Introduction To Artificial Intelligence And Machine Learning.

Adithya Nair

July 27, 2024

Contents

1 Notes

1.1 Iris Data Classification

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

irisdata = pd.read_csv('iris.csv')

test, train = train_test_split(irisdata, train_size=0.8, test_size=0.2)

print(np.size(test))
print(np.size(train))
print(irisdata.describe())
```

1.2 Data Pre-Processing

1.2.1 Handling Missing Values (Imputation)

When the no. of missing values in a feature or on a whole in a dataset, is beyond a certain percentage. It might lead to wrong interpretations and might misguide the ML models. Hence it is essential to handle the missing values.

1. CREATING A DATAFRAME

```
import pandas as pd
import numpy as np

# Load the Titanic dataset
df = pd.read_csv('titanic.csv')

# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(df.head())
```

   This dataset is not complete, Cabin and Age have values that are unfilled. We can verify this here.

```
# Identify missing values
print("\nMissing values in each column:")
print(df.isnull().sum())
```

2. There are two main methods in dealing with missing values.

   (a) Dropping rows with missing values.

   (b) Filling the empty missing values with zeros.

```
# Method 1: Drop rows with missing values
df_dropped = df.dropna()
print("\n METHOD 1 Shape of dataset after dropping rows with missing values:", df_

# Method 2: Fill missing values with a specific value (e.g., 0)
df_filled_zeros = df.fillna(0)
```

```
print("\nMETHOD 2 Missing values filled with 0:")
print(df_filled_zeros.isnull().sum())
```

This isn’t exactly ideal. Deleting the rows loses too much of the dataset, and filling with zeros does not work here when that might affect the correctness of the prediction. So here we replace the values with the mean for numerical values and mode for categorical values.

(a) **TODO** Look into other methods of imputation

```
# Method 3: Fill missing values with the mean (for numerical columns)
df[’Age’].fillna(df[’Age’].mean(), inplace=True)
print("\nMETHOD 3 Missing values in ’Age’ column after filling with mean:")
print(df[’Age’].isnull().sum())

# Method 4: Fill missing values with the most frequent value (mode)
df[’Embarked’].fillna(df[’Embarked’].mode()[0], inplace=True)
print("\nMETHOD 4 Missing values in ’Embarked’ column after filling with mode
print(df[’Embarked’].isnull().sum())
```

3. Forward fill and Backward Fill There are two better ways to fill the rows.

- Forward Fill - It iterates down the given data, and fills in missing values with the last value it saw.
- Backward Fill - it iterates up the given data, and fills in missing values with the last value it saw.

```
# Method 5: Forward fill method
df_ffill = df.fillna(method=’ffill’)
print("\nMethod 5 Missing values handled using forward fill method:")
print(df_ffill.isnull().sum())

# Method 6: Backward fill method
df_bfill = df.fillna(method=’bfill’)
print("\nMethod 6 Missing values handled using backward fill method:")
print(df_bfill.isnull().sum())
print("****************")
```

2 Table

Name	Marks
Adithya	100
Neha	400

```
Method 5 Missing values handled using forward fill method:
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          1
Embarked       0
dtype: int64

Method 6 Missing values handled using backward fill method:
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          1
Embarked       0
dtype: int64
*****************
```

2.0.1 Normalization

Used for multiple numerical features in the dataset, which belong to different ranges. I t would make ssense to normalize the data to a particular range.

Machine learning models tend to give a higher weightage to numerical attributres which have a larger value.

The solution is to normalize. Normalization reduces a given numerical feature into a range that is easier to manage as well as equate with other numerical features.

1. Types Of Normalization

   - MinMaxScaler - all data points are brought to the range $[0, 1]$
   - Z-score - Data points are converted in such a way that the mean becomes 0 and the standard deviation is 1.

   (a) NORMALISING A SET OF VALUES USING MIN MAX NORMALIZATION

   ```
   import numpy as np
   from sklearn.preprocessing import MinMaxScaler

