火箭班

注意事项:

Efficient code

Commenting

Authorship

Assignment 1: K-Nearest Neighbor (10 marks)

Student Name:

Student ID:

General info

Due date: Friday, 18 March 2022 5pm

Submission method: Canvas submission

Submission materials: completed copy of this iPython notebook

Late submissions: -10% per day up to 5 days (both weekdays and weekends count). Submissions more than 5 days late will not be accepted (resul in a mark of 0).

- one day late, -1.0;
- two days late, -2.0;
- three days late, -3.0;
- four days late, -4.0;
- five days late, -5.0;

Evaluation: Your iPython notebook should run end-to-end without any errors in a reasonable amount of time, and you must follow all instructions provided below, including specific implementation requirements and instructions for what needs to be printed (please avoid printing output we don't ask for). You should edit the sections below where requested, but leave the rest of the code as is. You should leave the output from running your code in the iPython notebook you submit, to assist with marking. The amount each section is worth is given in parenthesis after the instructions.

You will be marked not only on the correctness of your methods, but also the quality and efficency of your code: in particular, you should be careful to use Python built-in functions and operators when appropriate and pick descriptive variable names that adhere to Python style requirements. If you think it might be unclear what you are doing, you should comment your code to help the marker make sense of it. We reserve the right to deduct up to 2 marks for unreadable or exessively inefficient code.

IMPORTANT

Please carefully read and fill out the **Authorship Declaration** form at the bottom of the page. Failure to fill out this form results in the following deductions:

- missing Authorship Declaration at the bottom of the page, -5.0
- incomplete or unsigned Authorship Declaration at the bottom of the page, -3.0

Overview

In this homework, you'll be applying the K-nearest neighbor (KNN) classification algorithm to a real-world machine learning data set. In particular, we will predict the primary color of national flags given a diverse set of features, including the country's size and population and other structural properties of the flag.

Firstly, you will read in the dataset into a train and a test set, and you will create two feature sets (Q1). Secondly, you will implement different distance functions (Q2). Thirdly, you will implement two KNN classifiers (Q3, Q4) and apply it to the data set using different distance functions and parameter K (Q5). Finally, you will assess the quality of your classifier by comparing its class predictions to the true (or "gold standard") labels (Q6).

Question 1: Loading the data (1.0 marks)

Instructions: For this assignment we will develop a K-Nearest Neighbors (KNN) classifier to predict the predominant color of national flags. The list of classes (colors) is:

```
black
blue
brown
gold
green
orange
red
white
```

The dataset consists of 194 instances. Each instance corresponds to a national flag which has a unique identifier (itemX; first field) and is characterized with 25 features as described in the file *flags.names* which is provided as part of this assignment.

You need to first obtain this dataset, which is on Canvas (assignment 1). The files *flags.features* and *flags.labels* contain the data we will use in this notebook. Make sure the files are saved in the same folder as this notebook.

Both files are in comma-separated value (csv) format. The first line in each file is a header, naming each feature (or label).

flags.features contains 194 instances, one line per instance. The first field is the unique instance identifier (name of country). The following fields contain the 25 features, as described in the file *flags.names*.

flags.labels contains the true labels (i.e., one of the nine color classes above), one instance per line. Again, the first field is the instance identifier, and the second field the instance label.

flags.names contains additional explanations about the data set and the features.

All feature values are integers, and for Questions 1 through 5, we make the simplifying assumption that all values are indeed numeric. You may want to revisit this assumption in Question 6.

Question 1a [0.5 mark]

Task: Read the two files

1. create a **training_feature** set (list of features for the first 150 instances in the flags.* files) and a **training_label** set (list of labels for the corresponding).
2. create a **test_feature** set (list of features of the remaining instances in the flags.* files) and a **test_label** set (list of labels for the corresponding).

- Do **not** shuffle the data.
- Do **not** modify feature or label representations.

You may use any Python packages you want, but not change the specified data types (i.e., they should be of type List, and *not* dataframe, dictionary etc).

```
data = open("flags.features", 'r').readlines()
labels = open("flags.labels", 'r').readlines()

train_features = []
train_labels   = []
test_features = []
test_labels   = []


###########################
## YOUR CODE BEGINS HERE
###########################
counter = 0
for instance in data[1:]:
    if counter < 150:

        instance = instance.strip() #remove all leading and trailing whitespa
        instance = instance.split(",") # split each instance at each comma, 
        instance = instance[1:] # store the fields as the instance's features
        instance = [float(i) for i in instance]
        train_features.append(instance)
    else:

        instance = instance.strip() #remove all leading and trailing whitespa
        instance = instance.split(",") # split each instance at each comma, 
        instance = instance[1:] # store the fields as the instance's features
        instance = [float(i) for i in instance]
        test_features.append(instance)
    counter += 1


counter = 0
for instance in labels[1:]:
    if counter < 150:

        instance = instance.strip() #remove all leading and trailing whitespa
        instance = instance.split(",") # split each instance at each comma, 
        train_labels.append(instance[-1])
    else:
        instance = instance.strip() #remove all leading and trailing whitespa
        instance = instance.split(",") # split each instance at each comma, 
        test_labels.append(instance[-1])
    counter += 1
```

