Drexel University

College of Computing and Informatics

INFO 371 – Data Mining Applications

Assignment 1

Name: Yuming Chen

Student Number: 320180939611

Set date: Monday, March 29, 2021 Due Date: 11:59pm, Sunday, April 11, 2021

Catalog

1. Automated classification learning using nearest neighbor

1.1 Prepare the data in the proper ARFF format and save it as “mydata.arff”.

The origin dataset I used in this assignment was a heart disease dataset, which contains 14 descriptive features found in Cleveland dataset from UCI Machine Learning Repository, downloaded from Kaggle. Then, I used the python script to select 2 descriptive features and restructure it into standard format of Weka’s ARFF file. The objective of this dataset is to build a model to predict whether a patient diagnose a heart disease.

1.1.1 Python code

```
# -*- coding: utf-8 -*-
"""
INFO-371 Data-Mining-Applications Assignment 1
Simple python script to process and reformat "csv" file into "arff" file.
"""
__author__ = "Yuming Chen"
__copyright__ = "Copyright 2021, Study Project of INFO 371, China"
__license__ = "GPL V3"
__status__ = "Experimental"

import pandas as pd

class ArffWriter:
    def __init__(self, data: pd.DataFrame, attributes: dict, filename: str):
        self.data = data
        self.attributes = attributes
        self.filename = filename

    def write(self):
        with open(self.filename+".arff", "w") as f:
            f.write("@relation {}\n\n".format(self.filename))
            for name, attribute in attributes.items():
                f.write("%{}\n".format(attribute["des"]))
                f.write("@attribute {name} {type}\n"
                        .format(name=name, type=attribute["type"]))
            f.write("\n@data\n")
            for value in data.values:
                f.write("{},{},{}\n".format(value[0],value[1],value[2]))

if __name__ == "__main__":
    filename = "mydata"
    data = pd.read_csv('dataset.csv')
    data = data.groupby('target', group_keys=False)\
        .apply(pd.DataFrame.sample, n=10)[["age", "thalach", "target"]]
    data["target"] = data["target"].apply(lambda x: ["low", "high"][x])
    attributes = {"age":
                      {"type": "numeric",
                       "des": "The age of the patient"},
                  "thalach":
                      {"type": "numeric",
                       "des": "The maximum heart rate achieved"},
                  "target":
                      {"type": "{low, high}",
                       "des": "The chance of heart attack"}}
    arffwriter = ArffWriter(data,attributes,filename)
    arffwriter.write()
```

1.1.2 Introduction of Attributes

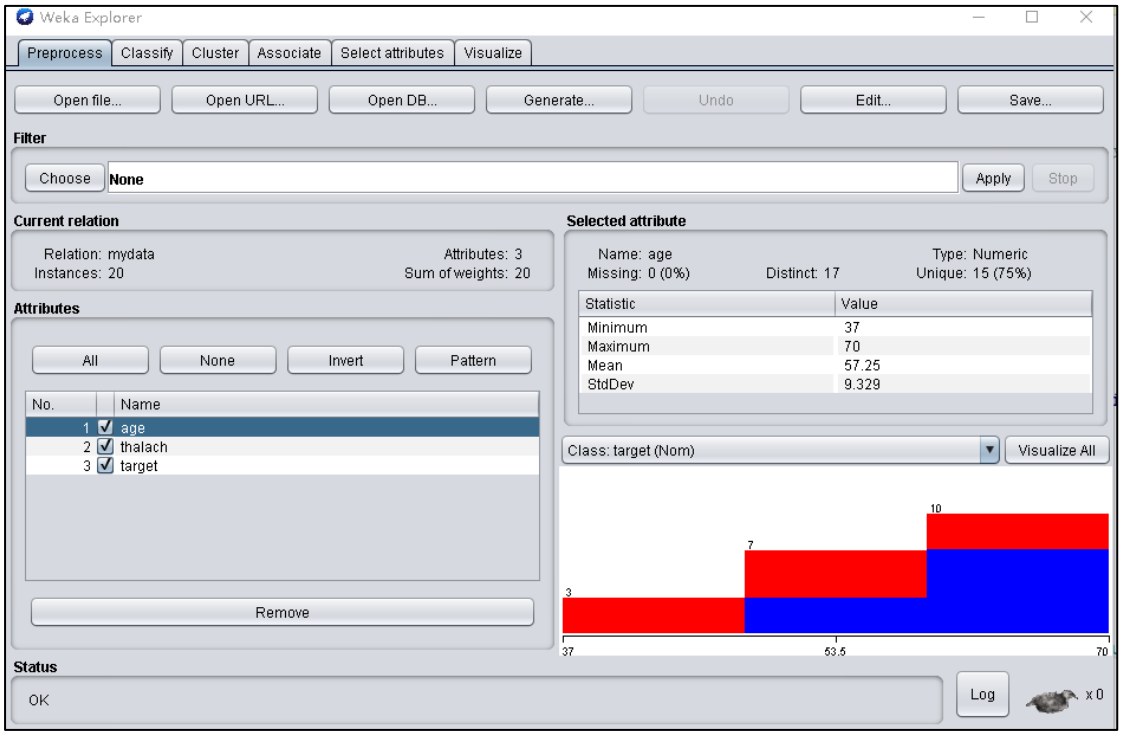
Name	Type	Description
age	numeric (numbers)	The age of the patient
thalach	numeric (numbers)	The maximum heart rate achieved of patient
target	nominal (categorical)	The chance of heart attack of patient 1. **high**: low level of chance of heart attack 2.**low**: high level of chance of heart attack

