**UJIAN TENGAH SEMESTER**

**Kecerdasan Buatan**

Nama : Paulus Miki Febrian

Nim : 3332190052

Mendownload materi yang diperlukan

!wget --no-check-certificate 'https://docs.google.com/uc?export=download&id=1zlJwPKAsKhlGlNu6Cn7XhKt\_IUFH87YF' -O data\_decision\_trees.txt

!wget --no-check-certificate 'https://docs.google.com/uc?export=download&id=1oNqpwqQarcuIavyZFHYRAtxBnShDOPIo' -O data\_clustering.txt

!wget --no-check-certificate 'https://docs.google.com/uc?export=download&id=13FHXPHZK3yRHhr0OMqz6YTM70HGxEeMa' -O data.txt

!wget --no-check-certificate 'https://docs.google.com/uc?export=download&id=13iqaBwvhTtyD7IX-5v5pSnAJQ0ouxGFT' -O coastal\_states.txt

!wget --no-check-certificate 'https://docs.google.com/uc?export=download&id=1AnbKOMXh\_3GB2Y9jTGkmF4bDXfpbgRAr' -O adjacent\_states.txt

Classifier Visualizer

def visualize\_classifier(classifier, X, y, title=''):

    # Define the minimum and maximum values for X and Y

    # that will be used in the mesh grid

    min\_x, max\_x = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0

    min\_y, max\_y = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0

    # Define the step size to use in plotting the mesh grid

    mesh\_step\_size = 0.01

    # Define the mesh grid of X and Y values

    x\_vals, y\_vals = np.meshgrid(np.arange(min\_x, max\_x, mesh\_step\_size), np.arange(min\_y, max\_y, mesh\_step\_size))

    # Run the classifier on the mesh grid

    output = classifier.predict(np.c\_[x\_vals.ravel(), y\_vals.ravel()])

    # Reshape the output array

    output = output.reshape(x\_vals.shape)

    # Create a plot

    plt.figure()

    # Specify the title

    plt.title(title)

    # Choose a color scheme for the plot

    plt.pcolormesh(x\_vals, y\_vals, output, cmap=plt.cm.gray)

    # Overlay the training points on the plot

    plt.scatter(X[:, 0], X[:, 1], c=y, s=75, edgecolors='black', linewidth=1, cmap=plt.cm.Paired)

    # Specify the boundaries of the plot

    plt.xlim(x\_vals.min(), x\_vals.max())

plt.ylim(y\_vals.min(), y\_vals.max())

    # Specify the ticks on the X and Y axes

    plt.xticks((np.arange(int(X[:, 0].min() - 1), int(X[:, 0].max() + 1), 1.0)))

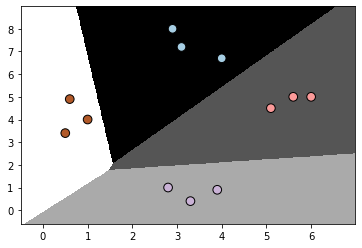
    plt.yticks((np.arange(int(X[:, 1].min() - 1), int(X[:, 1].max() + 1), 1.0)))

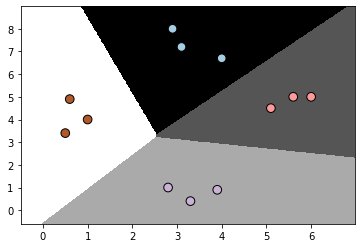
    plt.show()

Soal

1. Analisa algoritma untuk *logistic\_regression.py.* Dan analisa algoritmanya dan jalankan di komputer anda.

|  |
| --- |
| import numpy as np  from sklearn import linear\_model  import matplotlib.pyplot as plt  #from utilities import visualize\_classifier  # Define sample input data  X = np.array([[3.1, 7.2], [4, 6.7], [2.9, 8], [5.1, 4.5], [6, 5], [5.6, 5], [3.3, 0.4], [3.9, 0.9], [2.8, 1], [0.5, 3.4], [1, 4], [0.6, 4.9]])  y = np.array([0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3])  # Create the logistic regression classifier  classifier1 = linear\_model.LogisticRegression(solver='liblinear', C=1)  classifier2 = linear\_model.LogisticRegression(solver='liblinear', C=100)  # Train the classifier  classifier1.fit(X, y)  classifier2.fit(X, y)  # Visualize the performance of the classifier  visualize\_classifier(classifier1, X, y)  visualize\_classifier(classifier2, X, y) |