In [5]:

```
data
```

Out[5]:

```
['name,landmass,zone,area,population,language,bars,stripes,colours,red,g
reen,blue,gold,white,black,orange,circles,crosses,saltires,quarters,suns
tars,crescent,triangle,icon,animate,text\n',
 'item1,5,1,648,16,10,0,3,5,1,1,0,1,1,1,0,0,0,0,0,1,0,0,1,0,0\n',
 'item2,3,1,29,3,6,0,0,3,1,0,0,1,0,1,0,0,0,0,0,1,0,0,0,1,0\n',
 'item3,4,1,2388,20,8,2,0,3,1,1,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0\n',
 'item4,6,3,0,0,1,0,0,5,1,0,1,1,1,0,1,0,0,0,0,0,0,1,1,1,0\n',
 'item5,3,1,0,0,6,3,0,3,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0\n',
 'item6,4,2,1247,7,10,0,2,3,1,0,0,1,0,1,0,0,0,0,0,1,0,0,1,0,0\n',
 'item7,1,4,0,0,1,0,1,3,0,0,1,0,1,0,1,0,0,0,0,0,0,0,0,1,0\n',
 'item8,1,4,0,0,1,0,1,5,1,0,1,1,1,1,0,0,0,0,0,1,0,1,0,0,0\n',
 'item9,2,3,2777,28,2,0,3,2,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0\n',
 'item10,2,3,2777,28,2,0,3,3,0,0,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0\n',
 'item11,6,2,7690,15,1,0,0,3,1,0,1,0,1,0,0,0,1,1,1,6,0,0,0,0,0\n',
 'item12,3,1,84,8,4,0,3,2,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0\n',
 'item13,1,4,19,0,1,0,3,3,0,0,1,1,0,1,0,0,0,0,0,0,0,1,0,0,0\n',
 'item14,5,1,1,0,8,0,0,2,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0\n',
 'item15,5,1,143,90,6,0,0,2,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0\n',
 'item16,1,4,0,0,1,3,0,3,0,0,1,1,0,1,0,0,0,0,0,0,0,1,0,0\n',
```

In [4]:

```
labels[1:]
```

Out[4]:

```
['item1,green\n',
 'item2,red\n',
 'item3,green\n',
 'item4,blue\n',
 'item5,gold\n',
 'item6,red\n',
 'item7,white\n',
 'item8,red\n',
 'item9,blue\n',
 'item10,blue\n',
 'item11,blue\n',
 'item12,red\n',
 'item13,blue\n',
 'item14,red\n',
 'item15,green\n',
```

```
assert len(train_features[0]) == 25
assert len(train_features)  == 150
assert len(test_features)  == 44
```

In [21]:

```
train_features
```

Out[21]:

```
[[5.0,
  1.0,
  648.0,
  16.0,
  10.0,
  0.0,
  3.0,
  5.0,
  1.0,
  1.0,
  0.0,
  1.0,
  1.0,
  1.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
```

```
train_features[0]
```

```
[5.0,
 1.0,
 648.0,
 16.0,
 10.0,
 0.0,
 3.0,
 5.0,
 1.0,
 1.0,
 0.0,
 1.0,
 1.0,
 1.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 1.0,
 0.0,
 0.0,
 1.0,
 0.0,
 0.0]
```

```
train_labels
```

```
['green',
 'red',
 'green',
 'blue',
 'gold',
 'red',
 'white',
 'red',
 'blue',
 'blue',
 'blue',
 'red',
 'blue',
 'red',
 'green',
 'blue',
 'gold',
 'blue',
 'green',
```

Question 1b [0.5 marks]

Task Create a reduced feature set which only includes the "Structural Flag Features". The file *flag.names* specifies these features.