1.1.3 Overview of ARFF file

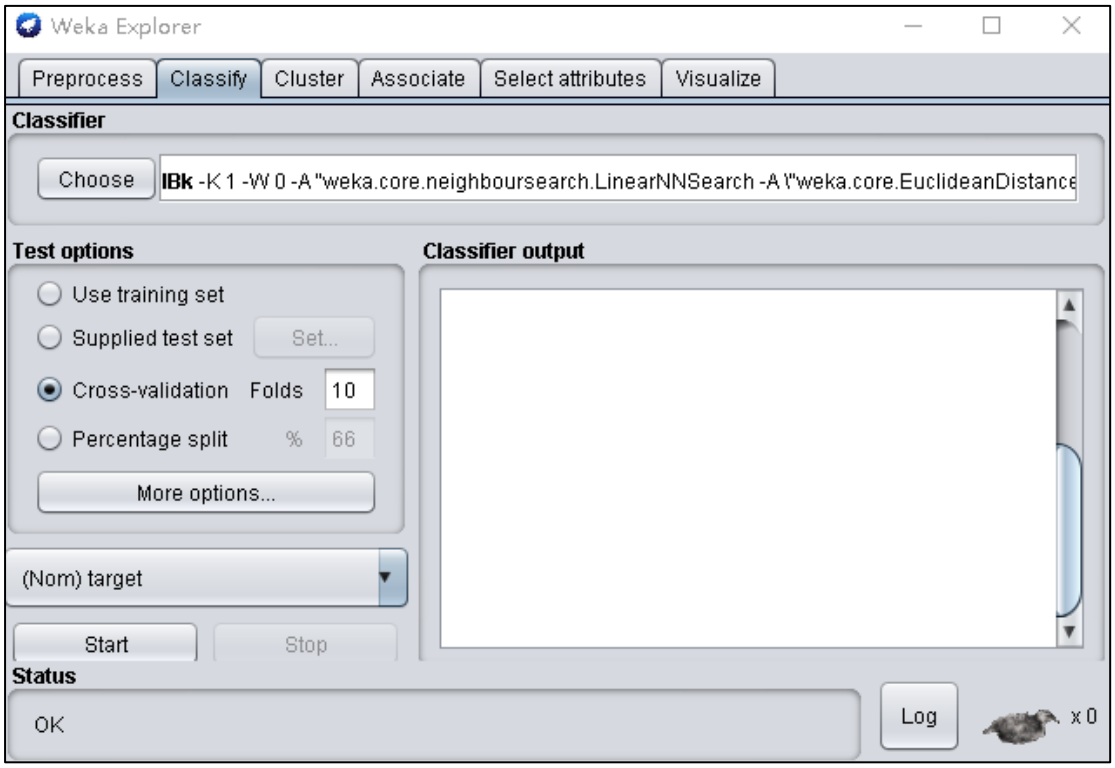
```
1   @relation mydata
2
3   %The age of the patient
4   @attribute age numeric
5   %The maximum heart rate achieved
6   @attribute thalach numeric
7   %The chance of heart attack
8   @attribute target {low, high}
9
10  @data
11  44,188,high
12  67,125,low
13  58,154,high

                    ......

29  50,163,low
30  42,178,high
```

1.2 Run in Weka

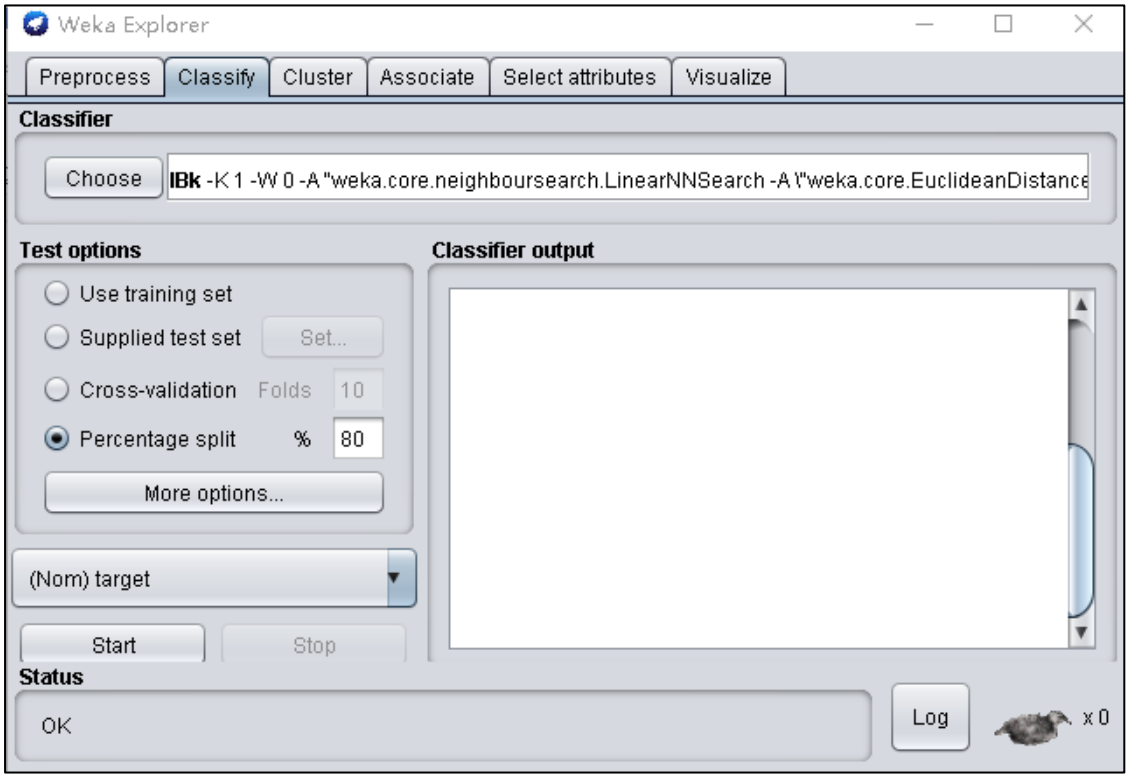
1.2.1 Open Weka and select Explorer; then click "Open file…" to load mydata.arff.



