|  |  |  |
| --- | --- | --- |
| Candidate elimination algorithm  def learn(concepts, target):      specific\_h = concepts[0].copy()      general\_h = [["?" for \_ in range(len(specific\_h))] for \_ in range(len(specific\_h))]      for i, h in enumerate(concepts):          if target[i] == "yes":              for x in range(len(specific\_h)):                  if h[x] != specific\_h[x]:                      specific\_h[x] = '?' general\_h[x][x] = '?'          elif target[i] == "no":              for x in range(len(specific\_h)):                  if h[x] != specific\_h[x]:                      general\_h[x][x] = specific\_h[x]                  else: general\_h[x][x] = '?'      general\_h = [h for h in general\_h if h != ['?' for \_ in range(len(specific\_h))]]      return specific\_h, general\_h  concepts = [      ['sunny', 'warm', 'normal', 'strong', 'warm', 'same'],      ['sunny', 'warm', 'high', 'strong', 'warm', 'same'],      ['rainy', 'cold', 'high', 'strong', 'warm', 'change'],      ['sunny', 'warm', 'high', 'strong', 'cool', 'change']  ] target = ["yes", "yes", "no", "yes"]  s\_final, g\_final = learn(concepts, target)  print("Final Specific Hypothesis:") print("\n",s\_final)  print("\nFinal General Hypothesis:")  for hypothesis in g\_final: print("\n",hypothesis)  Logistic regression  import pandas as pd from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LogisticRegression  from sklearn import metrics matplotlib.pyplot as plt  col\_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']  pima = pd.read\_csv("/content/drive/MyDrive/pima-indians-diabetes.csv", header=None, names=col\_names) pima.head()  feature\_cols = ['pregnant', 'insulin', 'bmi', 'age', 'glucose', 'bp', 'pedigree']  X = pima[feature\_cols]y = pima.label  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=16)  logreg = LogisticRegression(random\_state=16)  logreg.fit(X\_train, y\_train)  y\_pred = logreg.predict(X\_test)  cnf\_matrix = metrics.confusion\_matrix(y\_test, y\_pred)  print("Confusion Matrix:")  print(cnf\_matrix)  print("\nClassification Report:")  target\_names = ['without diabetes', 'with diabetes']  print(metrics.classification\_report(y\_test, y\_pred, target\_names=target\_names))  y\_pred\_proba = ogreg.predict\_proba(X\_test)[:,1]  fpr, tpr, \_ = metrics.roc\_curve(y\_test, y\_pred\_proba)  auc = metrics.roc\_auc\_score(y\_test, y\_pred\_proba) | Decision tree id3  from sklearn.tree import DecisionTreeClassifier, plot\_tree  import matplotlib.pyplot as plt import math  data = pd.read\_csv('/content/drive/MyDrive/id3.csv')  def calculate\_entropy(data, target\_column): total\_rows = len(data)    value\_counts = data[target\_column].value\_counts() entropy\_outcome =0      for value in value\_counts:          proportion = value / total\_rows          entropy\_outcome += -proportion \* math.log2(proportion)      return entropy\_outcome  initial\_entropy = calculate\_entropy(data, 'Answer')  print("Entropy of the dataset: ", initial\_entropy)  def calculate\_information\_gain(data, feature, target\_column): total\_rows = len(data) value\_counts = data[feature].value\_counts()      weighted\_entropy = sum((value\_counts[value] / total\_rows) \* calculate\_entropy(data[data[feature] == value], target\_column) for value in value\_counts.index) information\_gain = initial\_entropy - weighted\_entropy return information\_gain  best\_attribute = None highest\_information\_gain = -1  for column in data.columns[:-1]:  # Exclude the target column      information\_gain = calculate\_information\_gain(data, column, 'Answer')      if information\_gain > highest\_information\_gain:          highest\_information\_gain = information\_gain          best\_attribute = column  print("Best Attribute: ", best\_attribute, "(Information Gain: ", highest\_information\_gain, ")")  X = data.drop(columns='Answer') y = data['Answer']  X\_encoded = pd.get\_dummies(X)  y\_encoded = y.astype('category').cat.codes  clf = DecisionTreeClassifier(criterion='entropy', random\_state=0)  clf.fit(X\_encoded, y\_encoded) plt.figure(figsize=(20,10))  plot\_tree(clf, feature\_names=X\_encoded.columns, class\_names=y.astype('category').cat.categories, filled=True, rounded=True) plt.show()  