COSE474-2024F: Deep Learning

2.2 Data Prepocessing

2.2.1. Reading the Dataset

Comma-separated values (CSV) files are ubiquitous for the storing of tabular (spreadsheet-like) data. In them, each line corresponds to one record and consists of several (comma-separated) fields, e.g., “Albert Einstein,March 14 1879,Ulm,Federal polytechnic school,field of gravitational physics”. To demonstrate how to load CSV files with pandas, we create a CSV file below ../data/house_tiny.csv. This file represents a dataset of homes, where each row corresponds to a distinct home and the columns correspond to the number of rooms (NumRooms), the roof type (RoofType), and the price (Price).

```
import os

os.makedirs(os.path.join('..', 'data'), exist_ok=True)
data_file = os.path.join('..', 'data', 'house_tiny.csv')
with open(data_file, 'w') as f:
    f.write('''NumRooms,RoofType,Price
NA,NA,127500
2,NA,106000
4,Slate,178100
NA,NA,140000''')
```

```
import pandas as pd

data = pd.read_csv(data_file)
print(data)
```

```
   NumRooms RoofType   Price
0       NaN      NaN  127500
1       2.0      NaN  106000
2       4.0    Slate  178100
3       NaN      NaN  140000
```

2.2.2. Data Preparation

```
inputs, targets = data.iloc[:, 0:2], data.iloc[:, 2]
inputs = pd.get_dummies(inputs, dummy_na=True)
print(inputs)
```

```
   NumRooms  RoofType_Slate  RoofType_nan
0       NaN           False          True
1       2.0           False          True
2       4.0            True         False
3       NaN           False          True
```

```
inputs = inputs.fillna(inputs.mean())
print(inputs)
```

```
   NumRooms  RoofType_Slate  RoofType_nan
0       3.0           False          True
1       2.0           False          True
2       4.0            True         False
3       3.0           False          True
```

2.2.3. Conversion to the Tensor Format

```
import torch

X = torch.tensor(inputs.to_numpy(dtype=float))
```

```
y = torch.tensor(targets.to_numpy(dtype=float))
X, y
```

```
(tensor([[3., 0., 1.],
         [2., 0., 1.],
         [4., 1., 0.],
         [3., 0., 1.]], dtype=torch.float64),
 tensor([127500., 106000., 178100., 140000.], dtype=torch.float64))
```

2.2.4. Discussion and Takeaways

- The use of mean imputation is straightforward and useful when you have missing numerical values, but this method assumes that missing values are random.

2.2.4.1. My own exercise

```
import numpy as np

data = {
    'Name': ['John', 'Sarah', 'Tom', 'Lucy', 'David'],
    'Gender': ['Male', 'Female', 'Male', 'Female', 'Male'],
    'Math': [85, 78, None, 95, 70],
    'Physics': [90, 85, 80, None, 75],
    'Passed': ['Yes', 'No', 'Yes', 'Yes', 'No']
}

df = pd.DataFrame(data)

df['Math'].fillna(df['Math'].mean(), inplace=True)
df['Physics'].fillna(df['Physics'].mean(), inplace=True)

df_encoded = pd.get_dummies(df, columns=['Gender', 'Passed'])
df_encoded.drop(columns=['Name'], inplace=True)

X = torch.tensor(df_encoded.drop(columns=['Passed_No', 'Passed_Yes']).values.astype(np.float32), dtype=torch.float32)
y = torch.tensor(df_encoded['Passed_Yes'].values.astype(np.float32), dtype=torch.float32)


X
```

```
tensor([[85.0000, 90.0000,  0.0000,  1.0000],
        [78.0000, 85.0000,  1.0000,  0.0000],
        [82.0000, 80.0000,  0.0000,  1.0000],
        [95.0000, 82.5000,  1.0000,  0.0000],
        [70.0000, 75.0000,  0.0000,  1.0000]])
```

```
y
```

```
tensor([1., 0., 1., 1., 0.])
```

- In this exercise, I used mean imputation to fill in the missing scores for each subject. This approach is simple but effective for small datasets.
- I used one-hot encoding to convert the categorical Gender and Passed columns into numerical data.
- Passed_Yes is treated as the label for a binary classification task.