CS 760: Machine Learning - Fall 2020

Homework 5: Nearest Neighbors & Naive Bayes

Due : 11/12/2020

Zijie Zhang

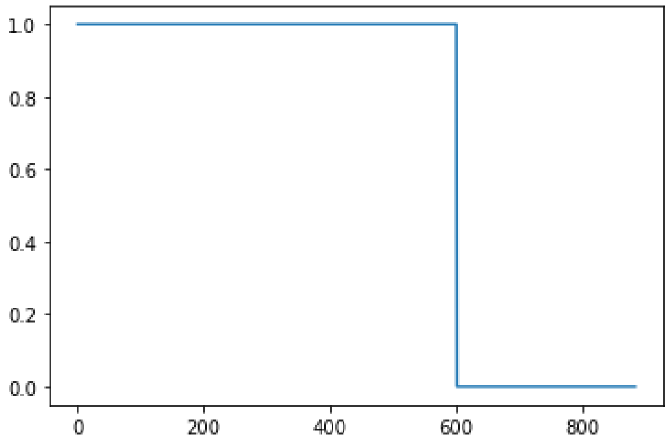
November 11, 2020

Problem 1

(a) I chose the KNN variant that,
weights = 'uniform', metric = 'euclidean'
The data should be min-max normalized before computing. The Accuracy is 81%.
KNN.py = https://github.com/z-zijie/2020Fall/blob/master/COMP760/Homework5/KNN.py

(b) Euclidean distance. Because this metric is more intuitive.

(c)



(d) The best choice of K is 5.

(e) We can use the voting ratio to decide reliability.

Problem 2

(a) **Bayes.py** = https://github.com/z-zijie/2020Fall/blob/master/COMP760/Homework5/Bayes.py

(b) Passenger Class, Gender, Siblings/Spouses and Parents/Children are Bernoulli.
Age and Fare are Gaussian.

(c) x = [1, 1, 22, 1, 0, 71.2833], survived.

(d) If prediction=survived, confidence = $P(\text{survived}|x)/P(\text{survived}|x) + P(\text{NOT survived}|x)$.

Problem 3

I prefer Random Forest, because this algorithm is very interpretable.

KNN

```
import numpy as np
from sklearn.model_selection import train_test_split

# import data
data = np.delete(np.genfromtxt('titanic_data.csv', delimiter=','), 0, 0)
X = data[:,1:]
y_label = data[:,0]

# transform
for i in range(X.shape[1]):
    if i == 1:
        continue
    X[:,i] = (X[:,i] - np.min(X[:,i]))/(np.max(X[:,i]) - np.min(X[:,i]))
#     X[:,i] = (X[:,i] - np.mean(X[:,i])) / np.std(X[:,i])

# metric = 'manhattan'
def dis(x,y):
    ans = np.sum(np.abs(y-x))
    if ans == 0:
        return 1e-10
    return ans
# metric = 'euclidean'
def l2(x,y):
    ans = np.linalg.norm(x-y)
    if ans == 0:
        return 1e-10
    return ans

# KNN
import heapq
def knn_predict(k, x, dataset, label):
    N = len(dataset)
    h = []
    for i in range(N):
        distance = l2(x, dataset[i])
        heapq.heappush(h, (distance, i))
    ans = [heapq.heappop(h)[1] for i in range(min(k,N))]
    p = sum([label[i] for i in ans])/len(ans)
    if (p >= 0.5):
        return 1
    return 0
i = 13
k = 5


# KFold
ans = np.zeros(len(X))
k = 5
from sklearn.model_selection import KFold
kf = KFold(n_splits = 10, shuffle = True)
kf.get_n_splits(X)
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y_label[train_index], y_label[test_index]
    for i in range(len(test_index)):
        ans[test_index[i]] = (knn_predict(k, X_test[i], X_train, y_train) == y_test[i])
accuracy = sum(ans)/len(ans)
print(accuracy)

# personal data
data = np.delete(np.genfromtxt('titanic_data.csv', delimiter=','), 0, 0)
x = np.array([1, 1, 22, 1, 0, 71.2833])
xx = data[:,1:]
# transform
for i in range(len(x)):
    if i == 1:
        continue
    x[i] = (x[i] - np.min(xx[:,i]))/(np.max(xx[:,i]) - np.min(xx[:,i]))
```

```
# predict
p = [knn_predict(k, x, X, y_label) for k in range(1,X.shape[0])]
import matplotlib.pyplot as plt
plt.plot(p)
```

