Practical 07: Data Analysis

Dr. Todd Jones

t.r.jones@reading.ac.uk

Department of Computer Science

Follow the instructions to complete each of these tasks.

This set of exercises focuses on practice working with regression, classification, and clustering tools with in `scikit-learn`, along with visualisation of their results.

This work is not assessed but will help you gain practical experience for the coursework.

Predicting Bicycle Traffic with Linear Regression

In this challenge, you will try to **model** the number of bicycle trips that might occur for given weather conditions (`Fremont_Bridge_Bicycle_Counter.csv`).

We will link the bicycle data with **weather station** data (`Weather_Station_data.csv`).

`Fremont_Bridge_Bicycle_Counter.csv` :

Column	Description
Date	The date and hour of day object(s) are detected by the sensor, Date & Time
Fremont Bridge Total	Total of both sidewalks, Number
Fremont Bridge East Sidewalk	The total number of bicyclists traveling on the East sidewalk in one hour as recorded by the sensor, Number
Fremont Bridge West Sidewalk	The total number of bicyclists traveling on the West sidewalk in one hour as recorded by the sensor, Number

Source: https://data.seattle.gov/widgets/65db-xm6k?mobile_redirect=true

---

`Weather_Station_data.csv`

Column	Description
NAME	Station Name
DATE	Observation Date (DD MM YYYY)
PRCP	Precipitation

Column	Description
SNWD	Snow depth
SNOW	Snowfall
TAVG	Average Temperature
TMAX	Maximum temperature
TMIN	Minimum temperature
WSF2	Fastest 2-minute wind speed
WSF5	Fastest 5-second wind speed
WT**	Weather types

Full Field Descriptions: https://docs.opendata.aws/noaa-ghcn-pds/readme.html

Source: https://www.ncei.noaa.gov/products/land-based-station/global-historical-climatology-network-daily

---

Imports

```
In [ ]: import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        import numpy as np
        from sklearn.model_selection import train_test_split
```

Read in the Data

Read in both files. Be sure to `parse_dates` and set the date information to be the index column.

Explore the contents a bit to get familiar with the data.

```
In [ ]: # Handle the file dates and select them to become the DatetimeIndex

        bx = pd.read_csv('./data/Fremont_Bridge_Bicycle_Counter.csv',
                         parse_dates=True,
                         date_format="%m/%d/%Y %I:%M:%S %p",
                         # date_format="mixed", # Less specific than above, but should also work
                         index_col=0)
        wx = pd.read_csv('./data/Weather_Station_data.csv',
                         parse_dates=True,
                         dayfirst=True,
                         index_col=1)
```

```
In [ ]: # Inspect bike data
        bx
```

```
In [ ]: # Inspect the bike data (to investigate correct date transformation)
        bx.iloc[24*24]
```

```
In [ ]: wx
```

Transform the Data

You will notice that the bicycle data and the weather data are recorded at different frequencies.

Use the `DataFrame.resample()` method to modify the bicycle data. We would like to have both datasets configured with the same time frequency.

In the resampled data, we should collect the **total** number of bicycle passes in each weather observational period.

Hint: If this is done correctly, the bicycle and weather data should have the same length.

https://pandas.pydata.org/pandas-docs/version/1.4/reference/api/pandas.DataFrame.resample.html

In [ ]:
```
# Create daily total values
bxd = bx.resample('D').sum()
bxd
```

Merge the Datasets

Using the dates as a common element, use `pd.merge()` to join the two datasets into a single `DataFrame`.

Hints:

- Be sure to merge `on` the date column name.
- Be sure that the date column name is the same in each original dataset.

https://pandas.pydata.org/pandas-docs/version/1.4/reference/api/pandas.merge.html

In [ ]:
```

```

Identify Response Variable and Predictors

From the collection of columns, identify the column we are trying to predict (total bicycle bridge traffic).

Then, have a look at the candidate predictors.

Establish whether there is any **correlation** between the target variable and the independent variables or among the independent variables (predictors).

- Consider looking at a correlation heatmap or a pairplot.

Then, arrange the data into an isolated matrix of features and a separate target variable.

- You might first test the upcoming regression model with all suitable variables as predictors and later test with various subsets based on the results of your correlation analysis.

