Chapter17Solution

November 29, 2022

1 Chapter 17 Textbook: Classification

Reading: * Classification

Please complete this notebook by filling in the cells provided.

Directly sharing answers is not okay, but discussing problems with the course staff or with other students is encouraged.

For all problems that you must write our explanations and sentences for, you must provide your answer in the designated space. Moreover, throughout this homework and all future ones, please be sure to not re-assign variables throughout the notebook! For example, if you use max_temperature in your answer to one question, do not reassign it later on.

[ ]:
```
# Don't change this cell; just run it.
import numpy as np
from datascience import *
```

[ ]:
```
# These lines do some fancy plotting magic.
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import warnings
warnings.simplefilter('ignore', FutureWarning)
```

[ ]:
```
def standard_units(arr):
    return (arr - np.average(arr))/np.std(arr)
```

[ ]:
```
ckd = Table.read_table('ckd.csv').relabeled('Blood Glucose Random', 'Glucose')
ckd.row(0)
```

[ ]:
```
ckd = Table().with_columns(
    'Hemoglobin', standard_units(ckd.column('Hemoglobin')),
    'Glucose', standard_units(ckd.column('Glucose')),
    'White Blood Cell Count', standard_units(ckd.column('White Blood Cell␣
 ↪Count')),
    'Class', ckd.column('Class')
)
```

```
ckd
```

[ ]:
```
color_table = Table().with_columns(
    'Class', make_array(1, 0),
    'Color', make_array('darkblue', 'gold')
)
ckd = ckd.join('Class', color_table)
```

[ ]:
```
ckd
```

[ ]:
```
ckd.scatter('White Blood Cell Count', 'Glucose', group='Color')
plt.xlim(-4, 2)
plt.ylim(-2, 4);
```

[ ]:
```
ckd.scatter('Hemoglobin', 'Glucose', group='Color')
plt.xlim(-4, 2)
plt.ylim(-2, 4);
```

[ ]:
```
shuffled_ckd = ckd.sample(with_replacement=False)
training = shuffled_ckd.take(np.arange(79))
testing = shuffled_ckd.take(np.arange(79, 158))
```

[ ]:
```
training
```

[ ]:
```
training.scatter('Hemoglobin', 'Glucose', group='Color')
plt.xlim(-4, 2)
plt.ylim(-2, 4);
```

[ ]:
```
testing.scatter('Hemoglobin', 'Glucose', group='Color')
plt.xlim(-4, 2)
plt.ylim(-2, 4);
```

[ ]:
```
alice = make_array(0, 1.1)

training.scatter('Hemoglobin', 'Glucose', group='Color')
plt.xlim(-4, 2)
plt.ylim(-2, 4)
plt.scatter(alice.item(0), alice.item(1), color='red', s=30);
```

[ ]:
```
training_set= training.select('Class','Hemoglobin','Glucose')
test_set = testing.select('Class','Hemoglobin','Glucose')
```

[ ]:
```
training_set
```

[ ]:
```
test_set
```

[ ]:
```
training_attributes = training.select('Hemoglobin', 'Glucose')
training_attributes
```

[ ]:
```
alice
```

[ ]:
```
patient3 = np.array(training_attributes.row(3))
patient3
```

[ ]:
```
def distance(point1, point2):
    """Returns the distance between point1 and point2
    where each argument is an array
    consisting of the coordinates of the point"""
    return np.sqrt(np.sum((point1 - point2)**2))

def all_distances(training, new_point):
    """Returns an array of distances
    between each point in the training set
    and the new point (which is a row of attributes)"""
    attributes = training.drop('Class')
    def distance_from_point(row):
        return distance(np.array(new_point), np.array(row))
    return attributes.apply(distance_from_point)

def table_with_distances(training, new_point):
    """Augments the training table
    with a column of distances from new_point"""
    return training.with_column('Distance', all_distances(training, new_point))

def closest(training, new_point, k):
    """Returns a table of the k rows of the augmented table
    corresponding to the k smallest distances"""
    with_dists = table_with_distances(training, new_point)
    sorted_by_distance = with_dists.sort('Distance')
    topk = sorted_by_distance.take(np.arange(k))
    return topk

def majority(topkclasses):
    ones = topkclasses.where('Class', are.equal_to(1)).num_rows
    zeros = topkclasses.where('Class', are.equal_to(0)).num_rows
    if ones > zeros:
        return 1
    else:
        return 0

def classify(training, new_point, k):
    closestk = closest(training, new_point, k)
    topkclasses = closestk.select('Class')
```

```
    return majority(topkclasses)
```

[ ]:
```
distance(alice,patient3)
```

[ ]:
```
closest(training_attributes, alice, 5)
```

[ ]:
```
closest(training_attributes, patient3, 5)
```

[ ]:
```
def count_zero(array):
    """Counts the number of 0's in an array"""
    return len(array) - np.count_nonzero(array)

def count_equal(array1, array2):
    """Takes two numerical arrays of equal length
    and counts the indices where the two are equal"""
    return count_zero(array1 - array2)

def evaluate_accuracy(training, test, k):
    test_attributes = test.drop('Class')
    def classify_testrow(row):
        return classify(training, row, k)
    c = test_attributes.apply(classify_testrow)
    return count_equal(c, test.column('Class')) / test.num_rows
```

[ ]:
```
evaluate_accuracy(training_set, test_set, 5)
```

[ ]:
```

```