You may use any Python packages you want, but not change the specified data types. You may (but don't have to) hard-code feature indices.

```
info = open("flag.names", 'r').readlines()
info
```

```
['1. TItle: Flag database\n',
 '\n',
 '2. Source Information\n',
 '   -- Creators: Collected primarily from the "Collins Gem Guide to Flag
s":\n',
 '      Collins Publishers (1986).\n',
 '   -- Donor: Richard S. Forsyth \n',
 '             8 Grosvenor Avenue\n',
 '             Mapperley Park\n',
 '             Nottingham NG3 5DX\n',
 '             0602-621676\n',
 '   -- Date: 5/15/1990\n',
 '\n',
 '3. Past Usage:\n',
 "   -- None known other than what is shown in Forsyth's PC/BEAGLE User's G
uide.\n",
 '\n',
 '4. Relevant Information:\n',
 '   -- This data file contains details of various nations and their flag
s.\n',
 '      In this file the fields are separated by spaces (not commas).  With
\n',
 '      this data you can try things like predicting the religion of a coun
try\n',
 '      from its size and the colours in its flag.  \n',
 '   -- 10 attributes are numeric-valued.  The remainder are either Boolean
-\n',
 '      or nominal-valued.\n',
 '\n',
 '5. Number of Instances: 194\n',
 '\n',
 '6. Number of attributes: 25\n',
 '\n',
 '7. Instance identifier\n',
 '   1. name\tName of the country concerned\n',
 '\n',
 '\n',
```

```
 '8. Country Features:\n',
 '   1. landmass\t1=N.America, 2=S.America, 3=Europe, 4=Africa, 4=Asia, 6=O
ceania\n',
 '   2. zone\tGeographic quadrant, based on Greenwich and the Equator\n',
 '              1=NE, 2=SE, 3=SW, 4=NW\n',
 '   3. area\tin thousands of square km\n',
 '   4. population\tin round millions\n',
 '   5. language 1=English, 2=Spanish, 3=French, 4=German, 5=Slavic, 6=Othe
r \n',
 '              Indo-European, 7=Chinese, 8=Arabic, \n',
 '              9=Japanese/Turkish/Finnish/Magyar, 10=Others\n',
 '              \n',
 '9. Flag Structure Features:\n',
 '   6. bars     Number of vertical bars in the flag\n',
 '   7. stripes  Number of horizontal stripes in the flag\n',
 '   8. colours  Number of different colours in the flag\n',
 '   9. red      0 if red absent, 1 if red present in the flag\n',
 '  10. green    same for green\n',
 '  11. blue     same for blue\n',
 '  12. gold     same for gold (also yellow)\n',
 '  13. white    same for white\n',
 '  14. black    same for black\n',
 '  15. orange   same for orange (also brown)\n',
 '  16. circles  Number of circles in the flag\n',
 '  17. crosses  Number of (upright) crosses\n',
 '  18. saltires Number of diagonal crosses\n',
 '  19. quarters Number of quartered sections\n',
 '  20. sunstars Number of sun or star symbols\n',
 '  21. crescent 1 if a crescent moon symbol present, else 0\n',
 '  22. triangle 1 if any triangles present, 0 otherwise\n',
 '  23. icon     1 if an inanimate image present (e.g., a boat), otherwise
0\n',
 '  24. animate  1 if an animate image (e.g., an eagle, a tree, a human han
d)\n',
 '              present, 0 otherwise\n',
 '  25. text     1 if any letters or writing on the flag (e.g., a motto or
\n',
 '              slogan), 0 otherwise\n']
```

```
###########################
## YOUR CODE BEGINS HERE
###########################

test_features_structure = list(map(list, zip(*test_features)))[5:]
test_features_structure = list(map(list, zip(*test_features_structure)))
train_features_structure = list(map(list, zip(*train_features)))[5:]
train_features_structure = list(map(list, zip(*train_features_structure)))

###########################
## YOUR CODE ENDS HERE
###########################
```

```
###########################
## YOUR CODE BEGINS HERE
###########################
columns = range(5,25)
test_features_structure = [[row[i] for i in columns] for row in test_features]
train_features_structure = [[row[i] for i in columns] for row in train_features]

###########################
## YOUR CODE ENDS HERE
###########################
```

```
###########################
## YOUR CODE BEGINS HERE
###########################
def column(matrix, columns):

    return [[row[i] for i in columns] for row in matrix]


test_features_structure = column(test_features,range(5,25))

train_features_structure = column(train_features,range(5,25))

###########################
## YOUR CODE ENDS HERE
###########################
```

```
l = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
list(map(list, zip(*l)))
```

```
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

转置

取 columns

List comprehension

```
len([0.0,
  3.0,
  5.0,
  1.0,
  1.0,
  0.0,
  1.0,
  1.0,
  1.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
  1.0,
  0.0,
  0.0,
  1.0,
  0.0,
  0.0])
```