1.2.1.1 Go to the Classify tab and select Lazy -> IBk

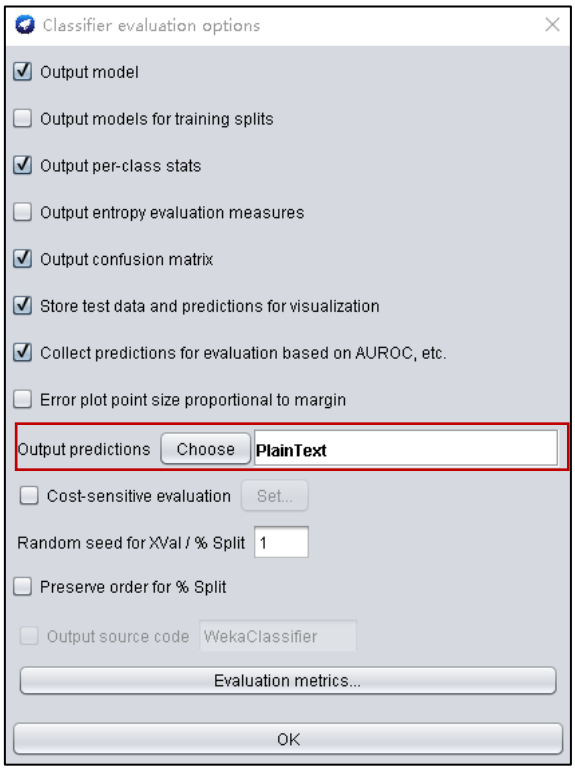


1.2.1.2 Choose 80% percentage split (for training vs. testing), which means 16 instances (80%) will be used for training and the last 4 instances (20%) will be used for testing.

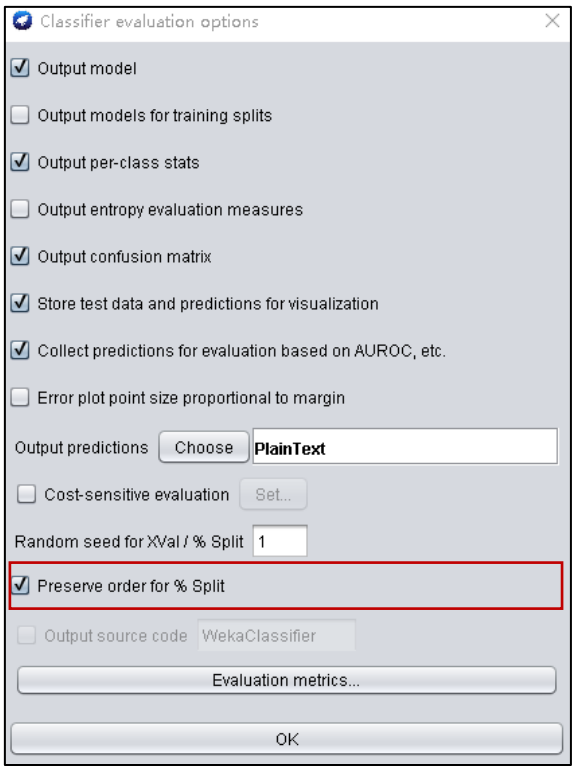


1.2.1.3 Click "More options…"

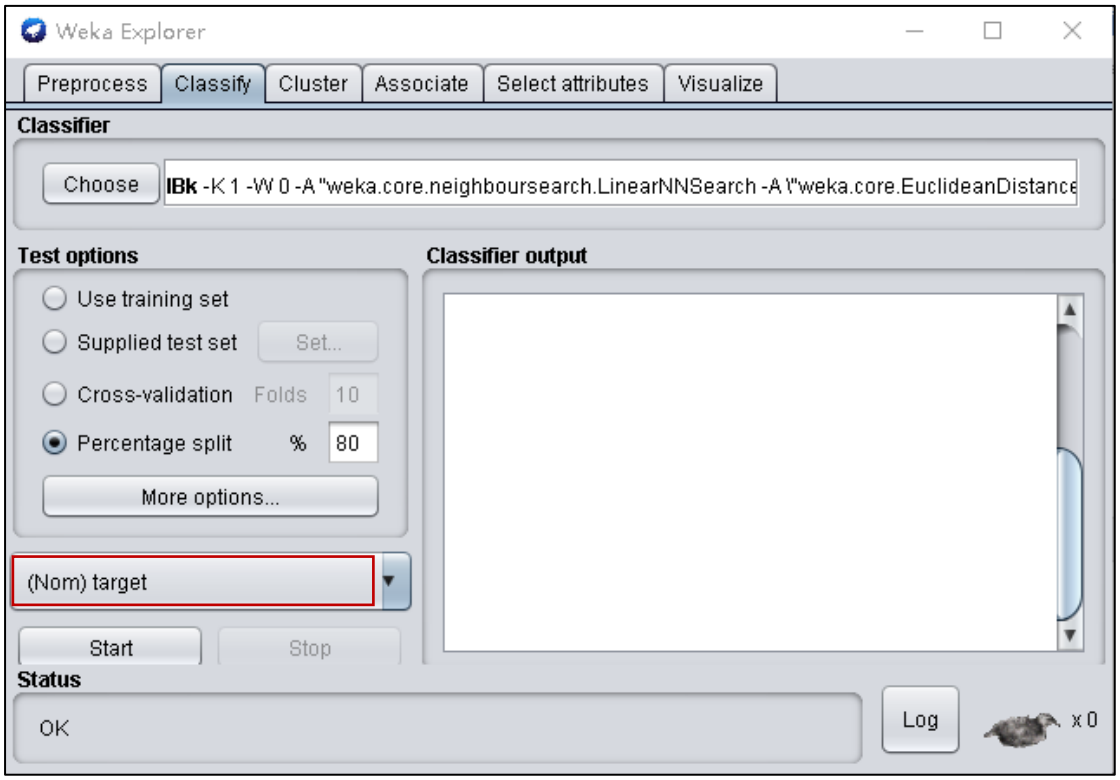
a. Choose "Plain text" for Output predictions;