   # Example usage:
   data = np.array([2, 5, 8, 11, 14]).reshape(-1, 1)  # Reshape to 2D array for

   # Initialize the MinMaxScaler
   scaler = MinMaxScaler()

   # Apply Min-Max normalization
   normalized_data = scaler.fit_transform(data)

   # Flatten the normalized data to 1D array
   normalized_data = normalized_data.flatten()

   print(normalized_data)
   ```

   (b) NORMALISING A SET OF VALUES USING Z-SCORE NORMALIZATION

   ```
   import numpy as np
   from sklearn.preprocessing import StandardScaler
   ```

```
# Example usage:
data = np.array([2, 5, 8, 11, 14]).reshape(-1, 1)  # Reshape to 2D array for

# Initialize the StandardScaler
scaler = StandardScaler()

# Apply Z-score normalization
normalized_data = scaler.fit_transform(data)

# Flatten the normalized data to 1D array
normalized_data = normalized_data.flatten()

print(normalized_data)
```

(c) NORMALIZING CERTAIN COLUMNS IN THE DATAFRAME

```
# Initialize the MinMaxScaler
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

# List of columns to be normalized
columns_to_normalize = ['Age', 'Fare']

# Apply Min-Max normalization
df[columns_to_normalize] = scaler.fit_transform(df[columns_to_normalize])

print("\nDataFrame after Min-Max normalization:")
print(df)
```

2.0.2 SAMPLING

1. RANDOM SAMPLING Random sampling is used for when the dataset is hella large.

```
import random

# Sample data
population = list(range(1, 101))  # Population from 1 to 100
sample_size = 10  # Size of the sample
```

```
# Simple random sampling
sample = random.sample(population, sample_size)
print("Simple Random Sample:", sample)
```

2. STRATIFIED SAMPLING

```
import random

# Sample data with strata
strata_data = {
    ’stratum1’: [1, 2, 3, 4, 5],
    ’stratum2’: [6, 7, 8, 9, 10],
}

# Sample size per stratum
sample_size_per_stratum = 2

# Stratified sampling
sample = []
for stratum, data in strata_data.items():
    stratum_sample = random.sample(data, sample_size_per_stratum)
    sample.extend(stratum_sample)

print("Stratified Sample:", sample)
```

3. Systematic Sampling

```
# Sample data
data = list(range(1, 101))  # Data from 1 to 100
n = 5  # Every nth data point to be included in the sample

# Systematic sampling
sample = data[::n]
print("Systematic Sample:", sample)

import random

# Sample data with clusters
clusters = {
    ’cluster1’: [1, 2, 3],
    'cluster2': [4, 5, 6],
    'cluster3': [7, 8, 9],
}

# Number of clusters to sample
clusters_to_sample = 2

# Cluster sampling
selected_clusters = random.sample(list(clusters.keys()), clusters_to_sample)
print("chosen clusters ", selected_clusters)
sample = []
for cluster in selected_clusters:
    sample.extend(clusters[cluster])

print("Cluster Sample:", sample)
```

2.0.3 BINNING

```
import pandas as pd

df = pd.read_csv('bollywood.csv')
budget_bins = [0, 10, 20, float('inf')]  # Define your budget bins
budget_labels = ['Low Budget', 'Medium Budget', 'High Budget']  # Labels for the bins
df['BudgetBin'] = pd.cut(df['Budget'], bins=budget_bins, labels=budget_labels)
print(df.head(10))

collection_bins = [0, 20, 40, 60, float('inf')]  # Define your collection bins
collection_labels = ['Low Collection', 'Medium Collection', 'High Collection', 'Very Hi

df['CollectionBin'] = pd.cut(df['BoxOfficeCollection'], bins=collection_bins, labels=co
df.head(10)

import matplotlib.pyplot as plt
budget_bin_counts = df['BudgetBin'].value_counts()
# Plot the data as a bar chart
plt.figure(figsize=(8, 6))
budget_bin_counts.plot(kind='bar', color='skyblue')
plt.title('Number of Movies in Each Budget Bin')
plt.xlabel('Budget Bin')
plt.ylabel('Number of Movies')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()
```

2.1 Supervised Learning