```
20
```

```
train_features_structure
```

```
[[0.0,
  3.0,
  5.0,
  1.0,
  1.0,
  0.0,
  1.0,
  1.0,
  1.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
  1.0,
  0.0,
  0.0,
  1.0,
  0.0,
```

Question 2: Distance Functions [1.0 mark]

Instructions: Implement the two distance functions specified below.

1. Manhattan distance
2. Cosine distance

Each distance function takes as input

- Two feature vectors (each of type List)

and returns as output

- The distance between the two feature vectors (float)

```python
import math

#check input
def manhattan_distance(fw1, fw2):
    # insert code here


    if len(fw1) != len(fw2):
        raise ValueError("Arrays must have the same size")



    sum = 0
    for i in range(len(fw1)):
        sum += abs(fw1[i] - fw2[i])
    distance =  sum

    return distance





def cosine_distance(fw1, fw2):
    # insert code here

    if len(fw1) != len(fw2):
        raise ValueError("Arrays must have the same size")



    sum1, sum2, sumdot = 0, 0, 0
    for i in range(len(fw1)):
        x = fw1[i]; y = fw2[i]
        sum1 += x*x
        sum2 += y*y
        sumdot += x*y
    distance =  1 - (sumdot/math.sqrt(sum1*sum2))

    return distance
```

```python
def manhattan_distance(fw1, fw2):


    if len(fw1) != len(fw2):
        raise ValueError("Arrays must have the same size")



    distance = sum(abs(val1-val2) for val1, val2 in zip(fw1,fw2))
    return distance
```

```python
assert manhattan_distance([1,0],[0.5,1])==1.5
assert cosine_distance([1,1,1,1], [0,1,0,0])==0.5
```

Question 3: KNN Classifier [2.0 marks]

Instructions: Here, you implement your KNN classifier. It takes as input

- training data features
- training data labels
- test data features
- parameter K
- distance function(s) based on which nearest neighbors will be identified

It returns as output

- the predicted labels for the test data

Ties among distances. If there are more than K instances with the same (smallest) distance value, consider the first K.

Ties at prediction time. Ties can also occur at class prediction time when two (or more) classes are supported by the same number of neighbors. In that case choose the class of the 1 nearest neighbor.

You should implement the classifier from scratch yourself, i.e., **you must not** use an existing implementation in any Python library. You may use Python packages (e.g., math, numpy, collections, ...) to help with your implementation.

```
def KNN(train_features, train_labels, test_features, k, dist_fun, weighted=False):

    predictions = []

    ###########################
    ## Your answer BEGINS HERE
    ###########################
    EPSILON = 0.00001

    for i in test_features:


        # compute distances to train_features
        if dist_fun == manhattan_distance:
            distances = [manhattan_distance(i,j) for j in train_features]


        elif dist_fun == cosine_distance:
            distances = [cosine_distance(i,j) for j in train_features]

        else:
            print("Distance function not supported")
            return
```

```python
        # get closest k neighbour

        # for potential tie breaking record 1nn

        onenn_index =distances.index(min(distances))
        onenn = train_labels[onenn_index]

        index_list = []

        weight_list = []

        for k_value in range(k):

            # find min and its index

            # find min and its index
            index_value_pair = list(enumerate(distances))
            value_index_pair = []
            for i,v in index_value_pair:
                value_index_pair.append((v,i))
            m,i = min(value_index_pair)

            #m,i = min((v,i) for i,v in enumerate(distances))


            # record index
            index_list.append(i)

            # record weight
            if weighted:

                weight_list.append(1/(distances[i]+EPSILON))
            else:
                weight_list.append(1)


            # set the position to maximum
            distances[i] = float("inf")


        # print(weight_list)

        # take corrsponding element
        knn_labels = [train_labels[i] for i in index_list]
```

```python
        # vote and predict

        mx = 0
        prediction = 0
        for i in set(knn_labels):

            # this gives the indices of the same label
            indices = [index for index,label in enumerate(knn_labels) if label == i]
            vote = sum([weight_list[index] for index in indices])


            if vote == mx:
                # 检测 tie,  如果出现, 替换 prediction
                prediction = onenn
            if vote > mx:
                mx = vote
                prediction = i


        predictions.append(prediction)




    ###########################
    ## Your answer ENDS HERE
    ###########################

    return predictions
```

In [105]:
```
KNN([[1,2],[4,5],[1,1.9], [9,2]], [1,0,0,1], [[1,1]], 3, manhattan_distance)
```

