week-02

October 2, 2024

1 Week 02 ML ZoomCamp

[37]:
```python
# import libraries
import pandas as pd
import numpy as np

# import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import mean_squared_error

import warnings
warnings.filterwarnings('ignore')
```

Next, use only the following columns: ##### 'ram', 'storage', 'screen', 'final_price'

EDA :
Look at the final_price variable. Does it have a long tail?

[38]:
```python
# Read CSV and select only specified columns, convert all capital to small cases
def read(file_path):
    df = pd.read_csv(file_path)
    df_new = df[['RAM', 'Storage', 'Screen', 'Final Price']]
    df_new = df_new.rename(columns={'Final Price':'final_price'})
    df_new.columns = df_new.columns.str.lower().str.replace(' ', '_')
    return df_new
```

[39]:
```python
dataframe = read('data/laptops.csv')
dataframe.head(2)
```

[39]:
```
   ram  storage  screen  final_price
0    8      512    15.6       1009.0
1    8      256    15.6        299.0
```

1.1 Question 1

1.1.1 There's one column with missing values. What is it?

1) 'ram'

2) 'storage'
3) 'screen'
4) 'final_price'

Correct Answer is 'screen' column

```
[40]: dataframe.isnull().sum()>0
```

```
[40]: ram            False
      storage        False
      screen          True
      final_price    False
      dtype: bool
```

1.2 Question 2

1.2.1 What's the median (50% percentile) for variable 'ram'?

1) 8
2) 16
3) 24
4) 32

Correct Answer is 2) 16

```
[41]: dataframe['ram'].median()
```

```
[41]: np.float64(16.0)
```

****** End of Question 2 ******

1.3 Question 3

Prepare and split the dataset
Shuffle the dataset (the filtered one you created above), use seed 42.
Split your data in train/val/test sets, with 60%/20%/20% distribution.

We need to deal with missing values for the column from Q1.
We have two options: fill it with 0 or with the mean of this variable.
Try both options. For each, train a linear regression model without regularization using the code from the lessons.
For computing the mean, use the training only!
Use the validation dataset to evaluate the models and compare the RMSE of each option.
Round the RMSE scores to 2 decimal digits using round(score, 2)

1.3.1 Which option gives better RMSE?

Options: 1) With 0 2) With mean 3) Both are equally good

option 1) filling with 0 value

[42]:
```
# Shuffel the dataset with random seed 42
# splitting the data into train validation and test 60:20:20 ratio
#
def split_dataset(dataframe):
    dataframe_new = dataframe.sample(frac=1, random_state=42)
    train, val_test = train_test_split(dataframe_new, test_size=0.4,␣
 ↪random_state=42)
    val, test = train_test_split(val_test, test_size=0.5, random_state=42)
    return train, val, test
```

[43]:
```
train, val, test = split_dataset(dataframe)

# Lets check if the data is split into proper ratio or not
print(f'Size of the Dataframe = {len(dataframe)}')
print(f'Size of the Training data = {len(train)}')
print(f'Size of the Validation data = {len(val)}')
print(f'Size of the Test data = {len(test)}')
```

```
Size of the Dataframe = 2160
Size of the Training data = 1296
Size of the Validation data = 432
Size of the Test data = 432
```

[44]:
```
# Lest first copy the data before performing filling mean and 0 values
train_zero = train.copy()
val_zero = val.copy()

train_mean = train.copy()
val_mean = val.copy()
```

[45]:
```
# Lets implement Linear Regression with Zero fill

def linear_regression(train, val, regularisation):

    if regularisation == 'zero':
        train_ = train.copy()
        val_ = val.copy()
        train_['screen'].fillna(0, inplace=True)
        val_['screen'].fillna(0, inplace=True)

    elif regularisation == 'mean':

        mean_value = train['screen'].mean()
        train_ = train.copy()
        val_ = val.copy()

        train_['screen'].fillna(mean_value, inplace=True)
        val_['screen'].fillna(mean_value, inplace=True)

    X_train = train_.drop('final_price', axis=1)
    y_train = train_['final_price']
    X_val = val_.drop('final_price', axis=1)
    y_val = val_['final_price']

    zero_fill_model = LinearRegression()
    zero_fill_model.fit(X_train, y_train)

    y_predict_zero = zero_fill_model.predict(X_val)
    rmse_score = round(np.sqrt(mean_squared_error(y_val, y_predict_zero)), 2)

    return rmse_score
```

[46]:
```
rmse_zero = linear_regression(train,val, 'zero')
rmse_mean = linear_regression(train,val, 'mean')
print(f'RMSE score with zero fill = {rmse_zero}')
print(f'RMSE score with mean value fill = {rmse_mean}')
```

```
RMSE score with zero fill = 675.08
RMSE score with mean value fill = 675.16
```

[47]:
```
if rmse_zero < rmse_mean:
    print("Filling with 0 gives better RMSE")
elif rmse_mean < rmse_zero:
    print("Filling with mean gives better RMSE")
else:
    print("Both options are equally good")
```

```
Filling with 0 gives better RMSE
```

****** End of Question 3 ******

1.4 Question 4

Now let's train a regularized linear regression.
For this question, fill the NAs with 0.
Try different values of r from this list: [0, 0.01, 0.1, 1, 5, 10, 100].
Use RMSE to evaluate the model on the validation dataset.
Round the RMSE scores to 2 decimal digits.

1.4.1 Which r gives the best RMSE?

If there are multiple options, select the smallest r.