Em algorithm  from sklearn.cluster import KMeans from sklearn import preprocessing  from sklearn.mixture import GaussianMixture  from sklearn.datasets import load\_iris import sklearn.metrics as sm  import pandas as pd import numpy as np import matplotlib.pyplot as plt  dataset = load\_iris() X = pd.DataFrame(dataset.data)  X.columns = ['Sepal\_Length', 'Sepal\_Width', 'Petal\_Length', 'Petal\_Width']  y = pd.DataFrame(dataset.target) y.columns = ['Targets'] print(X)  plt.figure(figsize=(14, 7)) colormap = np.array(['red', 'lime', 'black'])  plt.subplot(1, 3, 1)  plt.scatter(X['Petal\_Length'], X['Petal\_Width'], c=colormap[y['Targets']], s=40) plt.title('Real') plt.subplot(1, 3, 2) model = KMeans(n\_clusters=3) model.fit(X)  predY = np.choose(model.labels\_, [0, 1, 2]).astype(np.int64)  plt.scatter(X['Petal\_Length'], X['Petal\_Width'], c=colormap[predY], s=40)  plt.title('KMeans') scaler = preprocessing.StandardScaler() scaler.fit(X)  xsa = scaler.transform(X) xs = pd.DataFrame(xsa, columns=X.columns)  gmm = GaussianMixture(n\_components=3) gmm.fit(xs)  y\_cluster\_gmm = gmm.predict(xs) plt.subplot(1, 3, 3)  plt.scatter(X['Petal\_Length'], X['Petal\_Width'], c=colormap[y\_cluster\_gmm], s=40) plt.title('GMM Classification') | Naïve bayes  import csv, random, math def load\_csv(filename):  with open(filename, "r") as file:  return [list(map(float, line)) for line in csv.reader(file)]  def split\_dataset(dataset, split\_ratio):  train\_size = int(len(dataset) \* split\_ratio)  train\_set = random.sample(dataset, train\_size)  test\_set = [row for row in dataset if row not in train\_set]  return train\_set, test\_set  def separate\_by\_class(dataset): separated = {}  for vector in dataset:  class\_value = vector[-1] if class\_value not in separated:  separated[class\_value] = []  separated[class\_value].append(vector) return separated  def mean(numbers): return sum(numbers) / float(len(numbers))  def stdev(numbers): avg = mean(numbers)  variance = sum([(x - avg) \*\* 2 for x in numbers]) / float(len(numbers) - 1) return math.sqrt(variance)  def summarize(dataset):  summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(\*dataset)] del summaries[-1] return summaries  def summarize\_by\_class(dataset):  separated = separate\_by\_class(dataset)  return {class\_value: summarize(instances) for class\_value, instances in separated.items()}  def calculate\_probability(x, mean, stdev):  exponent = math.exp(-((x - mean) \*\* 2 / (2 \* stdev \*\* 2)))  return (1 / (math.sqrt(2 \* math.pi) \* stdev)) \* exponent  def calculate\_class\_probabilities(summaries, input\_vector):  probabilities = {}  for class\_value, class\_summaries in summaries.items():  probabilities[class\_value] = 1  for i in range(len(class\_summaries)): mean, stdev = class\_summaries[i] probabilities[class\_value] \*= calculate\_probability(input\_vector[i], mean, stdev) return probabilities  def predict(summaries, input\_vector):  probabilities = calculate\_class\_probabilities(summaries, input\_vector)  return max(probabilities, key=probabilities.get)  def get\_predictions(summaries, test\_set):  return [predict(summaries, row) for row in test\_set]  def get\_accuracy(test\_set, predictions):  correct = sum(1 for i in range(len(test\_set)) if test\_set[i][-1] == predictions[i])  return (correct / float(len(test\_set))) \* 100.0  def main():  filename = '/content/drive/MyDrive/Colab Notebooks/naivedata.csv'  split\_ratio = 0.67 dataset = load\_csv(filename)  train\_set, test\_set = split\_dataset(dataset, split\_ratio)  print(f'Split {len(dataset)} rows into \ntrain = {len(train\_set)} and \ntest = {len(test\_set)} rows') summaries = summarize\_by\_class(train\_set)  predictions = get\_predictions(summaries, test\_set)  accuracy = get\_accuracy(test\_set, predictions)  print(f'\nAccuracy of the classifier = {round(accuracy, 2)}%') main() |
|  | #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/wait.h>  #include <fcntl.h>  int main() {  int pid;  pid = fork();  if (pid < 0) {  printf("fork failed");  exit(0);  } else if (pid == 0) {  int fd;  fd = open("T2.txt", O\_CREAT | O\_TRUNC, 0777);  close(fd);  } else {  wait(NULL);  int fp, n;  char buffer[100];  fp = open("T2.txt", O\_RDWR, 0777);  n = read(0, buffer, 100);  write(fp, buffer, n);  write(1, buffer, n);  close(fp);  }  return 0;  } |  |
