_{i=1}^{m} |h(x^{(i)}) - y^{(i)}|$$

RMSE & MAE are ways to measure the distance b/w 2 vectors. In our case, the distance b/w the vector of predictions and the vector of targets/labels.

Get the Data:

```
import os
import tarfile
import urllib.

DOWNLOAD_ROOT = ""
HOUSING_PATH = os.path.join("data", "01")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url = HOUSING_URL, housing_path =
                       HOUSING_PATH)

# Creates HOUSING_PATH, Downloads & Extracts the contents of
# HOUSING_URL into HOUSING_PATH
    os.makedirs(name = housing_path, exist_ok = True)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(url = hosting_url, filename = tgz_path)
    housing_tgz = tarfile.open(name = tgz.path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()

fetch_housing_data().
```

load data using pandas:

```
def load_housing_data(housing_path = HOUSING_PATH):
    # loads housing data into pandas dataframe
    data_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(data_path)
```

Analyzing the data:

```
housing = load_housing_data()
housing.head()
```

The head() method returns the first 5 rows. The column name
will be returned in addition to specified rows.

```
housing.info()
```

The info method is useful to take a quick look at the data
it tells about - no. of rows, NaN values (per column), data types (per column)

From this we came to know, there exist 20,640 rows in the dataset. 207 districts are missing in "total_bedrooms".
All attributes are numerical except "ocean_proximity".
'ocean_proximity' is a categorical column.

```
housing.describe()
```

describe() gives Statistical values such as - count, mean, std, min, 25%, 50%, 75%, max values.

SP
26/3/24.

04/04/24

describe() ignores null values.
std shows standard deviation which measures how dispersed the values are.
25% of districts have <= 18 yrs housing median age.

# To get numerical continuous data : histogram.
import matplotlib.pyplot as plt
import seaborn as sns.
housing.hist(bins=50, figsize=(20, 15))
plt.show()

The median_income attribute isn't expressed in US dollars. The nos. roughly represents tens of thousands of dollars.
housing_median_age & median_house_values were capped. This means, in case of housing_median_age any value greater than 50 are represented as 50 in graph

Create test case:
Test ratio gives ratio of test data to total data.

Data is randomized and split. 80% data is used for training and 20% is used in test case

The random split can be maintained by using seed.
↓
This is not possible in case data gets updated

If unique identifier is not found, index can be used

Using train-test-split in scikit-learn:
If dataset is large enough, random sampling methods are good.
If dataset is small, there is a chance of introducing a significant sampling bias.

The population is divided into homogeneous subgroups called strata.
Each stratum represents an instance percentage of the overall population.

Test set to maintain the percentages of each stratum to make it's representative of the whole-population → stratified sampling.

Most median income values are clustered around $15,000 to $60000 but some median incomes go far beyond $60000.

It is important to have, for each stratum, a sufficient no. of instances in dataset or else there might be a bias while assessing the stratum importance

```
# pd.cut()   # Used to create strata.
housing['income_cat'] = pd.cut(x=housing['median_income'], bins=[
~~housing['income_cat']~~, 0, 1.5, 3, 4.5, 6, np.inf], labels=[1,2,3,4,5]).
housing['income_cat'].hist()
```

Discover & Visualize the Data to Gain Insights:
Exploring the training set:
strat_train_set.shape, strat_test_set.shape.

→

Visualizing Geographical Data:

```
housing.plot(kind='scatter', x='longitude', y='latitude')
plt.show()
```

The output scatter plot looks like California but we can't see any other pattern, Set the alpha to 0.1 to make it easier to estimate densities.

```
housing.plot(kind='scatter', x='longitude', y='latitude', alpha=0.1)
plt.show()
```

The radius of each circle represents the districts population. The color represents price.

```
housing.plot(kind='scatter', x='longitude', y='latitude', alpha=0.4,
             s=housing['population']/100., label='population',
             figsize=(10,7), c='median_house_value',
             cmap=plt.get_cmap(name='jet'), colorbar=True)
plt.legend()
```

The image generated tells us that the median housing price is related to location (closer to sea → more expensive).

Looking for correlations:

```
corr_matrix = housing.corr()
corr_matrix['median_house_value'].sort_values(ascending=False)
```

The correlation coefficient is $\in [-1, 1]$. When coeff. is close to 1 it means there exist a strong positive correlation b/w the 2 variables. When coeff. is close to -1, it means there exist a strong -ve correlation b/w 2 variables. When coeff is close to 0, weak correlation b/w 2 variables

Experimenting with attribute combinations:

bedrooms_per_room is much more correlated with median_house_value meaning, more expensive the house, the less the bedrooms per room ratio.

Prepare the Data for ML Algorithms:

Data cleaning: remove the null values from the dataset so that the ML Algo. works properly, or set the missing values to some value (zero, mean, median, etc).

Handling Text & Categorical ~~Attributes~~:

Categorical data converted to ordinal categorical number using ordinal encoder.

Custom Transformers to easily test/experiment with diff. attributes.

Select and Train a Model:
Training & Evaluating on the Training set:
Train a Linear Regression model
Performance measured using RMSE metric.
Better evaluation using K-fold Cross-Validation.

Fine Tune Your Model
Grid search can be used to check on various parameters.

Randomized search

→

Launch, Monitor & Maintain your system

Sp 1
4/4/24