Bayes

```
import numpy as np
from sklearn.model_selection import train_test_split
# import data
data = np.delete(np.genfromtxt('titanic_data.csv', delimiter=','), 0, 0)
X = data[:,1:]
y_label = data[:,0]

import math
def gaussian(val, mean, var):
    ans = math.exp(-(val-mean)*(val-mean)/(2*var))
    ans = ans/math.sqrt(2*math.pi*var)
    return ans

def fit(dataset, label):
    index_survived = np.where(label==1)[0]
    index_not_survived = np.where(label==0)[0]
    p_survived = len(index_survived) / len(label)


    # BernoulliNB
    Bernoulli_feature = [0, 1, 3, 4]
    p_estimate_0 = [] # NOT survived
    p_estimate_1 = [] # survived
    for feature in Bernoulli_feature:
        p_feature_0 = []
        p_feature_1 = []
        value = np.unique(dataset[:,feature])
        K = len(value)
        for val in value:
            p_feature_0.append(len(1+np.where((dataset[:, feature]==val) & (label==0))[0])/(
    K+len(index_not_survived)))
            p_feature_1.append(len(1+np.where((dataset[:, feature]==val) & (label==1))[0])/(
    K+len(index_survived)))
        p_estimate_0.append(p_feature_0)
        p_estimate_1.append(p_feature_1)

    # GaussianNB
    Gaussian_feature = [2, 5]
    Gaussian_parameter_0 = []
    Gaussian_parameter_1 = []
    dataset_0 = dataset[index_not_survived]
    dataset_1 = dataset[index_survived]

    for feature in Gaussian_feature:
        Gaussian_parameter_0.append([np.mean(dataset_0[:, feature]), np.var(dataset_0[:,
    feature])])
        Gaussian_parameter_1.append([np.mean(dataset_1[:, feature]), np.var(dataset_1[:,
    feature])])

    return p_estimate_0, p_estimate_1, Gaussian_parameter_0, Gaussian_parameter_1, p_survived
    , dataset, label


def predictNB(x, estimate):
    dataset = estimate[5]
    label = estimate[6]
    Bernoulli_feature = [0, 1, 3, 4]
    Gaussian_feature = [2, 5]

    # survived
    p = estimate[4]
    p_estimate_1 = estimate[1]
    Gaussian_parameter_1 = estimate[3]
```

```
    for i in range(len(Bernoulli_feature)):
        feature = Bernoulli_feature[i]
        unique_feature = np.unique(dataset[:, feature]).tolist()
        if x[feature] in unique_feature:
            j = unique_feature.index(x[feature])
        else:
            j = len(unique_feature)-1
#           j = np.where(np.unique(dataset[:, feature]) == x[feature])[0][0]
        p = p * p_estimate_1[i][j]
    for i in range(len(Gaussian_feature)):
        feature = Gaussian_feature[i]
        Mean = Gaussian_parameter_1[i][0]
        Var = Gaussian_parameter_1[i][1]
        p = p * gaussian(x[feature], Mean, Var)
    p_survived = p

    # NOT survived
    p = 1 - estimate[4]
    p_estimate_0 = estimate[0]
    Gaussian_parameter_0 = estimate[2]

    for i in range(len(Bernoulli_feature)):
        feature = Bernoulli_feature[i]
        unique_feature = np.unique(dataset[:, feature]).tolist()
        if x[feature] in unique_feature:
            j = unique_feature.index(x[feature])
        else:
            j = len(unique_feature)-1
#           j = np.where(np.unique(dataset[:, feature]) == x[feature])[0][0]
        p = p * p_estimate_0[i][j]
    for i in range(len(Gaussian_feature)):
        feature = Gaussian_feature[i]
        Mean = Gaussian_parameter_0[i][0]
        Var = Gaussian_parameter_0[i][1]
        p = p * gaussian(x[feature], Mean, Var)
    p_not_survived = p
    if p_survived > p_not_survived:
        return 1
    return 0
# estimate = fit(X, y_label)
# predictNB(X[1], estimate)


# KFold
from sklearn.model_selection import KFold
ans = np.zeros(len(X))
kf = KFold(n_splits = 10, shuffle = True)
kf.get_n_splits(X)
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y_label[train_index], y_label[test_index]

    estimate = fit(X_train, y_train)
    for i in range(len(test_index)):
        ans[test_index[i]] = (predictNB(X_test[i], estimate) == y_test[i])
accuracy = sum(ans)/len(ans)
print(accuracy)


# personal data
estimate = fit(X, y_label)
x = np.array([1, 1, 22, 1, 0, 71.2833])
print(predictNB(x, estimate))
```