| Naïve bayes text classification  import pandas as pd from sklearn.model\_selection import train\_test\_split  from sklearn.feature\_extraction.text import CountVectorizer  from sklearn.naive\_bayes import MultinomialNB  from sklearn import metrics  msg = pd.read\_csv('/content/drive/MyDrive/naivetext.csv', names=['message', 'label'])  print('The dimensions of the dataset', msg.shape)  msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0}) X = msg.message  y = msg.labelnum print(X) print(y)  xtrain, xtest, ytrain, ytest = train\_test\_split(X, y)  print('\n The total number of Training Data:', ytrain.shape)  print('\n The total number of Test Data:', ytest.shape)  count\_vect = CountVectorizer() print('\n The words or Tokens in the text documents \n') xtrain\_dtm = count\_vect.fit\_transform(xtrain)  xtest\_dtm = count\_vect.transform(xtest)  print(count\_vect.get\_feature\_names\_out())  df = pd.DataFrame(xtrain\_dtm.toarray(), columns=count\_vect.get\_feature\_names\_out())  clf = MultinomialNB().fit(xtrain\_dtm, ytrain)  predicted = clf.predict(xtest\_dtm)  print('\n Accuracy of the classifier is', metrics.accuracy\_score(ytest, predicted)) print('\n Confusion matrix')  print(metrics.confusion\_matrix(ytest, predicted))  print('\n The value of Precision', metrics.precision\_score(ytest, predicted))  print('\n The value of Recall', metrics.recall\_score(ytest, predicted))  K-nearest neighbour  from sklearn.model\_selection import train\_test\_split  from sklearn.neighbors import KNeighborsClassifier  from sklearn.metrics import classification\_report, confusion\_matrix  import pandas as pd  df = pd.read\_csv('/content/drive/MyDrive/Iris.csv')  x = df[['SepalLengthCm', 'SepalWidthCm']].values  data = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}  y = df['Species'].map(data).values  x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3)  print('class: 0-Iris-Setosa, 1-Iris-Versicolor, 2-Iris-Virginica')  print(y)  x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3)  classifier = KNeighborsClassifier(n\_neighbors=4)  classifier.fit(x\_train, y\_train)  y\_pred = classifier.predict(x\_test)  print('Confusion Matrix')  print(confusion\_matrix(y\_test, y\_pred))  print('Accuracy Metrics')  print(classification\_report(y\_test, y\_pred))  sepal\_length = float(input('Enter sepal\_length: '))  sepal\_width = float(input('Enter sepal\_width: '))  test\_data = [[sepal\_length, sepal\_width]]  predicted\_species = classifier.predict(test\_data)  pred\_sp = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'][predicted\_species[0]]  print(f'For sepal length {sepal\_length} and sepal width {sepal\_width}, the predicted species is: {pred\_sp}') | Bayesian clasiifier  from pgmpy.estimators import MaximumLikelihoodEstimator  from pgmpy.models import BayesianNetwork  from pgmpy.inference import VariableElimination  corona\_data = pd.read\_csv('/content/drive/MyDrive/covid\_data\_2020-2021.csv')  corona\_data.replace('?', np.nan, inplace=True)  print('Sample instances from the dataset are given below')  print(corona\_data.head())  print('\nAttributes and datatypes')  print(corona\_data.dtypes)  model = BayesianNetwork([      ('cough', 'corona\_result'),      ('fever', 'corona\_result'),      ('sore\_throat', 'corona\_result'),      ('shortness\_of\_breath', 'corona\_result'),      ('head\_ache', 'corona\_result'),      ('age\_60\_and\_above', 'corona\_result'),      ('gender', 'corona\_result'),      ('test\_indication', 'corona\_result')  ])  print('\nLearning CPD using Maximum likelihood estimators')  model.fit(corona\_data, estimator=MaximumLikelihoodEstimator)  print('\nInferencing with Bayesian Network:')  corona\_infer = VariableElimination(model)  print('\n1. Probability of Corona given evidence for cough')  q1 = corona\_infer.query(variables=['corona\_result'], evidence={'cough':1})  print(q1)  print('\n2. Probability of Corona given evidence for fever')  q2 = corona\_infer.query(variables=['corona\_result'], evidence={'fever': 1})  print(q2)  GVM  import matplotlib.pyplot as plt  from sklearn import datasets  from sklearn.model\_selection import train\_test\_split  from sklearn import svm  from sklearn import metrics  cancer\_data = datasets.load\_breast\_cancer()  X\_train, X\_test, y\_train, y\_test = train\_test\_split(      cancer\_data.data, cancer\_data.target, test\_size=0.4, random\_state=109 )  cls = svm.SVC(kernel="linear")  cls.fit(X\_train, y\_train)  pred = cls.predict(X\_test)  print("Accuracy:", metrics.accuracy\_score(y\_test, pred))  print("Precision:", metrics.precision\_score(y\_test, pred))  print("Recall:", metrics.recall\_score(y\_test, pred))  print(metrics.classification\_report(y\_test, pred))  digits = datasets.load\_digits()  clf = svm.SVC(gamma=0.001, C=100)  X\_digits, y\_digits = digits.data[:-10], digits.target[:-10]  clf.fit(X\_digits, y\_digits)  digits\_predictions = clf.predict(digits.data[:-10])  plt.imshow(digits.images[6], interpolation='nearest') plt.show()  print("Predicted digit for the example image:", digits\_predictions[6]) | Locally weighted regression  from bokeh.plotting import figure, show, output\_notebook  from bokeh.layouts import gridplot  def local\_regression(x0, X, Y, tau):      x0 = np.r\_[1, x0] X = np.c\_[np.ones(len(X)), X]      weights = radial\_kernel(x0, X, tau) XW = X.T \* weights      beta = np.linalg.pinv(XW @ X) @ XW @ Y return x0 @ beta  def radial\_kernel(x0, X, tau):      return np.exp(np.sum((X - x0) \*\* 2, axis=1) / (-2 \* tau \*\* 2))  n = 1000 X = np.linspace(-3, 3, num=n)  Y = np.log(np.abs(X \*\* 2 - 1) + 5)  X += np.random.normal(scale=0.1, size=n)  domain = np.linspace(-3, 3, num=300)  def plot\_lwr(tau):      prediction = [local\_regression(x0, X, Y, tau) for x0 in domain]      plot = figure(width=400, height=400, title=f'tau={tau}')      plot.scatter(X, Y, alpha=0.5, legend\_label='Data')      plot.line(domain, prediction, line\_width=2, color='red', legend\_label='Fitted Curve')      plot.legend.location = "top\_left" return plot  output\_notebook()  show(gridplot([[plot\_lwr(10.), plot\_lwr(1.)], [plot\_lwr(0.1), plot\_lwr(0.01)]]))  Back propogation  X = np.array([[2, 9], [1, 5], [3, 6]], dtype=float) Y = Y / 100  Y = np.array([[92], [86], [89]], dtype=float) X = X / np.amax(X, axis=0)  def sigmoid(x): return 1 / (1 + np.exp(-x))  def derivatives\_sigmoid(x): return x \* (1 - x)  epoch = 5000 lr = 0.1 inputlayer\_neurons = 2 hiddenlayer\_neurons = 3  output\_neurons = 1  wh = np.random.uniform(size=(inputlayer\_neurons, hiddenlayer\_neurons))  bh = np.random.uniform(size=(1, hiddenlayer\_neurons))  wout = np.random.uniform(size=(hiddenlayer\_neurons, output\_neurons))  bout = np.random.uniform(size=(1, output\_neurons))  for i in range(epoch):  hinp1 = np.dot(X, wh) + bh hlayer\_act = sigmoid(hinp1)  outinp1 = np.dot(hlayer\_act, wout) + bout  output = sigmoid(outinp1) EO = Y - output  outgrad = derivatives\_sigmoid(output) d\_output = EO \* outgrad  EH = d\_output.dot(wout.T)  hiddengrad = derivatives\_sigmoid(hlayer\_act)  d\_hiddenlayer = EH \* hiddengrad  wout += hlayer\_act.T.dot(d\_output) \* lr  wh += X.T.dot(d\_hiddenlayer) \* lr predicted\_output = output \* 100  accuracy = np.mean(np.abs(predicted\_output - Y)) \* 100  print("Input: \n" + str(X)) print("Actual Output: \n" + str(Y))  print("Predicted Output: \n" + str(output)) |
|  | PROGRAM -SENDER:  #include <stdio.h>  #include <sys/shm.h>  #include <sys/ipc.h>  #define size 32  int main()  {  int shmid; char \*str[100],\*str;  printf("\nipc message passing using shared memory sender:");  shmid=shmget(60,size,IPC\_CREAT|0666);  str = shmat(shmid,0,0);  printf("enter the message to be sent:");  gets(s);  strcpy(str,s);  print("\nyour message has been sent");  printf("your message has been sent");  }  RECEIVER:  #include <stdio.h>  #include <sys/shm.h>  #include <sys/ipc.h>  #define size 32  int main()  {  printf("nipc message passing using shared memory-receiver");  int shmid;  char \*str;  shmid=shmget(60,size,IPC\_CREAT|0666);  str=shmat(shmid,0,0);  printf("received message is:");  puts(str);  return 0;  } |  |