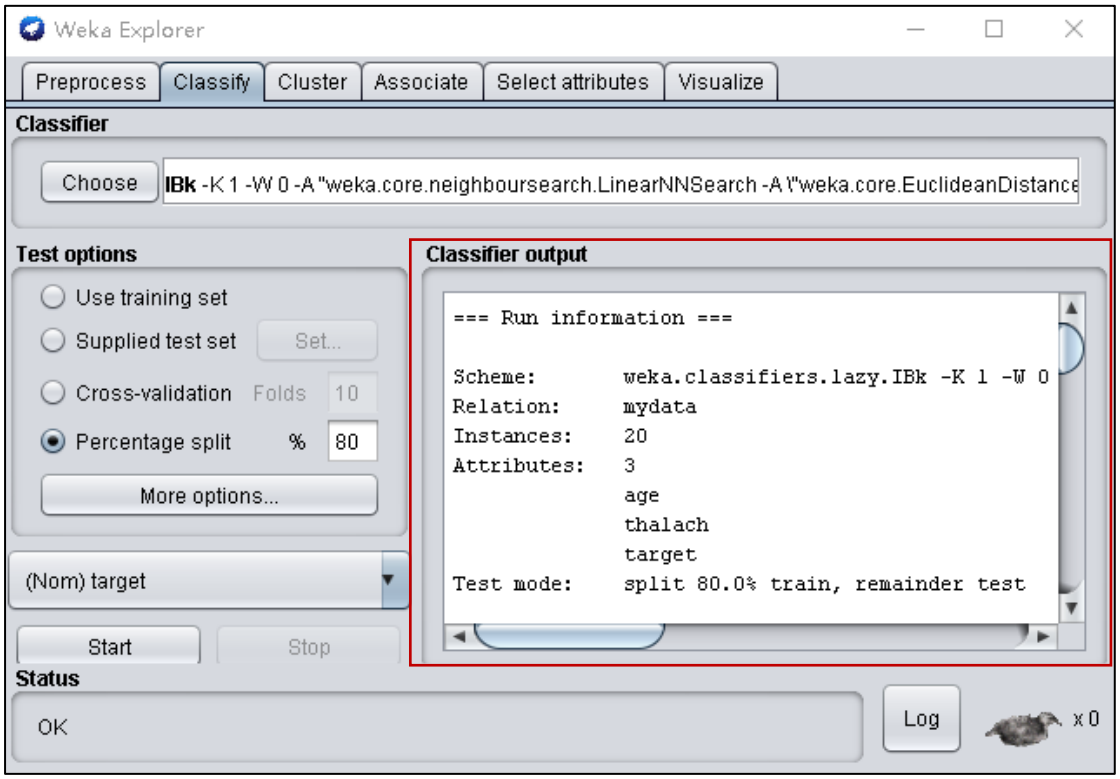
b. Choose "Preserve Order for % Split";



1.2.1.4 By default the third (categorical) attribute is set as the class (do NOT change it);



1.2.1.5 Run the classifier and copy the result/output for your documentation.



```
=== Run information ===
Scheme:       weka.classifiers.lazy.IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A
\"weka.core.EuclideanDistance -R first-last\""
Relation:     mydata
Instances:    20
Attributes:   3
              age
              thalach
              target
Test mode:    split 80.0% train, remainder test
=== Classifier model (full training set) ===
IB1 instance-based classifier
using 1 nearest neighbour(s) for classification
Time taken to build model: 0 seconds
=== Predictions on test split ===
    inst#     actual  predicted error prediction
        1     1:low     2:high     +     0.944
        2     2:high    2:high           0.944
        3     1:low     1:low            0.944
        4     2:high    2:high           0.944
=== Evaluation on test split ===
Time taken to test model on test split: 0 seconds
=== Summary ===
Correctly Classified Instances           3               75      %
Incorrectly Classified Instances         1               25      %
Kappa statistic                          0.5
Mean absolute error                      0.2778
Root mean squared error                  0.4747
Relative absolute error                 55.5556 %
Root relative squared error             94.9334 %
Total Number of Instances                4
=== Detailed Accuracy By Class ===

                 TP Rate   FP Rate   Precision   Recall  F-Measure   MCC      ROC Area  PRC Area  Class
                 0.500     0.000     1.000       0.500   0.667       0.577    0.750     0.750     low
                 1.000     0.500     0.667       1.000   0.800       0.577    0.750     0.667     high
Weighted Avg.    0.750     0.250     0.833       0.750   0.733       0.577    0.750     0.708

=== Confusion Matrix ===

 a b   <-- classified as
 1 1 | a = low
 0 2 | b = high
```

2. Manual plotting

I use python script with matplotlib package which is a comprehensive library for creating static, animated, and interactive visualizations to plot the figure.

2.1 Preparation

Firstly, I did some preparation with python script including loading packages which is need, loading data and initializing the figure.

2.1.1 Python code

```
from matplotlib.font_manager import FontProperties
import matplotlib.pyplot as plt
import pandas as pd
# Load data
data = pd.read_csv(r"data.csv")
train = data[:16]
test = data[16:]
# Init plot
fig,ax = plt.subplots()
```

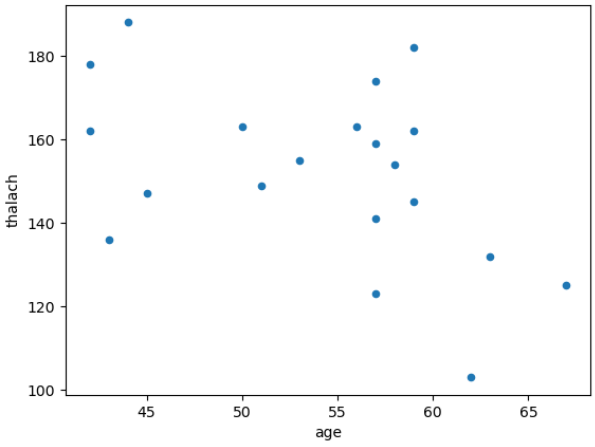
2.2 On a piece of paper, draw X and Y coordinates (two lines perpendicular to each other), with X representing the first numeric attribute and Y the second numeric attribute.

In the figure of 2nd step, X represents the age of each patient, the Y represent “thalach” which is the maximum heart rate achieved of each patient. There are 20 points in blue color representing the 20 instances of the dataset in this plot.