Hasilnya :

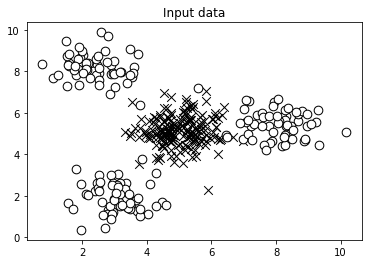


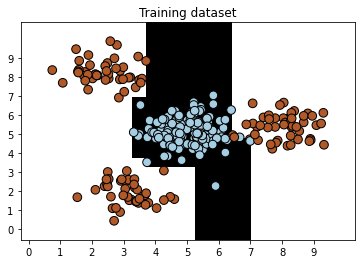
Gambar di atas merupakan grafik dari proses klasifikasi. Titik-titik di atas adalah variabel dependent dan area yang mengitari titik tersebut merupakan klasifikasi yang membedakan data-data tersebut.

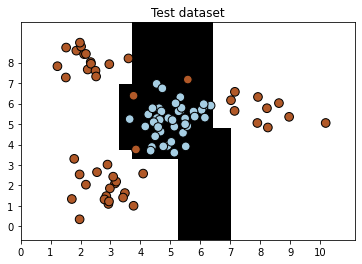
1. Analisa algoritma untuk *decision\_trees.py.* Dan analisa algoritmanya dan jalankan di komputer anda.

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from sklearn.metrics import classification\_report  #from sklearn import cross\_validation  from sklearn.tree import DecisionTreeClassifier  from sklearn.model\_selection import train\_test\_split  #from utilities import visualize\_classifier  # Load input data  input\_file = 'data\_decision\_trees.txt'  data = np.loadtxt(input\_file, delimiter=',')  X, y = data[:, :-1], data[:, -1]  # Separate input data into two classes based on labels  class\_0 = np.array(X[y==0])  class\_1 = np.array(X[y==1])  # Visualize input data  plt.figure()  plt.scatter(class\_0[:, 0], class\_0[:, 1], s=75, facecolors='black',  edgecolors='black', linewidth=1, marker='x')  plt.scatter(class\_1[:, 0], class\_1[:, 1], s=75, facecolors='white',  edgecolors='black', linewidth=1, marker='o')  plt.title('Input data')  # Split data into training and testing datasets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  X, y, test\_size=0.25, random\_state=5)  # Decision Trees classifier  params = {'random\_state': 0, 'max\_depth': 4}  classifier = DecisionTreeClassifier(\*\*params)  classifier.fit(X\_train, y\_train)  visualize\_classifier(classifier, X\_train, y\_train, 'Training dataset')  y\_test\_pred = classifier.predict(X\_test)  visualize\_classifier(classifier, X\_test, y\_test, 'Test dataset')  # Evaluate classifier performance  class\_names = ['Class-0', 'Class-1']  print("\n" + "#"\*40)  print("\nClassifier performance on training dataset\n")  print(classification\_report(y\_train, classifier.predict(X\_train), target\_names=class\_names))  print("#"\*40 + "\n")  print("#"\*40)  print("\nClassifier performance on test dataset\n")  print(classification\_report(y\_test, y\_test\_pred, target\_names=class\_names))  print("#"\*40 + "\n")  plt.show() |

Hasil:







########################################

Classifier performance on training dataset

precision recall f1-score support

Class-0 0.99 1.00 1.00 137

Class-1 1.00 0.99 1.00 133

accuracy 1.00 270

macro avg 1.00 1.00 1.00 270

weighted avg 1.00 1.00 1.00 270

########################################

########################################

Classifier performance on test dataset

precision recall f1-score support

Class-0 0.93 1.00 0.97 43

Class-1 1.00 0.94 0.97 47

accuracy 0.97 90

macro avg 0.97 0.97 0.97 90

weighted avg 0.97 0.97 0.97 90

########################################

1. Analisa algoritma untuk *mean\_shift.py.* Dan analisa algoritmanya dan jalankan di komputer anda.

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from sklearn.cluster import MeanShift, estimate\_bandwidth  from itertools import cycle  # Load data from input file  X = np.loadtxt('data\_clustering.txt', delimiter=',')  # Estimate the bandwidth of X  bandwidth\_X = estimate\_bandwidth(X, quantile=0.1, n\_samples=len(X))  # Cluster data with MeanShift  meanshift\_model = MeanShift(bandwidth=bandwidth\_X, bin\_seeding=True)  meanshift\_model.fit(X)  # Extract the centers of clusters  cluster\_centers = meanshift\_model.cluster\_centers\_  print('\nCenters of clusters:\n', cluster\_centers)  # Estimate the number of clusters  labels = meanshift\_model.labels\_  num\_clusters = len(np.unique(labels))  print("\nNumber of clusters in input data =", num\_clusters)  # Plot the points and cluster centers  plt.figure()  markers = 'o\*xvs'  for i, marker in zip(range(num\_clusters), markers):      # Plot points that belong to the current cluster      plt.scatter(X[labels==i, 0], X[labels==i, 1], marker=marker, color='black')      # Plot the cluster center      cluster\_center = cluster\_centers[i]      plt.plot(cluster\_center[0], cluster\_center[1], marker='o',              markerfacecolor='black', markeredgecolor='black',              markersize=15)  plt.title('Clusters')  plt.show() |

Hasil :

Centers of clusters:

[[2.95568966 1.95775862]

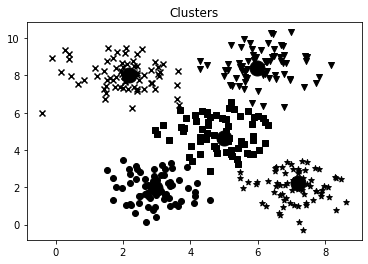
[7.20690909 2.20836364]

[2.17603774 8.03283019]

[5.97960784 8.39078431]

[4.99466667 4.65844444]]

Number of clusters in input data = 5



1. Analisa algoritma untuk *nearest\_neighbors\_classifier.py.* Dan analisa algoritmanya dan jalankan di komputer anda.

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  import matplotlib.cm as cm  from sklearn import neighbors, datasets  # Load input data  input\_file = 'data.txt'  data = np.loadtxt(input\_file, delimiter=',')  X, y = data[:, :-1], data[:, -1].astype(np.int)  # Plot input data  plt.figure()  plt.title('Input data')  marker\_shapes = 'v^os'  mapper = [marker\_shapes[i] for i in y]  for i in range(X.shape[0]):      plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],              s=75, edgecolors='black', facecolors='none')  # Number of nearest neighbors  num\_neighbors = 12  # Step size of the visualization grid  step\_size = 0.01  # Create a K Nearest Neighbours classifier model  classifier = neighbors.KNeighborsClassifier(num\_neighbors, weights='distance')  # Train the K Nearest Neighbours model  classifier.fit(X, y)  # Create the mesh to plot the boundaries  x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1  y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1  x\_values, y\_values = np.meshgrid(np.arange(x\_min, x\_max, step\_size),          np.arange(y\_min, y\_max, step\_size))  # Evaluate the classifier on all the points on the grid  output = classifier.predict(np.c\_[x\_values.ravel(), y\_values.ravel()])  # Visualize the predicted output  output = output.reshape(x\_values.shape)  plt.figure()  plt.pcolormesh(x\_values, y\_values, output, cmap=cm.Paired)  # Overlay the training points on the map  for i in range(X.shape[0]):      plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],              s=50, edgecolors='black', facecolors='none')  plt.xlim(x\_values.min(), x\_values.max())  plt.ylim(y\_values.min(), y\_values.max())  plt.title('K Nearest Neighbors classifier model boundaries')  # Test input datapoint  test\_datapoint = [5.1, 3.6]  plt.figure()  plt.title('Test datapoint')  for i in range(X.shape[0]):      plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],              s=75, edgecolors='black', facecolors='none')  plt.scatter(test\_datapoint[0], test\_datapoint[1], marker='x',          linewidth=6, s=200, facecolors='black')  # Extract the K nearest neighbors  \_, indices = classifier.kneighbors([test\_datapoint])  indices = indices.astype(np.int)[0]  # Plot k nearest neighbors  plt.figure()  plt.title('K Nearest Neighbors')  for i in indices:      plt.scatter(X[i, 0], X[i, 1], marker=mapper[y[i]],              linewidth=3, s=100, facecolors='black')  plt.scatter(test\_datapoint[0], test\_datapoint[1], marker='x',          linewidth=6, s=200, facecolors='black')  for i in range(X.shape[0]):      plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],              s=75, edgecolors='black', facecolors='none')  print("Predicted output:", classifier.predict([test\_datapoint])[0])  plt.show() |

Hasil :

/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:9: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.