```
[1, 1, 1]
```

Out[105]:
```
[0]
```

In [108]:
```
KNN([[1,3],[1,2],[1,3], [1,2]], [1,0,0,1], [[1,1]], 4, manhattan_distance, True)
```

```
[0.9999900000999989, 0.9999900000999989, 0.49999750001249993, 0.49999750001249993]
```

```
    # get closest k neighbour

    # for potential tie breaking record 1nn

    onenn_index =distances.index(min(distances))
    onenn = train_labels[onenn_index]


    if weighted:

        weight_list = [1/(distances[i]+EPSILON) for i in sorted(range(len(distances)), key=lambda x: distances[x])[:k]]
    else:
        weight_list =[1] * k

    knn_labels  = [train_labels[i] for i in sorted(range(len(distances)), key=lambda x: distances[x])[:k]]

    # print(weight_list)
```

```
    # vote and predict
    #if there are multiple max, break tie

    vote_dict = {}
    for i in set(knn_labels):
        vote_dict[i] = 0
    for j in range(len(knn_labels)):

        vote_dict[knn_labels[j]] += weight_list[j]


    # print(vote_dict)

    # now we find mx value in this dict
    max_value = max(vote_dict.values())

    labels_with_max_weight = [key for key,val in vote_dict.items() if val == max_value]



    if len(labels_with_max_weight) != 1:
        # print("tie occured")
        predictions.append(onenn)

    else:
        predictions.append(labels_with_max_weight[0])
```

```
KNN([[1,2],[4,5],[1,3], [9,2]], [1,0,0,1], [[1,1]], 2, manhattan_distance)
```

```
[1, 1]
tie occured

[1]
```

```
KNN([[1,3],[1,2],[1,3], [1,2]], [1,0,0,1], [[1,1]], 4, manhattan_distance, True)
```

```
[0.9999900000999989, 0.9999900000999989, 0.49999750001249993, 0.49999750001249993]
tie occured

[0]
```

```python
distance = [1,2,3,4,5,6,5,3,4,3]
train_label = ["a","a","b","a","a","b","a","c","a","b"]

list(enumerate(distance))

k_value = 4

index_list = []



for k in range(k_value):

    # find min and its index
    m = min(distance)
    i = distance.index(m)


    # record index
    index_list.append(i)


    # set the position to maximum
    distance[i] = float("inf")




# take corrsponding element
[train_label[i] for i in index_list]
```

```
['a', 'a', 'b', 'c']
```

```python
values = [3,4,5]

index_value_pair = list(enumerate(values))
value_index_pair = []
for i,v in index_value_pair:
    value_index_pair.append((v,i))
m,i = min(value_index_pair)



print (m,i)
```

```
3 0
```

```python
(m,i) = min((v,i) for i,v in enumerate(values))


print (m,i)
```

```
3 0
```

```python
list(enumerate(values))
```

```
[(0, 3), (1, 4), (2, 5)]
```

```python
assert KNN([[1,1],[5,5],[1,2]], [1,0,1], [[1,1]], 1, cosine_distance) == [1]
assert KNN([[1,1],[5,5],[1,2]], [1,0,1], [[1,1]], 1, manhattan_distance) == [1]
assert KNN([[1,1],[4,5],[1,2], [5,4]], [1,0,0,1], [[1,1]], 3, manhattan_distance) == [0]
```

Question 4: Weighted KNN Classifier [1.0 mark]

Instructions: Extend your implementation of the KNN classifier in Question 3 to a Weighted KNN classifier. Use Inverse Distance as weights:

$w_j = \frac{1}{d_j+\epsilon}$

where

- d_j is the distance of of the jth nearest neighbor to the test instance
- $\epsilon = 0.00001$

Use the Boolean parameter `weighted` to specify the KNN version when calling the function.

Question 5: Applying your KNN classifiers to the Flags Dataset [0.5 marks]

Using the functions you have implemented above, please

1. For each of the distance functions you implemented in Question 2, construct (a) two majority voting KNN classifiers and (b) two weighted KNN classifiers, respectively, with

- K=1
- K=5

You will obtain a total of 16 (2 distance functions x 2 K values x 2 KNN versions x 2 feature sets) classifiers.

2. Compute the test accuracy for each model, where the accuracy is the fraction of correctly predicted labels over all predictions. Use the `accuracy_score` function from the `sklearn.metrics` package to obtain your accuracy.