2.2.1 Python code

```
data.plot(x="age",y="thalach",kind="scatter",ax=ax)
plt.show()
```

2.2.2 Figure



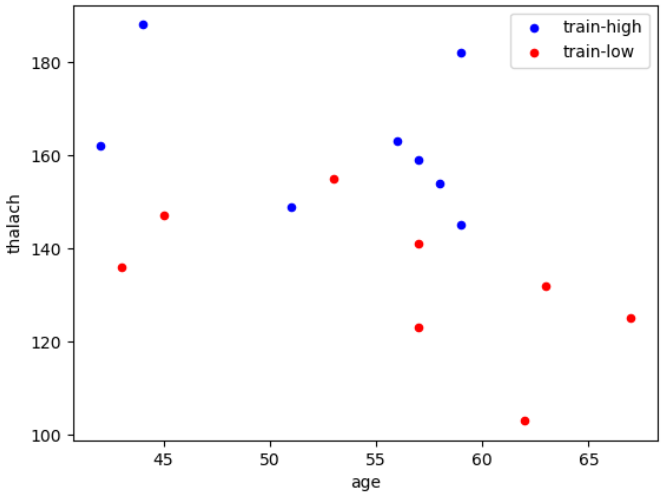
2.3 Mark the first 16 instances (training data) of your data on the paper, using color coding for the third (categorical) attribute. For example, if you have categories A and B then you can use a red dot for each instance in category A and a blue dot for category B.

In the figure of 3rd step, there are 16 points in blue color representing the 16 instances of the train dataset. The points with blue color represent the patients who have high chance of heart attack and the points with red color represent the patients who have high chance of heart.

2.3.1 Python code

```
ax = train[train["target"] ==
       "high"].plot(x="age",y="thalach",kind="scatter",c="blue",ax=ax)
ax = train[train["target"] ==
       "low"].plot(x="age",y="thalach",kind="scatter",c="red",ax=ax)
ax.legend(["train-high","train-low"])
plt.show()
```

2.3.2 Figure



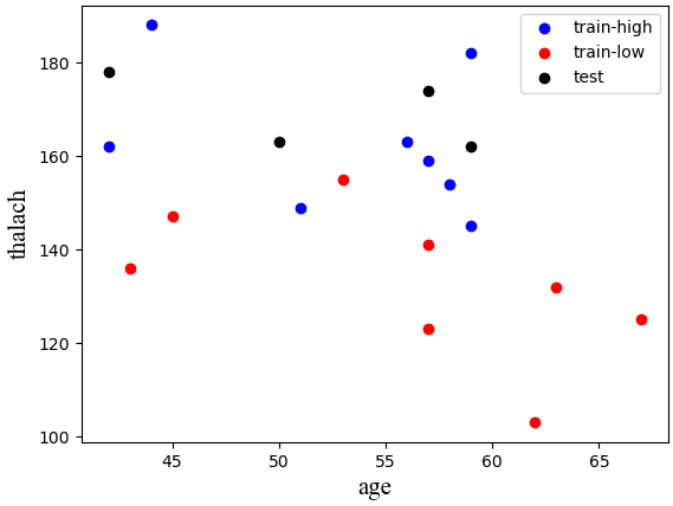
2.4 Mark the last 4 instances (testing data) of your data with a different color (e.g. black) on the paper.

In the figure of 4th step, the points with black color represent testing data.

2.4.1 Python code

```
ax = train[train["target"] ==
        "high"].plot(x="age",y="thalach",kind="scatter",c="blue",ax=ax)
ax = train[train["target"] ==
        "low"].plot(x="age",y="thalach",kind="scatter",c="red",ax=ax)
ax = test.plot(x="age",y="thalach",kind="scatter",c="black",ax=ax)
ax.legend(["train-high","train-low","test"])
plt.show()
```

2.4.2 Figure



3. Test and discussion

Manually examine each of the last 4 instances, identify its closest/nearest training data point (in the first 16 instances), and predict what category it will be put into using the nearest neighbor classification (consult textbook section 3.5). Include the plotting and discuss your observation/finding in the submission.

3.1 Standardization.

According to the lecture, the nearest neighbor classification algorithm is sensitive to the scales of data. Therefore, in order to eliminate the negative impact of the scales of data, I used the scikit-learn package which is third party modules commonly used in machine learning to standardize the dataset. The method I used to standardize is rescaling data to mean 0 with standard deviation 1:

$$x_{sc} = \frac{x - \mu}{\sigma}$$

3.1.1 Python code

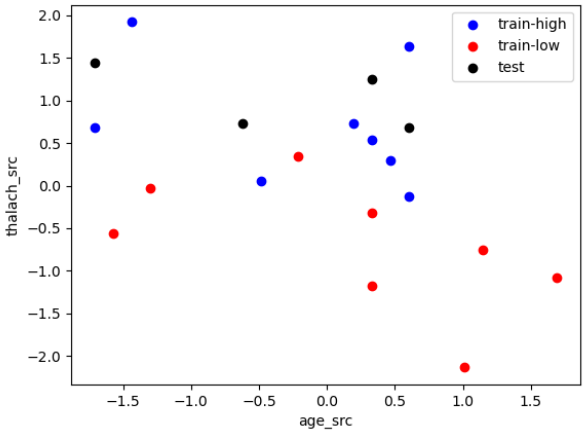
```
from sklearn.preprocessing import StandardScaler
x_train = data.loc[:16,:"thalach"]
y_train = data.loc[:16,:"target"]
x_test = data.loc[16:,:"thalach"]
y_test = data.loc[16:,:"target"]
sc = StandardScaler().fit(x_train)
x_train_sc = sc.transform(x_train)
x_test_sc = sc.transform(x_test)
```

