Classification of different brands of cereals by manufacturer name using k-NN classifier in Python

University of New Brunswick-Electrical and Computer Engineering
Burak Koryan | burak@koryan.ca | http://koryan.ca
Date : May 11 2018

Objective:

The aim of this project is to use Python for classifying different brands of cereal produced by different companies listed in the introduction.The classification is done using k-NN in Python with the provided models/libraries in sklearn.

Introduction:

In order to make sense out of accumulated data that contains information about cereals produced by different companies with variety of calorie counts,protein,fat,and carbohydrate amount,there needs to be sorting done.If the amount of data in the database is hundreds of lines,then this sorting cannot be done manually and has to be done by a machine.In this case pattern recognition and machine learning can be used greatly to simplify this process.

The dataset used in this project,obtained from kaggle.com,has numerical data on ingredients (i.e. protein,sugar,carbohydrate,fat,sodium amount) of cereals.

name	mfr	type	calories	protein	fat	sodium	fiber	carbo	sugars	potass
100% Bran	N	C	70	4	1	130	10	5	6	280
100% Natural Bran	Q	C	120	3	5	15	2	8	8	135
All-Bran	K	C	70	4	1	260	9	7	5	320
All-Bran with Extra Fiber	K	C	50	4	0	140	14	8	0	330
Almond Delight	R	C	110	2	2	200	1	14	8	-1
Apple Cinnamon Cheerios	G	C	110	2	2	180	1.5	10.5	10	70

Table 1: Data sample from the cereal.csv dataset[3]

The manufacturers in the dataset are listed as below:

- A = American Home Food Products;
- G = General Mills
- K = Kellogg’s
- N = Nabisco

- P = Post
- Q = Quaker Oats
- R = Ralston Purina

In order to simplify the classification process only three columns of the dataset were used; the manufacturer column(#1),carbonhydrates column(#8),and potassium column(#10).The manufacturer column (column #1) has various companies listed with their first letters as shown below. The carbonhydrates and potassium columns have numerical data in grams. To make plotting more clear as well as classification process easier,only data from three manufacturers (Kellogg's, General Mills, and Post) are used :

- K = Kellogg's
- G = General Mills
- P = Post
-

Sorting and classifying the dataset shown above was done using Python with functions and models in the sklearn library then the results were plotted with help of matplotlib,numpy, and pandas libraries.

Procedure:

Step 1 :

As a first step, the libraries that will be used for plotting and classification are imported as from numpy,matplotlib,sklearn,and pandas.

Step 2:

In step 2, the number of neighbours for k-NN classification is determined as well as the dataset is imported. Then in order to conduct preprocessing, column #3 and column #5 of the dataset are assigned to x-axis, and column #1 is assigned to y-axis.

Step 3:

The number of data points(N is chosen 20) from Kellogg's, General Mills,and Post are counted using two for-loops and two empty arrays with the number of data points from these manufacturers are created.

Step 4:

Colormaps are created with hex codes for plotting (see Appendix A : Python code for more detail)

Step 5:

Classification is done in this step by assigning the number of neighbours and classification method of k-NN.

Step 6:

With the results obtained from the k-NN classifier. The plotting is done at this stage of Python code and shown in the results section.

Results:

There are two plots shown below which shows k-NN classification of the cereal data used in classification. Two different weights are used in the process : uniform and distance.

In Figure 1, the weights in the classification is chosen as ‘uniform’ with the number of neighbours of 20.The red color represents the manufacturer General Mills, the green color represents Kellogg’s, and the purple color represents the manufacturer Post.

k	**Uniform**	**Distance**
1	96.29%	96.29%
2	70.37%	96.29%
3	61.11%	96.29%
7	55.55%	96.29%
10	59.26%	96.29%

Table 2 : Accuracy results from k-NN classification of dataset

From the classification accuracy results shown in Table 2, we can conclude that the weight function ‘distance’ doesn’t affect the accuracy result even if the *k* is changed in the classification. However, when the weight function is changed to ‘uniform’, the classification accuracy decreases as the value of *k* increases. This definitely might be due to the classification weight function in the classification process. With *uniform* weight function, all points in each neighbourhood are weighted equally whereas in *distance* weight function, the points are weighted by the inverse of their distance [1].The difference from the classification results can be seen by looking through the plots from Figure 1 to Figure 10.