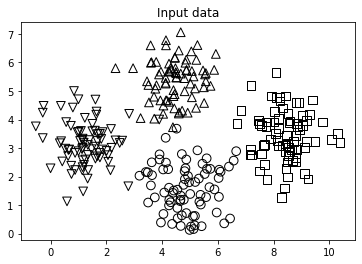
Deprecated in NumPy 1.20; for more details and guidance: <https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations>

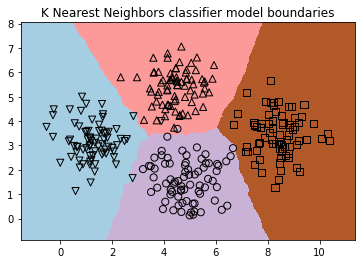
if \_\_name\_\_ == '\_\_main\_\_':

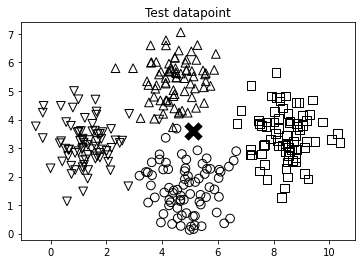
/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:68: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.

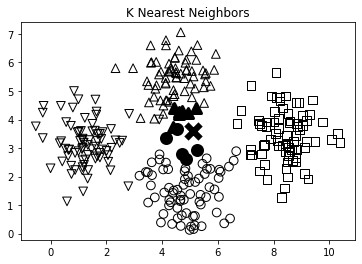
Deprecated in NumPy 1.20; for more details and guidance: <https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations>

Predicted output: 1









1. Analisa algoritma untuk *states.py.* Dan analisa algoritmanya dan jalankan di komputer anda.

|  |
| --- |
| !pip install 'git+https://github.com/MHordecki/LogPy#egg=logPy'  !pip install logic |

|  |
| --- |
| from logpy import run, fact, eq, Relation, var  adjacent = Relation()  coastal = Relation()  file\_coastal = 'coastal\_states.txt'  file\_adjacent = 'adjacent\_states.txt'  # Read the file containing the coastal states  with open(file\_coastal, 'r') as f:      line = f.read()      coastal\_states = line.split(',')  # Add the info to the fact base  for state in coastal\_states:      fact(coastal, state)  # Read the file containing the coastal states  with open(file\_adjacent, 'r') as f:      adjlist = [line.strip().split(',') for line in f if line and line[0].isalpha()]  # Add the info to the fact base  for L in adjlist:      head, tail = L[0], L[1:]      for state in tail:          fact(adjacent, head, state)  # Initialize the variables  x = var()  y = var()  # Is Nevada adjacent to Louisiana?  output = run(0, x, adjacent('Nevada', 'Louisiana'))  print('\nIs Nevada adjacent to Louisiana?:')  print('Yes' if len(output) else 'No')  # States adjacent to Oregon  output = run(0, x, adjacent('Oregon', x))  print('\nList of states adjacent to Oregon:')  for item in output:      print(item)  # States adjacent to Mississippi that are coastal  output = run(0, x, adjacent('Mississippi', x), coastal(x))  print('\nList of coastal states adjacent to Mississippi:')  for item in output:      print(item)  # List of 'n' states that border a coastal state  n = 7  output = run(n, x, coastal(y), adjacent(x, y))  print('\nList of ' + str(n) + ' states that border a coastal state:')  for item in output:      print(item)  # List of states that adjacent to the two given states  output = run(0, x, adjacent('Arkansas', x), adjacent('Kentucky', x))  print('\nList of states that are adjacent to Arkansas and Kentucky:')  for item in output:      print(item) |

Hasil

Is Nevada adjacent to Louisiana?:

No

List of states adjacent to Oregon:

Idaho

Washington

Nevada

California

List of coastal states adjacent to Mississippi:

Louisiana

Alabama

List of 7 states that border a coastal state:

Maryland

Indiana

Texas

California

New Jersey

North Carolina

Wisconsin

List of states that are adjacent to Arkansas and Kentucky:

Tennessee

Missouri