3.2 Plot for the standardized dataset.

3.2.1 Python code

```
fig, ax = plt.subplots()
ax.set_xlabel("age_src")
ax.set_ylabel("thalach_src")
x_train_sc_high = x_train_sc[y_train=="high"]
ax.scatter(x=x_train_sc_high[:,0],y=x_train_sc_high[:,1],c="blue")
x_train_sc_low = x_train_sc[y_train=="low"]
ax.scatter(x=x_train_sc_low[:,0],y=x_train_sc_low[:,1],c="red")
ax.scatter(x=x_test_sc[:,0],y=x_test_sc[:,1],c="black")
ax.legend(["train-high","train-low","test"])
plt.show()
```

3.2.2 Figure

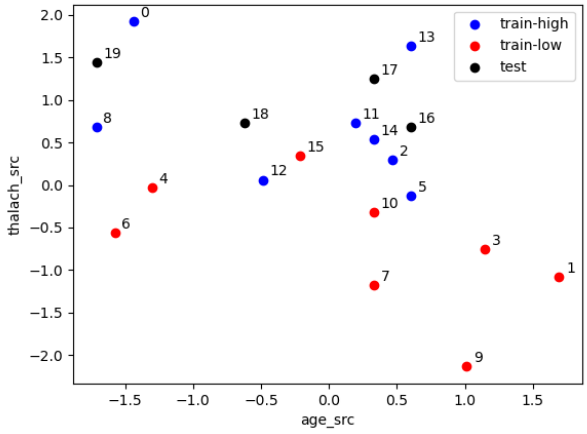


3.3 Assign Identify number to each instance for convenient.

3.3.1 Python code

```
for label in range(x_train_sc.shape[0]):
    ax.text(x = x_train_sc[label,0]+0.05,y =x_train_sc[label,1]+0.05,s = label)
for label in range(x_test_sc.shape[0]):
    ax.text(x = x_test_sc[label,0]+0.05,y =x_test_sc[label,1]+0.05,s =
            label+x_train_sc.shape[0])
plt.show()
```

3.3.2 Figure



3.4 Compute distance between each point.

In this step, in order to identify the closest/nearest training data point for each instance in testing dataset, I used python script to compute the distance between each data points and displayed the result as a table for convenient. The formula to compute the distance which I used to verify is “Euclidean Distance”:

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{n} \left(x_{i_k} - x_{j_k}\right)^2}$$

3.4.1 Python code

```
for test_id in range(x_test_sc.shape[0]):
    test_id += x_train_sc.shape[0]
    for train_id in range(x_train_sc.shape[0]):
        d = np.linalg.norm(x_test_sc[test_id-x_train_sc.shape[0]] -
              x_train_sc[train_id])
```

3.4.2 Result

ID	X	Y	X_SC	Y_SC	Neighbor ID	X	Y	X_SC	Y_SC	Distance
16	**59**	**162**	**0.60**	**0.68**	**14**	57	159	0.33	0.54	**0.31**
					2	58	154	0.47	0.30	0.41
					11	56	163	0.20	0.73	0.41
					5	59	145	0.60	-0.13	0.81
					15	53	155	-0.21	0.35	0.88
					13	59	182	0.60	1.64	0.95
					10	57	141	0.33	-0.32	1.04
					12	51	149	-0.49	0.06	1.25
					3	63	132	1.15	-0.75	1.53
					7	57	123	0.33	-1.18	1.88
					4	45	147	-1.30	-0.04	2.04
					1	67	125	1.69	-1.09	2.08
					8	42	162	-1.71	0.68	2.32
					0	44	188	-1.44	1.92	2.39
					6	43	136	-1.58	-0.56	2.51
					9	62	103	1.01	-2.14	2.85
17	**57**	**174**	**0.33**	**1.25**	**13**	59	182	0.60	1.64	**0.47**
					11	56	163	0.20	0.73	0.54
					14	57	159	0.33	0.54	0.72
					2	58	154	0.47	0.30	0.96

					15	53	155	-0.21	0.35	1.06
					5	59	145	0.60	-0.13	1.41
					12	51	149	-0.49	0.06	1.45
					10	57	141	0.33	-0.32	1.58
					0	44	188	-1.44	1.92	1.89
					4	45	147	-1.30	-0.04	2.08
					8	42	162	-1.71	0.68	2.12
					3	63	132	1.15	-0.75	2.17
					7	57	123	0.33	-1.18	2.43
					6	43	136	-1.58	-0.56	2.63
					1	67	125	1.69	-1.09	2.71
					9	62	103	1.01	-2.14	3.46
18	**50**	**163**	**-0.62**	**0.73**	**15**	53	155	-0.21	0.35	**0.56**
					12	51	149	-0.49	0.06	0.68
					11	56	163	0.20	0.73	0.82
					14	57	159	0.33	0.54	0.97
					4	45	147	-1.30	-0.04	1.02
					8	42	162	-1.71	0.68	1.09
					2	58	154	0.47	0.30	1.17
					10	57	141	0.33	-0.32	1.42
					0	44	188	-1.44	1.92	1.45
					5	59	145	0.60	-0.13	1.50
					13	59	182	0.60	1.64	1.53
					6	43	136	-1.58	-0.56	1.60
					7	57	123	0.33	-1.18	2.13
					3	63	132	1.15	-0.75	2.31
					1	67	125	1.69	-1.09	2.94
					9	62	103	1.01	-2.14	3.30
19	**42**	**178**	**-1.71**	**1.44**	**0**	44	188	-1.44	1.92	**0.55**
					8	42	162	-1.71	0.68	0.76
					4	45	147	-1.30	-0.04	1.54
					12	51	149	-0.49	0.06	1.85
					15	53	155	-0.21	0.35	1.86
					6	43	136	-1.58	-0.56	2.01
					11	56	163	0.20	0.73	2.04
					14	57	159	0.33	0.54	2.24
					13	59	182	0.60	1.64	2.32
					2	58	154	0.47	0.30	2.46

10	57	141	0.33	-0.32	2.70
5	59	145	0.60	-0.13	2.80
7	57	123	0.33	-1.18	3.33
3	63	132	1.15	-0.75	3.61
1	67	125	1.69	-1.09	4.24
9	62	103	1.01	-2.14	4.50

