KNN-prediction-by-retzam-ai

April 5, 2024

[49]:
```
# We'll use pandas library for data manipulation.
import pandas as pd
```

[50]:
```
# Import the car data file and use the first column as the title
df = pd.read_csv('CarsData.csv', header=0)
df.head()
```

[50]:
```
           model  year  price transmission  mileage fuelType  tax   mpg  \
0            I10  2017   7495       Manual    11630   Petrol  145  60.1
1           Polo  2017  10989       Manual     9200   Petrol  145  58.9
2       2 Series  2019  27990    Semi-Auto     1614   Diesel  145  49.6
3   Yeti Outdoor  2017  12495       Manual    30960   Diesel  150  62.8
4         Fiesta  2017   7999       Manual    19353   Petrol  125  54.3

   engineSize Manufacturer
0         1.0       hyundi
1         1.0   volkswagen
2         2.0          BMW
3         2.0        skoda
4         1.2         ford
```

[51]:
```
# Convert each column with nominal data to numbers from 0, 1, 2...
df["model"], _ = pd.factorize(df["model"])
df["fuelType"], _ = pd.factorize(df["fuelType"])
df["transmission"], _ = pd.factorize(df["transmission"])
df.head()
```

[51]:
```
   model  year  price  transmission  mileage  fuelType  tax   mpg  engineSize  \
0      0  2017   7495             0    11630         0  145  60.1         1.0
1      1  2017  10989             0     9200         0  145  58.9         1.0
2      2  2019  27990             1     1614         1  145  49.6         2.0
3      3  2017  12495             0    30960         1  150  62.8         2.0
4      4  2017   7999             0    19353         0  125  54.3         1.2

  Manufacturer
0       hyundi
1   volkswagen
2          BMW
```

```
3         skoda
4          ford
```

[52]:
```
"""
  Manufacturer feature is our output/target, it also has nominal data.
  We want to convert it to numbers, but we want to know what each number␣
 ↪represents.
  So we first get all the unique strings and replace them in the column.
"""
unique_values = df['Manufacturer'].unique()
unique_values
```

[52]:
```
array(['hyundi', 'volkswagen', 'BMW', 'skoda', 'ford', 'toyota', 'merc',
       'vauxhall', 'Audi'], dtype=object)
```

[53]:
```
# Create a map using the unique values array above.
mapping = {
   'hyundi': 0,
   'volkswagen': 1,
   'BMW': 2,
   'skoda': 3,
   'ford': 4,
   'toyota': 5,
   'merc': 6,
   'vauxhall': 7,
   'Audi': 8,
}

# Replace the values
df['Manufacturer'] = df['Manufacturer'].replace(mapping)
```

[54]:
```
# Use numpy library for array manipulations and calculations.
import numpy as np
```

[57]:
```
"""
  Split dataset into training and test.

  df.sample: randomizes the array so the training data get all feature vectors.
  np.split: splits an array in multiple sub arrays.
  0.8*len(df): this makes the split to be 0-80% for training and, 20% for test.
"""
train, test = np.split(df.sample(frac=1), [int(0.8*len(df))])
```

[58]:
```
# Import StandardScaler for standardization
from sklearn.preprocessing import StandardScaler
# Import RandomOverSampler of over-sampling
from imblearn.over_sampling import RandomOverSampler
```

[59]:

```python
# Scale dataset so better prediction can be made.
def scale_dataset(dataframe, oversample=False):
  # This selects all columns in the DataFrame except the last one as the␣
 ↪features.
  X = dataframe[dataframe.columns[:-1]].values

  # This selects the last column in the DataFrame as the target.
  y = dataframe[dataframe.columns[-1]].values

  # This removes the mean and scaling to unit variance
  # Known as standardization. Basically removes outliers.
  scaler = StandardScaler()
  X = scaler.fit_transform(X)

  """
    Make both x and y sets equal sets as appropriate.

    RandomOverSampler is important in cases where there is alot more features␣
 ↪vector of a
    specific output.

    Example if you have a dataset with 100 rows with output as "Yes" and 20
    rows with "No".
    You can see that our datasets would be biased towards the output with "Yes".
    To solve this, RandomOverSampler strategically duplicates rows with "No" so␣
 ↪the dataset ends up
    having 100 rows with "Yes" and 100 with "No" outputs.

    This is called over-sampling.
  """
  if oversample:
    ros = RandomOverSampler()
    X, y = ros.fit_resample(X, y)

  # Stack horizontally
  # Reshape y and concatenate it with X
  # This simply means attaching each feature vector with the appropriate output.
  data = np.hstack((X, np.reshape(y, (-1, 1))))

  return data, X, y
```

[60]:

```python
# Scale training and test datasets
train, X_train, y_train = scale_dataset(train, oversample=True)

# Test sets are not oversampled because they are used to test new data
test, X_test, y_test = scale_dataset(test, oversample=False)
```

[61]:
```python
# Import KNeighborsClassifier as our classifier
from sklearn.neighbors import KNeighborsClassifier
```

[62]:
```python
# Use the KNN classifier lib from sklearn, use neighbours: k=5
knn_model = KNeighborsClassifier(n_neighbors=5)

# Train model with training dataset
knn_model.fit(X_train, y_train)
```

[62]:
```
KNeighborsClassifier()
```

[63]:
```python
# Make predictions for all the features in the test dataset
y_pred = knn_model.predict(X_test)
y_pred
```

[63]:
```
array([8, 5, 2, …, 1, 2, 6])
```

[64]:
```python
# Use classification_report to get the performance report of the model
from sklearn.metrics import classification_report
```

[65]:
```python
# Get the performance of the model the prediction it made and the output on the␣
↪actual test dataset.
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.96      0.92       951
           1       0.88      0.86      0.87      3028
           2       0.75      0.81      0.78      2065
           3       0.79      0.86      0.82      1203
           4       0.92      0.91      0.92      3505
           5       0.92      0.94      0.93      1394
           6       0.87      0.85      0.86      2561
           7       0.93      0.92      0.93      2618
           8       0.86      0.81      0.83      2218

    accuracy                           0.87     19543
   macro avg       0.87      0.88      0.87     19543
weighted avg       0.88      0.87      0.88     19543
```

[ ]:
```python

```