In Table 3,it can be seen that there are 38 (22+15+1) correctly predicted data points and 16 (8+2+6) false predictions in total in the k-NN classification when k = 2.This also verifies the accuracy obtained, when the weighting method is 'uniform', as 70.37%.Similar results can be seen in Table 4 and Table 5 when k = 2 and k = 3 when the weighting function is distance and accuracy.

22	**0**	**0**
8	**15**	**0**
2	**6**	**1**

Table 3 : Confusion Matrix of k-NN Classification when k = 2 (weighting method = 'uniform')

19	**2**	**1**
6	**15**	**2**
1	**4**	**4**

Table 4 : Confusion Matrix of k-NN Classification when k = 3 (weighting method = 'uniform')

22	**0**	**0**
1	**22**	**0**
1	**0**	**8**

Table 5 : Confusion Matrix of k-NN Classification when k = 2 (weighting method = 'distance')

22	**0**	**0**
1	**22**	**0**
1	**0**	**8**

Table 6 : Confusion Matrix of k-NN Classification when k = 3 (weighting method = 'distance')

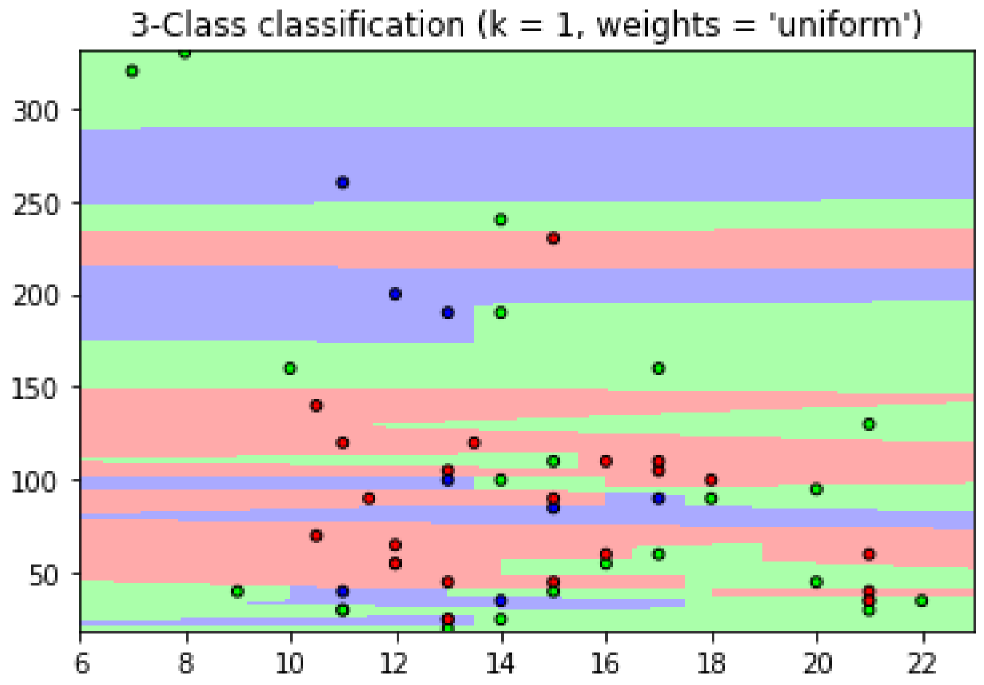

Figure 1: k-NN classification result with k = 1 (weighting = ‘ uniform’)