```
# Your Code STARTS HERE
########################
predictions = []
param = []
for func in [manhattan_distance, cosine_distance]:
    for w in [False,True]:
        for s in [(train_features,test_features), (train_features_structure,test_features_structure)]:
            for k in [1,5]:
                predictions.append(KNN(s[0], train_labels,s[1],k,func,w))
                param.append([func,w,len(s[0]),k])



accuracy_knn_man_1 = accuracy_score(test_labels,predictions[0])
accuracy_knn_man_5 = accuracy_score(test_labels,predictions[1])

accuracy_knn_man_1_structure = accuracy_score(test_labels,predictions[2])
accuracy_knn_man_5_structure = accuracy_score(test_labels,predictions[3])

accuracy_knn_man_1_w = accuracy_score(test_labels,predictions[4])
accuracy_knn_man_5_w = accuracy_score(test_labels,predictions[5])

accuracy_knn_man_1_w_structure = accuracy_score(test_labels,predictions[6])
accuracy_knn_man_5_w_structure = accuracy_score(test_labels,predictions[7])

accuracy_knn_cos_1 = accuracy_score(test_labels,predictions[8])
accuracy_knn_cos_5 = accuracy_score(test_labels,predictions[9])

accuracy_knn_cos_1_structure = accuracy_score(test_labels,predictions[10])
accuracy_knn_cos_5_structure = accuracy_score(test_labels,predictions[11])

accuracy_knn_cos_1_w = accuracy_score(test_labels,predictions[12])
accuracy_knn_cos_5_w = accuracy_score(test_labels,predictions[13])

accuracy_knn_cos_1_w_structure = accuracy_score(test_labels,predictions[14])
accuracy_knn_cos_5_w_structure = accuracy_score(test_labels,predictions[15])


accuracy_knn_sm_1_structure = 0
accuracy_knn_sm_5_structure = 0

accuracy_knn_sm_1_w_structure = 0
accuracy_knn_sm_5_w_structure = 0

########################
```

```
Results on the *full* feature set

manhattan (majority vote)
K=1 0.273
K=5 0.386
-----------
manhattan (weighted)
K=1 0.273
K=5 0.364

cosine (majority vote)
K=1 0.341
K=5 0.364
-----------
cosine (weighted)
K=1 0.341
K=5 0.295


Results on the *structure* feature set

manhattan (majority vote)
K=1 0.364
K=5 0.409
-----------
manhattan (weighted)
K=1 0.364
K=5 0.432

cosine (majority vote)
K=1 0.386
K=5 0.386
-----------
cosine (weighted)
K=1 0.386
K=5 0.386
```