Options: 1) 0 2) 0.01 3) 1 4) 10 5) 100

Option 5) 100

[50]:
```
# Lets implement Linear Regression with Zero fill

def ridge(train, val):


    train_ = train.copy()
    val_ = val.copy()
    train_['screen'].fillna(0, inplace=True)
    val_['screen'].fillna(0, inplace=True)


    X_train = train_.drop('final_price', axis=1)
    y_train = train_['final_price']
    X_val = val_.drop('final_price', axis=1)
    y_val = val_['final_price']

    # List of r (regularization strength) values to try
    r_values = [0, 0.01, 0.1, 1, 5, 10, 100]
    best_r = None
    best_rmse = float('inf')

    # Loop over r values
    for r in r_values:
        # Train Ridge regression model with regularization strength r
        model_ridge = Ridge(alpha=r)
        model_ridge.fit(X_train, y_train)

        # Predict on validation set
        y_pred = model_ridge.predict(X_val)

        # Calculate RMSE
        rmse = np.sqrt(mean_squared_error(y_val, y_pred))
        rmse = round(rmse, 2)

        # Print RMSE for each r
        print(f"RMSE with r={r}: {rmse}")

        # Check if this is the best RMSE and update accordingly
        if rmse < best_rmse:
            best_rmse = rmse
            best_r = r

    print(f"Best r value: {best_r} with RMSE: {best_rmse}")
```

[51]:
```
ridge(train, val)
```

```
RMSE with r=0: 675.08
RMSE with r=0.01: 675.08
RMSE with r=0.1: 675.08
RMSE with r=1: 675.08
RMSE with r=5: 675.08
RMSE with r=10: 675.08
RMSE with r=100: 675.01
Best r value: 100 with RMSE: 675.01
```

***** end of Question 4 ****

1.5 Question 5

We used seed 42 for splitting the data. Let's find out how selecting the seed influences our score.
Try different seed values: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9].
For each seed, do the train/validation/test split with 60%/20%/20% distribution.
Fill the missing values with 0 and train a model without regularization.
For each seed, evaluate the model on the validation dataset and collect the RMSE scores.
What's the standard deviation of all the scores? To compute the standard deviation, use np.std.
Round the result to 3 decimal digits (round(std, 3))

1.5.1 What's the value of std?

1) 19.176
2) 29.176
3) 39.176
4) 49.176

Note: Standard deviation shows how different the values are. If it's low, then all values are approximately the same. If it's high, the values are different. If standard deviation of scores is low, then our model is stable.

[52]:
```python
seeds = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
rmse_scores = []

for seed in seeds:
    # Shuffle and split the dataset
    df_shuffled = dataframe.sample(frac=1, random_state=seed)
    train, temp = train_test_split(df_shuffled, test_size=0.4,␣
↪random_state=seed)
    val, test = train_test_split(temp, test_size=0.5, random_state=seed)

    # Fill missing values with 0
    train_ = train.copy()
    val_ = val.copy()
    train_['screen'].fillna(0, inplace=True)
    val_['screen'].fillna(0, inplace=True)

    # Prepare features and target
    X_train = train_.drop('final_price', axis=1)  # Adjust 'Final Price' as␣
↪needed
    y_train = train_['final_price']
    X_val = val_.drop('final_price', axis=1)
    y_val = val_['final_price']

    # Train the linear regression model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Predict on validation set and compute RMSE
    y_pred = model.predict(X_val)
    rmse = np.sqrt(mean_squared_error(y_val, y_pred))

    # Round RMSE before appending
    rmse_scores.append(round(rmse, 2))

# Compute standard deviation of RMSE scores
std = round(np.std(rmse_scores), 3)

# Print RMSE scores and the standard deviation
print(f"RMSE scores: {rmse_scores}")
print(f"Standard deviation of RMSE scores: {std}")
```

```
RMSE scores: [np.float64(614.85), np.float64(618.61), np.float64(597.36),
np.float64(627.3), np.float64(575.78), np.float64(558.34), np.float64(595.68),
np.float64(562.84), np.float64(575.49), np.float64(564.54)]
Standard deviation of RMSE scores: 23.888
```

1.6 Question 6

Split the dataset like previously, use seed 9.
Combine train and validation datasets.
Fill the missing values with 0 and train a model with r=0.001.

1.6.1 What's the RMSE on the test dataset?

Options: 1) 598.60 2) 608.60 3) 618.60 4) 628.60

[53]:
```
# Step 1: Shuffle and split the dataset using seed 9
seed = 9
df_shuffled = dataframe.sample(frac=1, random_state=seed)
train, temp = train_test_split(df_shuffled, test_size=0.4, random_state=seed)
val, test = train_test_split(temp, test_size=0.5, random_state=seed)

# Step 2: Combine train and validation datasets
train_val_combined = pd.concat([train, val], ignore_index=True)
```

```python
# Step 3: Fill missing values with 0
train_val_combined['screen'].fillna(0, inplace=True)

# Prepare features and target
X_train_val = train_val_combined.drop('final_price', axis=1)  # Adjust 'Final␣
↪Price' as needed
y_train_val = train_val_combined['final_price']

# Step 4: Train a Ridge regression model with r=0.001
r = 0.001
model = Ridge(alpha=r)  # Use alpha as the regularization parameter for Ridge
model.fit(X_train_val, y_train_val)

# Step 5: Evaluate the model on the test dataset
# Fill missing values in the test set
test['screen'].fillna(0, inplace=True)
X_test = test.drop('final_price', axis=1)
y_test = test['final_price']

# Predict and calculate RMSE
y_pred_test = model.predict(X_test)
rmse_test = np.sqrt(mean_squared_error(y_test, y_pred_test))

# Print the RMSE on the test dataset
print(f"RMSE on the test dataset: {round(rmse_test, 2)}")
```

```
RMSE on the test dataset: 552.86
```

1.6.2 Note:

Question 5 and 6 Score are nearest to the options provided