main

March 16, 2023

[ ]:
```
import numpy as np
import random
from collections import Counter
```

[ ]:
```
########         (LIBSVM )
def load_svmfile(filename):
    X = []
    Y = []
    with open(filename, 'r') as f:
        filelines = f.readlines()
        for fileline in filelines:
            fileline = fileline.strip().split(' ')
            #print(fileline)
            Y.append(int(fileline[0]))
            tmp = []
            for t in fileline[1:]:
                if len(t)==0:
                    continue
                tmp.append(float(t.split(':')[1]))
            X.append(tmp)

    return np.array(X), np.array(Y)
```

[ ]:
```
########      https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.
 ↪html#svmguide1
########
########
dataset = 'svmguide1'
print('Start loading dataset {}'.format(dataset))
X, Y = load_svmfile(dataset) # train set
X_test, Y_test = load_svmfile('{}.t'.format(dataset)) # test set
print('trainset X shape {}, train label Y shape {}'.format(X.shape, Y.shape))
print('testset X_test shape {}, test label Y shape {}'.format(X_test.shape,␣
 ↪Y_test.shape))
```

```
Start loading dataset svmguide1
trainset X shape (3089, 4), train label Y shape (3089,)
testset X_test shape (4000, 4), test label Y shape (4000,)
```

```
[ ]: ########  KNN         train, test _calculate_distances
     class KNN_model():
         def __init__(self, k=1):
             self.k = k

         def train(self, x_train, y_train):
             """Implement the training code for KNN
             Input:
                 x_train: Training instances of size (N, D), where N denotes the␣
      ↪number of instances and D denotes the feature dimension
                 y_train: Training labels of size (N, )
             """
             self.x_train = x_train
             self.y_train = y_train

         def test(self, x_test):
             """
             Input: Test instances of size (N, D), where N denotes the number of␣
      ↪instances and D denotes the feature dimension
             Return: Predicted labels of size (N, )
             """
             #
             distances = np.array([self._calculate_distances(x_test_sample) for␣
      ↪x_test_sample in x_test])
             #    k
             indices = np.argsort(distances)[:, :self.k]
             #    k
             labels = self.y_train[indices]
             #
             y_pred = np.array([Counter(labels_for_one).most_common()[0][0] for␣
      ↪labels_for_one in labels])
             return y_pred

         def _calculate_distances(self, point):
             """Calculate the euclidean distance between a test instance and all␣
      ↪points in the training set x_train
             Input: a single point of size (D, )
             Return: distance matrix of size (N, )
             """
             return ((self.x_train - point) ** 2).sum(axis=1) ** 0.5

     # m = KNN_model(k=1)
     # m.train(np.array([[0, 0, 0], [1, 1, 1], [2, 2, 2]]), np.array([0, 1, 0]))
     # print(m._calculate_distances(np.array([0, 0, 0])))
     # print(m.test(np.array([[0, 0, 0], [0, 1, 1]])))
```

```
[ ]: #########
     random.seed(777777) #
     N = X.shape[0]
     valid_frac = 0.2 #      20%
     valid_size = int(N*valid_frac)

     #        random shuffle
     shuffle_index = [i for i in range(N)]
     random.shuffle(shuffle_index)
     valid_index, train_index = shuffle_index[:valid_size], shuffle_index[valid_size:
      ↪]
     X_valid, Y_valid = X[valid_index], Y[valid_index]
     X_train, Y_train = X[train_index], Y[train_index]
     print('trainset X_train shape {}, validset X_valid shape {}'.format(X_train.
      ↪shape, X_valid.shape))
```

```
trainset X_train shape (2472, 4), validset X_valid shape (617, 4)
```