```
param
```

```
[[<function __main__.manhattan_distance(fw1, fw2)>, False, 25, 1],
 [<function __main__.manhattan_distance(fw1, fw2)>, False, 25, 5],
 [<function __main__.manhattan_distance(fw1, fw2)>, False, 20, 1],
 [<function __main__.manhattan_distance(fw1, fw2)>, False, 20, 5],
 [<function __main__.manhattan_distance(fw1, fw2)>, True, 25, 1],
 [<function __main__.manhattan_distance(fw1, fw2)>, True, 25, 5],
 [<function __main__.manhattan_distance(fw1, fw2)>, True, 20, 1],
 [<function __main__.manhattan_distance(fw1, fw2)>, True, 20, 5],
 [<function __main__.cosine_distance(fw1, fw2)>, False, 25, 1],
 [<function __main__.cosine_distance(fw1, fw2)>, False, 25, 5],
 [<function __main__.cosine_distance(fw1, fw2)>, False, 20, 1],
 [<function __main__.cosine_distance(fw1, fw2)>, False, 20, 5],
 [<function __main__.cosine_distance(fw1, fw2)>, True, 25, 1],
 [<function __main__.cosine_distance(fw1, fw2)>, True, 25, 5],
 [<function __main__.cosine_distance(fw1, fw2)>, True, 20, 1],
 [<function __main__.cosine_distance(fw1, fw2)>, True, 20, 5]]
```

Question 6: Analysis [4.5 marks]

1. (a) Discuss the appropriateness of each of the distance functions to our two versions of the *flags* data set. Where appropriate, explain why you expect them to perform poorly referring to both their mathematical properties and the given feature set. **[0.5 marks]**

   (b) Imagine you could choose a third distance function for the *Structure* feature set: either Hamming or Euclidean. Which one would you choose and why? Do you expect the results to be similar or different from Manhattan and Cosine [*N.B. you should only hypothesize based on the definitions of the metrics. You do not need to write any code*] **[0.5 marks]**

2. Does the Weighted KNN outperform the Majority voting version, or vice versa? Hypothesize why (not). **[1 mark]**

3. (a) Plot a histogram of the actual class frequencies in the test set, and a histogram of the predicted test labels for the knn_man_5 model. You should produce **a single plot** which shows the histogram both true and predicted labels. Label the x-axis and y-axis. [*N.B. you may use libraries like matplotlib or seaborne*] **[1 mark]**

   (b) Describe and explain the discrepancy between the true and predicted distributions. **[1 mark]**

4. Do you think the accuracy is an appropriate evaluation metric for the *Flags* data set? Why (not)? **[0.5 marks]**

Each question should be answered in no more than 3-4 sentences.

1a)
Manhattan   full   suitable as the range for different attribute varies (area pop)
                     structure less suitable as there are less numeric att,and the range are closer

Cosine   full   not suitable as the data point can be far away yet have similar
  flag   and some att shouldnt be treated as numeric   zone lan
                           structure   more suitable

data   man/cos   numeric

overall both are not approprieate, most features shouldnt be treated as numeric, use hamming

poor because soem feature can introduce a lot of distance area / pop

majority of the features are not numeric

1b) hamming   cat features,   different and better
similar structured flag are likely to have similar color

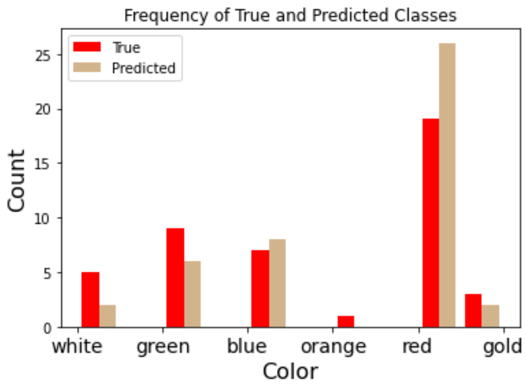
2
it does not, typicaly country will try not to have their flag designed too similar, so the nearer neighbour does not neccessarilly have greater weight.

Type code for 3.(a) in the cell below, and answer 3.(b) below

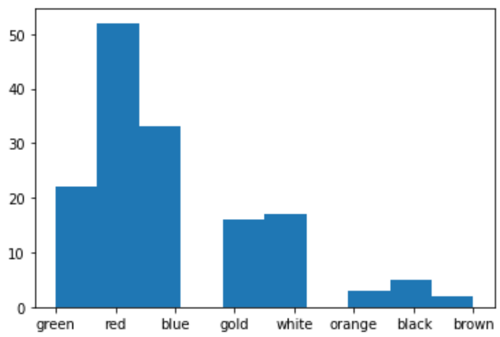
3b)
红的太多了， 什么多就会预测成什么，irrelevant class

4
dataset is inbalanced, the ability to predict rarer color is not reflected by acc

```python
import matplotlib.pyplot as plt

##################################################
# Your answer to Question 6 (3) STARTS HERE
##################################################

colors = ['red', 'tan']
plt.hist([test_labels,predictions[1]], histtype='bar', color=colors,label = ['True','Predicted'])
plt.legend(prop={'size': 10})
plt.xlabel("Color", fontsize=16)
plt.ylabel("Count", fontsize=16)
plt.xticks(fontsize=14)
plt.title("Frequency of True and Predicted Classes")
plt.show()


##################################################
# Your answer to Question 6 (3) ENDS HERE
##################################################
```




```python
plt.hist(train_labels)
plt.show()
```



Assignment 1 更新

有同学反应说 instructor 在 piazza 中明确了 tie breaking 中 1NN 的定义，

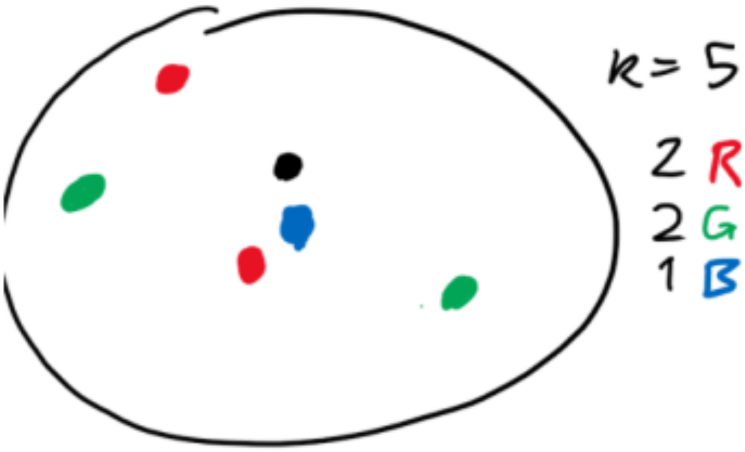
具体为，平手的 label 中的 nearest neighbour。

这与我们火箭班中讲解有所不同，所以大家需要更新一下自己的代码

Ties at prediction time - 1 nearest neighbour

Hi Lea,

I am confused over the wording in Q3:
Ties can also occur at class prediction time when two (or more) classes are supported by the same number of neighbors. **In that case choose the class of the 1 nearest neighbor.**
Does this refer to **the** closest neighbough, or the closest neighbour with a tied label?
I have illustrated this in the figure below:



Would blue or red be the correct prediction here?
Thank you

assignment1

Thank you

assignment1

edit · good question | 1 Updated 3 days ago by Anonymous Gear

the students' answer, *where students collectively construct a single answer*

Tie occurs for red & green, not blue. Therefore, I think you pick red for this situation.

edit · thanks! | 0 Updated 2 days ago by Zian Wang

the instructors' answer, *where instructors collectively construct a single answer*

yes it's true. The correct label here would be red.

thanks! | 0 Updated 16 hours ago by Hasti Samadi

具体会影响到 KNN 函数的第三部分和 Evaluation 结果