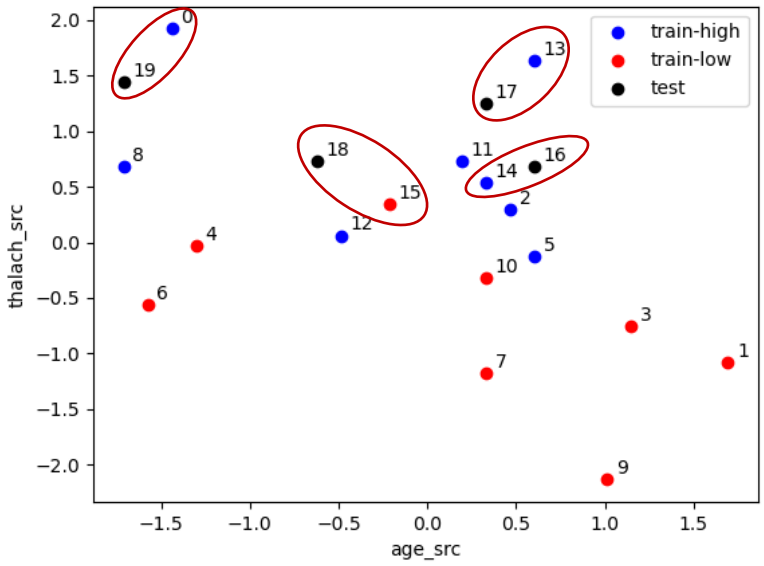
3.5 Identify the closest/nearest training data point of each instance in testing dataset.

In this step, I found the closest (the minimize distance) training data point of each instance in testing dataset through the result of step 3.2 and draw on the plot for a clearer view.

3.5.1 Result

ID	Neighbor ID	Distance
16	14	0.31
17	13	0.47
18	15	0.56
19	0	0.55

3.5.2 Figure



3.6 Predict the category for each instance in testing dataset will be put into using the nearest neighbor classification.

According to the nearest neighbor classification algorithm, each instance can be represented by the several (k) nearest neighbors. In this case, the number of neighbors to represent the instance is 1. Therefore, the predict category for each instance in testing dataset is equal to the category of their nearest neighbor's category.

3.6.1 Result

ID	Neighbor ID	Neighbor category	Predicted category
16	14	high	**high**
17	13	high	**high**
18	15	low	**low**
19	0	high	**high**

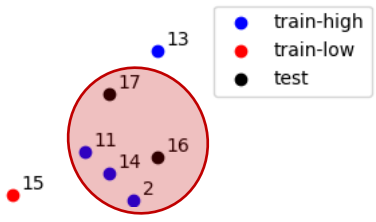
3.7 Verify the predict category for each instance in testing dataset with the real category and the result of Weka's algorithm.

3.7.1 Result

ID	Predicted category	Weka's category	Real category
16	**high**	**high**	**low**
17	high	high	high
18	low	low	low
19	high	high	high

3.7.2 Analyze

According to the result, for each test instance, the predicted categories generated by the above steps is equal to the categories generated by the Weka's IBk algorithm, which refers to that the above steps has high similarity with Weka's IBk algorithm. Both algorithms correctly predict the categories of all the test instances except that whose ID is 16. According to the figure, I think the reason that cause this error is all the instances near the "16" test instance is represent "the high chance of heart attack".



4. Consider test and compare with k=3

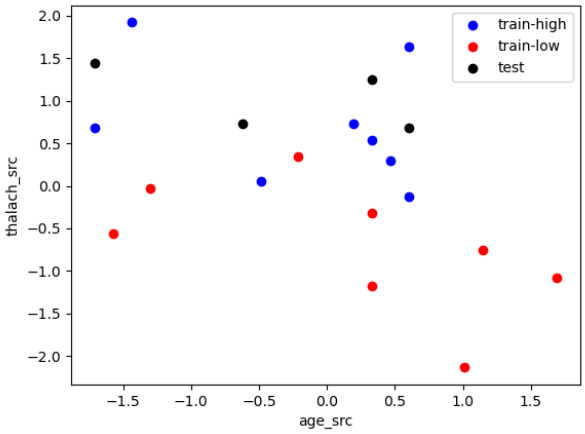
Manually test the same 4 instances but use 3 nearest neighbors in each case for the prediction. Compare to step 3 for any difference (how and why so).

4.1 Standardization.

Same as the step 3.

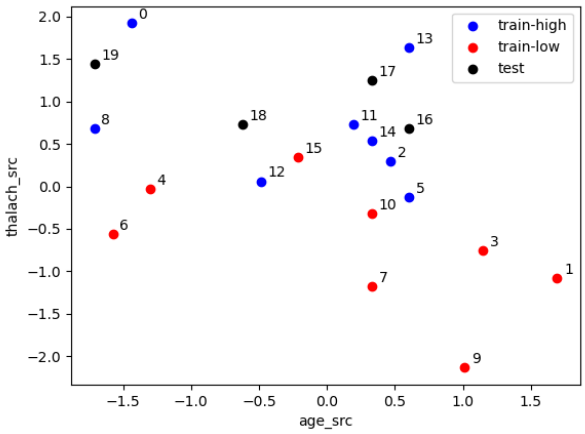
4.2 Plot for the standardized dataset.

Same as the step 3.



4.3 Assign Identify number to each instance for convenient.

Same as the step 3.



4.4 Compute distance between each point.

Same as the step 3.