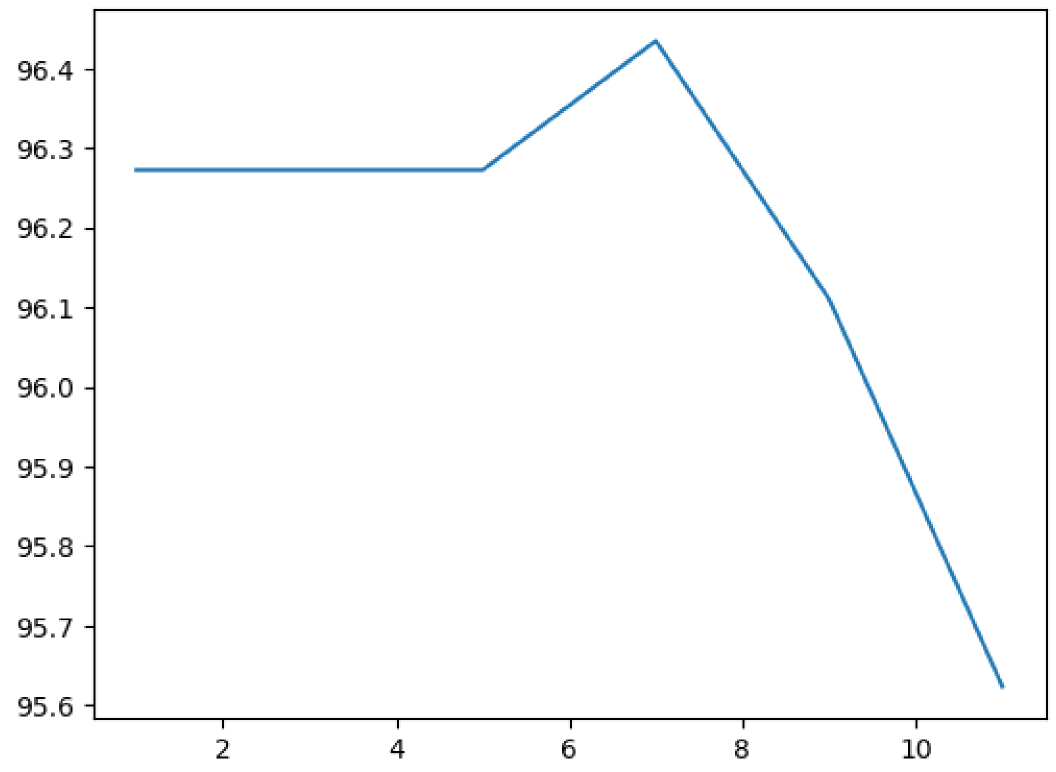
```
[ ]: #########                     0.95     95
     def cal_accuracy(y_pred, y_gt):
         '''
         y_pred: predicted labels (N,)
         y_gt: ground truth labels (N,)
         Return: Accuracy (%)
         '''
         return np.sum(y_pred == y_gt) / y_gt.shape[0] * 100
     assert abs(cal_accuracy(np.zeros(Y.shape[0]), Y)-100*1089.0/3089.0)<1e-3
```

```
[ ]: #####
     possible_k_list = [1,3,5,7,9,11] #
     accs = [] #    k
     for k in possible_k_list:
         #####     k
         model = KNN_model(k)
         #####    ,  : model.train()
         model.train(X_train, Y_train)
         #####  X_valid     Y_pred_valid,   model.test()
         Y_pred_valid = model.test(X_valid)
         #####
         acc_k = cal_accuracy(Y_pred_valid, Y_valid)
         #####   k
         accs.append(acc_k)
         print('k={}, accuracy on validation={}%'.format(k, acc_k))

     import matplotlib.pyplot as plt
     plt.plot(possible_k_list, accs) #   k
```

```
k=1, accuracy on validation=96.27228525121556%
k=3, accuracy on validation=96.27228525121556%
k=5, accuracy on validation=96.27228525121556%
k=7, accuracy on validation=96.43435980551054%
k=9, accuracy on validation=96.11021069692059%
k=11, accuracy on validation=95.62398703403565%
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x257cefacdf0>]
```