```
def KNN(train_features, train_labels, test_features, k, dist_fun, weighted=False):

    predictions = []

    ############################
    ## Your answer BEGINS HERE
    ############################
    EPSILON = 0.00001
    for i in test_features:


        # compute distances to train_features
        if dist_fun == manhattan_distance:
            distances = [manhattan_distance(i,j) for j in train_features]


        elif dist_fun == cosine_distance:
            distances = [cosine_distance(i,j) for j in train_features]

        else:
            print("Distance function not supported")
            return


        # get closest k neighbour

        knn_index = sorted(range(len(distances)), key=lambda x: distances[x])[:k]

        if weighted:

            weight_list = [1/(distances[i]+EPSILON) for i in knn_index]
        else:
            weight_list =[1] * k

        knn_labels  = [train_labels[i] for i in knn_index]
```

```
        # vote and predict
        #if there are multiple max, break tie

        vote_dict = {}
        for i in set(knn_labels):
            vote_dict[i] = 0
        for j in range(len(knn_labels)):

            vote_dict[knn_labels[j]] += weight_list[j]


        # now we find mx value in this dict
        max_value = max(vote_dict.values())

        labels_with_max_weight = [key for key,val in vote_dict.items() if val == max_value]

        if len(labels_with_max_weight) != 1:
            # print("tie occured")


            # tie occured, 1nn 方法1
            """

            tied_color_min_distance = float("inf")



            #针对每个平手的颜色，去找neighbour的distance，同时记录下最近的，
            #loop 跑完自然会把最近的那个颜色assign 给onenn

            for color in labels_with_max_weight:
                for i in range(len(knn_labels)):
                    if knn_labels[i] == color:
                        dis = distances[knn_index[i]]
                        if dis < tied_color_min_distance:
                            tied_color_min_distance = dis
                            onenn = color
            """
```

```python
            # 1nn 方法2
            '''

            tie_color = []
            tie_color_distance = []


            #找到所有符合平手颜色的 distance, 拿到对应最近neighbour的index


            for color in labels_with_max_weight:
                for i in range(len(knn_labels)):
                    if knn_labels[i] == color:
                        tie_color.append(color)
                        index = knn_index[i]
                        # store in tuple to make sure if tie occur when doing 1nn, we get the first one by index (occurence)
                        tie_color_distance.append((distances[index],index))
            ind = tie_color_distance.index(min(tie_color_distance))
            onenn = tie_color[ind]
            '''


            #1nn 方法3


            #knn label里拿第一个符合颜色的就行, 因为knn label是按距离排的


            for i in range(len(knn_labels)):
                if knn_labels[i] in labels_with_max_weight:
                    onenn = knn_labels[i]
                    break


            predictions.append(onenn)

        else:
            predictions.append(labels_with_max_weight[0])

    ###########################
    ## Your answer ENDS HERE
    ###########################

    return predictions
```

```
Results on the *full* feature set

manhattan (majority vote)
K=1 0.273
K=5 0.386
-----------
manhattan (weighted)
K=1 0.273
K=5 0.364

cosine (majority vote)
K=1 0.341
K=5 0.318
-----------
cosine (weighted)
K=1 0.341
K=5 0.295


Results on the *structure* feature set

manhattan (majority vote)
K=1 0.364
K=5 0.432
-----------
manhattan (weighted)
K=1 0.364
K=5 0.455

cosine (majority vote)
K=1 0.386
K=5 0.386
-----------
cosine (weighted)
K=1 0.386
K=5 0.386
```