AI Jungle Docs

EsharkyTheGreat

11-05-2024

Contents

1 Statistics

1.1 Correlation

Pearson's Correlation Coefficient The correlation coefficient ranges from –1 to 1. When it is close to 1, it means that there is a strong positive correlation; for example, the median house value tends to go up when the median income goes up. When the coefficient is close to –1, it means that there is a strong negative correlation; you can see a small negative correlation between the latitude and the median house value (i.e., prices have a slight tendency to go down when you go north). Finally, coefficients close to zero mean that there is no linear correlation

2 Data Processing

2.1 Data Splitting

It is important that we immediately split the data into training and testing sets so that there is no data snooping bias (Our brain always looks at pattern and is good at overfitting therefore when we look at the whole data we might apply techniques that overfit the data and the model becomes unusable).

2.1.1 Random Splitting

We take a random seed and split the data into training and testing sets. But the problem is that the data might not be evenly distributed and if we add more data the split might change due to this some data in the original training set will come in the testing set causing bias.

```
import numpy as np

def split_train_test(data, test_ratio):
    np.random.seed(42)
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

2.1.2 Checksum Splitting

We calculate the hash of the data and split the data based on the hash. This way the data will always be split the same way and we can add new data without worrying about the split changing.

```
from zlib import crc32

    def test_set_check(identifier, test_ratio):
        return crc32(np.int64(identifier)) & 0xffffffff < test_ratio * 2**32
    def split_train_test_by_id(data, test_ratio, id_column):
        ids = data[id_column]
        in_test_set = ids.apply(lambda id_: test_set_check(id_, test_ratio))
        return data.loc[~in_test_set], data.loc[in_test_set]
```

2.1.3 Stratified Splitting

When we want the same distribution of categorical data in the main dataset to be present in both the training and testing data set because of the importance of the distribution to the model we use Stratified Splitting

```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

3 Machine Learning

3.1 Linear Regression

3.2 Logistic Regression

3.3 Error

Root Mean Square Error (RMSE) - L2 Norm

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(h(x_i) - \hat{y_i})^2} \tag{1}$$

$$\begin{aligned} where, \quad h(x_i) &= \text{predicted value} \\ h &= \text{hypothesis function} \\ \hat{y_i} &= \text{actual value} \end{aligned}$$

This is useful when you want to know how far off your predictions are from the actual values. Cases where the model is far from the correct value is treated more harshly then the cases that are nearby but not exactly correct.

Mean Absolute Error (MAE) - L1 Norm

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|h(x_i) - \hat{y_i}| \tag{2}$$

$$\begin{aligned} where, \quad h(x_i) &= \text{predicted value} \\ h &= \text{hypothesis function} \\ \hat{y_i} &= \text{actual value} \end{aligned}$$

We mostly use RMSE but when there are too many outlier cases that we can't have the model be harsh on all of them we use MAE that averages out the error.

4 Python Tips and Tricks

4.1 Sklearn

4.2 Matplot

4.3 Numpy

4.4 Pandas