[ ]:
```
#####          k   k
best_k = 7

#####   k
best_model = KNN_model(best_k)

#####
best_model.train(X, Y)

#####       Y_pred_test
Y_pred_test = best_model.test(X_test)
print('Test Accuracy={}%'.format(cal_accuracy(Y_pred_test, Y_test)))
```

```
Test Accuracy=96.575%
```

```
[ ]: #####   5
     folds = 5

     #####(k,  ,  )
     k_mean_std_list = []

     for k in possible_k_list: #     k
         print('******k={}******'.format(k))
         valid_accs = []

         #########
         random.seed(777777) #
         N = X.shape[0]
         valid_frac = 1 / folds  #       1/folds
         valid_size = int(N * valid_frac)

         #        random shuffle
         shuffle_index = [i for i in range(N)]
         random.shuffle(shuffle_index)
         shuffle_X, shuffle_Y = X[shuffle_index], Y[shuffle_index]
         for i in range(folds): #  i
             #####   i    X_train_i, Y_train_i   X_valid_i, Y_valid_i;      random␣
      ↪shuffle   index
             X_valid_i, Y_valid_i = X[i * valid_size: (i+1) * valid_size], Y[i *␣
      ↪valid_size: (i+1) * valid_size]
             X_train_i = np.vstack((X[:i * valid_size], X[(i+1) * valid_size:]))
             Y_train_i = np.append(Y[:i * valid_size], Y[(i+1) * valid_size:])
             #####     k
             model = KNN_model(k)
             ##### Fold-i
             model.train(X_train_i, Y_train_i)
             #####  Fold-i X_valid_i    Y_pred_valid_i
             Y_pred_valid_i = model.test(X_valid_i)
             acc = cal_accuracy(Y_pred_valid_i, Y_valid_i)
             valid_accs.append(acc)
             print('Valid Accuracy on Fold-{}: {}%'.format(i+1, acc))

         k_mean_std_list.append((k, np.mean(valid_accs), np.std(valid_accs)))
         print('k={}, Accuracy {}+-{}%'.format(*k_mean_std_list[len(k_mean_std_list)␣
      ↪- 1]))

     print('k_mean_std_list:', k_mean_std_list)
```

```
******k=1******
Valid Accuracy on Fold-1: 96.11021069692059%
Valid Accuracy on Fold-2: 95.78606158833063%
Valid Accuracy on Fold-3: 96.75850891410049%
Valid Accuracy on Fold-4: 88.49270664505673%
Valid Accuracy on Fold-5: 92.70664505672609%
k=1, Accuracy 93.9708265802269+-3.0741232335892827%
******k=3******
Valid Accuracy on Fold-1: 96.27228525121556%
Valid Accuracy on Fold-2: 95.78606158833063%
Valid Accuracy on Fold-3: 97.24473257698541%
Valid Accuracy on Fold-4: 90.76175040518638%
Valid Accuracy on Fold-5: 93.67909238249594%
k=3, Accuracy 94.74878444084278+-2.3094344415632944%
******k=5******
Valid Accuracy on Fold-1: 96.75850891410049%
Valid Accuracy on Fold-2: 96.27228525121556%
Valid Accuracy on Fold-3: 97.08265802269044%
Valid Accuracy on Fold-4: 91.24797406807131%
Valid Accuracy on Fold-5: 93.03079416531604%
k=5, Accuracy 94.87844408427875+-2.320780851820909%
******k=7******
Valid Accuracy on Fold-1: 96.5964343598055%
Valid Accuracy on Fold-2: 96.5964343598055%
Valid Accuracy on Fold-3: 97.24473257698541%
Valid Accuracy on Fold-4: 92.05834683954619%
Valid Accuracy on Fold-5: 93.354943273906%
k=7, Accuracy 95.17017828200972+-2.06643467753766%
******k=9******
Valid Accuracy on Fold-1: 96.5964343598055%
Valid Accuracy on Fold-2: 96.27228525121556%
Valid Accuracy on Fold-3: 97.08265802269044%
Valid Accuracy on Fold-4: 92.05834683954619%
Valid Accuracy on Fold-5: 93.03079416531604%
k=9, Accuracy 95.00810372771474+-2.051123762665333%
******k=11******
Valid Accuracy on Fold-1: 96.5964343598055%
Valid Accuracy on Fold-2: 96.43435980551054%
Valid Accuracy on Fold-3: 96.92058346839546%
Valid Accuracy on Fold-4: 92.22042139384116%
Valid Accuracy on Fold-5: 92.86871961102106%
k=11, Accuracy 95.00810372771474+-2.0279406573247405%
k_mean_std_list: [(1, 93.9708265802269, 3.0741232335892827), (3,
94.74878444084278, 2.3094344415632944), (5, 94.87844408427875,
2.320780851820909), (7, 95.17017828200972, 2.06643467753766), (9,
95.00810372771474, 2.051123762665333), (11, 95.00810372771474,
2.0279406573247405)]
```

[ ]:
```
#####          k   k
best_k = 7
#####   k
```

```
best_model = KNN_model(best_k)
#####
best_model.train(X, Y)
#####      Y_pred_test
Y_pred_test = best_model.test(X_test)
print('Test Accuracy chosing k using cross-validation={}%'.
 ↪format(cal_accuracy(Y_pred_test, Y_test)))
```

```
Test Accuracy chosing k using cross-validation=96.575%
```

[ ]:
```
#####  /
#####             1
N_test = int(X_test.shape[0]*0.7)
X_test, Y_test = X_test[:N_test], Y_test[:N_test]
print(Counter(Y_test)) #

model = KNN_model(k=7) #         k
model.train(X, Y)
Y_pred_test = model.test(X_test)

#  percision  recall F1 score
def cal_prec_recall_f1(Y_pred, Y_gt):
    '''
    Input: predicted labels y_pred, ground truth labels Y_gt
    Retur: precision, recall, and F1 score
    '''
    TP = np.bitwise_and(Y_pred == 1, Y_gt == 1).sum()
    FP = np.bitwise_and(Y_pred == 1, Y_gt == 0).sum()
    FN = np.bitwise_and(Y_pred == 0, Y_gt == 1).sum()
    TN = np.bitwise_and(Y_pred == 0, Y_gt == 0).sum()

    precision = TP / (TP + FP) if (TP + FP) != 0 else 1
    recall = TP / (TP + FN) if (TP + FN) != 0 else 1
    f1 = 2 * precision * recall / (precision + recall)
    return precision, recall, f1

print(cal_prec_recall_f1(Y_pred_test, Y_test))
```

```
Counter({0: 2000, 1: 800})
(0.910271546635183, 0.96375, 0.936247723132969)
```

0.1

0.1.1 Precision Recall

Precision Recall , , , 1.

0.1.2 Numpy

Numpy , , .

0.2

, , .