In [ ]:
```
mx.columns
```

In [ ]:
```
# Solutions will vary
data = mx.drop(columns=[???])
```

In [ ]:
```
# Split target variable and independent variables
```

Multiple Linear Regression

Complete the following tasks:

1. Split the data into training and test datasets (holdout testing).

2. Configure a linear regression model with `sklearn`.

3. Fit the model to the data.

4. Check the robustness of the model through cross validation.

5. Make predictions based on the test holdout data and evaluate their utility using appropriate evaluation metrics (values and figures).

In [ ]:
```
# Solutions will vary
```

In [ ]:
```
# Splitting to training and testing data
```

In [ ]:
```
y_train.shape
```

In [ ]:
```
# Create a Linear regression model instance
```

In [ ]:
```
## Report the metrics in the testing data
```

In [ ]:
```
# Visualise the results
```

In [ ]:
```
## Begin cross validation of the results
```

Load and Prepare the Data for Classification and Clustering

This work has been done for you. We will work with the **iris dataset**.

The data is split into training and testing sets with `train_test_split` with a 70/30 split.

The data is scaled using the `StandardScaler` class to avoid biasing the model in the direction of any particular variable.

- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

Inspect the `iris` variable, the predictors, and the response variable to understand their form.

In [ ]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import seaborn as sns

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import classification_report

from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN

from clustering_utils import elbow_point

from sklearn.neighbors import NearestNeighbors

from sklearn.metrics import silhouette_score
from sklearn.metrics import silhouette_samples
from sklearn.metrics import rand_score, adjusted_rand_score

import warnings
# warnings.filterwarnings('ignore')
warnings.filterwarnings("ignore", category=FutureWarning, module="sklearn")
warnings.filterwarnings("ignore", category=UserWarning, module="sklearn")
```

In [ ]:
```
# Load the iris dataset
iris = load_iris()

# Prepare the data for modeling
X = iris.data
y = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = ???

# Scale the data
scaler = StandardScaler()
X_train = scaler.???
X_test = scaler.???
```

In [ ]:
```
print(iris['DESCR'])
```

In [ ]:
```
# Inspect labels
y
```

In [ ]:
```
df=pd.DataFrame(X_test, columns=iris.feature_names)
sns.pairplot(df)
```

Logistic Regression

Train and evaluate a logistic regression model on the iris dataset.

- For multiple classes, `sklearn` should default to `multinomial` for this dataset.

1. Create the logistic regression model object
2. Train the model on the training data
3. Evaluate the model by reporting its score
4. Obtain the predicted classes based on the test input.
5. Evaluate the model's performance:
   - Use `confusion_matrix`.
   - Use `classification_report`.
   - Create a scatterplot that compares both the known test classes and the predicted test classes.
     - This could be done with two calls to `sns.scatterplot()`: one for the known and one for the predicted values, giving the first a larger marker size so that any coincident points are both visible atop one another.

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

Support Vector Machine (Advanced Extension Activity)

Train and evaluate a support vector machine (SVM) model on the iris dataset.

You can reuse the same training and testing data from above.

1. Create the SVM object (try an `rbf` kernel)
2. Train the model on the training data
3. Evaluate the model by reporting its score
4. Obtain the predicted classes based on the test input.
5. Evaluate the model's performance:
   - Use `confusion_matrix`.
   - Use `classification_report`.
   - Create a scatterplot that compares both the known test classes and the predicted test classes.
     - This could be done with two calls to `sns.scatterplot()`: one for the known and one for the predicted values, giving the first a larger marker size so that any coincident points are both visible atop one another.

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

K-Means Clustering

Train and evaluate a k-means clustering model on the iris dataset.

- To start, see how well it does when assuming 3 clusters.
- What do the silhouette score and elbow method indicate about the correct number of clusters?
  - What does the clustering look like if we use this suggestion?

In [ ]:
```
# (3 clusters)
```

In [ ]:
```
```

In [ ]:
```
```

In [ ]:
```
```

In [ ]:
```
# Calculating the inertia for the elbow method.
```

In [ ]:
```
# Calculating the silhouette score
```

In [ ]:
```
# (2 clusters)
```

In [ ]:
```
```

In [ ]:
```
```

In [ ]:
```
```

DBSCAN (Advanced Extension Activity)

Train and evaluate a DBSCAN clustering model on the iris dataset.

- How many clusters are suggested with the default values of `eps` and `min_samples` ?
- Try to determine an optimal set of parameter values to return the known number of clusters.
  - When this is reached, how well does the clustering perform, in comparison to the known labels?

In [ ]:
```
```

In [ ]:
```
```

In [ ]:
```
```

In [ ]:
```
```

In [ ]:
```
```