ID	X	Y	X_SC	Y_SC	Neighbor ID	X	Y	X_SC	Y_SC	Distance
					14	57	159	0.33	0.54	**0.31**
					2	58	154	0.47	0.30	**0.41**
					11	56	163	0.20	0.73	**0.41**
					5	59	145	0.60	-0.13	0.81
					15	53	155	-0.21	0.35	0.88
					13	59	182	0.60	1.64	0.95
					10	57	141	0.33	-0.32	1.04
					12	51	149	-0.49	0.06	1.25
16	**59**	**162**	**0.60**	**0.68**	3	63	132	1.15	-0.75	1.53
					7	57	123	0.33	-1.18	1.88
					4	45	147	-1.30	-0.04	2.04
					1	67	125	1.69	-1.09	2.08
					8	42	162	-1.71	0.68	2.32
					0	44	188	-1.44	1.92	2.39
					6	43	136	-1.58	-0.56	2.51
					9	62	103	1.01	-2.14	2.85
					13	59	182	0.60	1.64	**0.47**
					11	56	163	0.20	0.73	**0.54**
					14	57	159	0.33	0.54	**0.72**
					2	58	154	0.47	0.30	0.96
					15	53	155	-0.21	0.35	1.06
					5	59	145	0.60	-0.13	1.41
					12	51	149	-0.49	0.06	1.45
					10	57	141	0.33	-0.32	1.58
17	**57**	**174**	**0.33**	**1.25**	0	44	188	-1.44	1.92	1.89
					4	45	147	-1.30	-0.04	2.08
					8	42	162	-1.71	0.68	2.12
					3	63	132	1.15	-0.75	2.17
					7	57	123	0.33	-1.18	2.43
					6	43	136	-1.58	-0.56	2.63
					1	67	125	1.69	-1.09	2.71
					9	62	103	1.01	-2.14	3.46
					15	53	155	-0.21	0.35	**0.56**
18	**50**	**163**	**-0.62**	**0.73**	**12**	51	149	-0.49	0.06	**0.68**
					11	56	163	0.20	0.73	**0.82**

					14	57	159	0.33	0.54	0.97
					4	45	147	-1.30	-0.04	1.02
					8	42	162	-1.71	0.68	1.09
					2	58	154	0.47	0.30	1.17
					10	57	141	0.33	-0.32	1.42
					0	44	188	-1.44	1.92	1.45
					5	59	145	0.60	-0.13	1.50
					13	59	182	0.60	1.64	1.53
					6	43	136	-1.58	-0.56	1.60
					7	57	123	0.33	-1.18	2.13
					3	63	132	1.15	-0.75	2.31
					1	67	125	1.69	-1.09	2.94
					9	62	103	1.01	-2.14	3.30
19	**42**	**178**	**-1.71**	**1.44**	**0**	44	188	-1.44	1.92	**0.55**
					8	42	162	-1.71	0.68	**0.76**
					4	45	147	-1.30	-0.04	**1.54**
					12	51	149	-0.49	0.06	1.85
					15	53	155	-0.21	0.35	1.86
					6	43	136	-1.58	-0.56	2.01
					11	56	163	0.20	0.73	2.04
					14	57	159	0.33	0.54	2.24
					13	59	182	0.60	1.64	2.32
					2	58	154	0.47	0.30	2.46
					10	57	141	0.33	-0.32	2.70
					5	59	145	0.60	-0.13	2.80
					7	57	123	0.33	-1.18	3.33
					3	63	132	1.15	-0.75	3.61
					1	67	125	1.69	-1.09	4.24
					9	62	103	1.01	-2.14	4.50

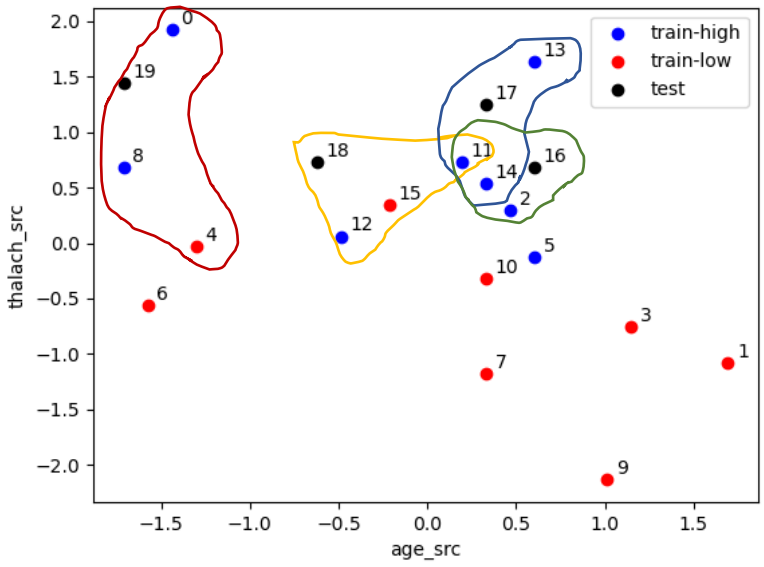
4.5 Identify the closest/nearest training data point of each instance in testing dataset.

Compare with the step 3, the difference in there is the number of closest/nearest training data point of each instance in testing dataset is 3, that is 3 nearest neighbors need to be found in this case.

4.5.1 Result

ID	Neighbor ID	Distance
16	14	0.31
	2	0.41
	11	0.41
17	13	0.47
	11	0.54
	14	0.72
18	15	0.56
	12	0.68
	11	0.82
19	0	0.55
	8	0.76
	4	1.54

4.5.2 Figure



4.6 Predict the category for each instance in testing dataset will be put into using the nearest neighbor classification.

Compare with the step 3, the difference is that there are three neighbors need to be considered rather than one neighbor. In this phase, I used the simplest method called "majority voting" to classify the category for each test instance by assigning the label which is most frequent among the three train instances nearest to that query point.

4.6.1 Result

ID	Neighbor ID	Neighbor category	Predicted category
16	14	high	**high**
	2	high	
	11	high	
17	13	high	**high**
	11	high	
	14	high	
18	15	low	**high**
	12	high	
	11	high	
19	0	high	**high**
	8	high	
	4	low	

4.7 Verify the predict category for each instance in testing dataset with the real category and the result of Weka's algorithm.

4.7.1 Result

ID	Predicted category	Real category
16	**high**	**low**
17	high	high
18	**high**	**low**
19	high	high

4.7.2 Analyze

Compare with the step 3, there are more errors when predicting the categories for test instances. For test instance whose ID is 16, I think the reason is as same as the step 3, that is the dataset is not suitable to this algorithm. For test instance whose ID is 18, I think the reason cause the error is the weakness of "majority voting" with simple weight mechanism which brings noisy into classification phase.