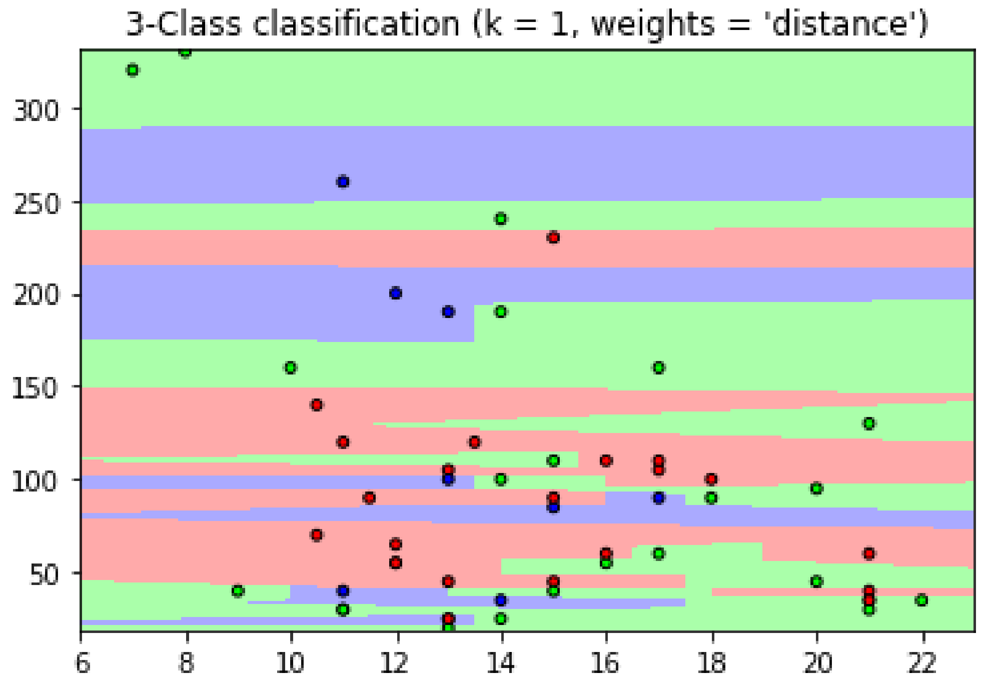

Figure 2 : k-NN classification result with k = 1 (weighting = ‘ distance’)

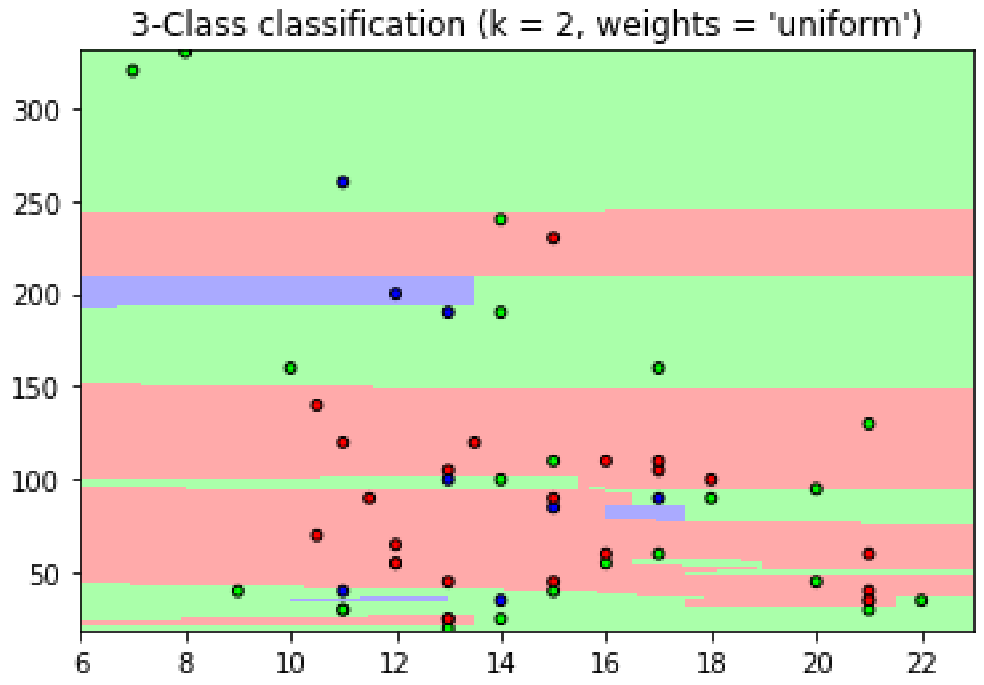

Figure 3: k-NN classification result with k = 2 (weighting = ‘ uniform’)

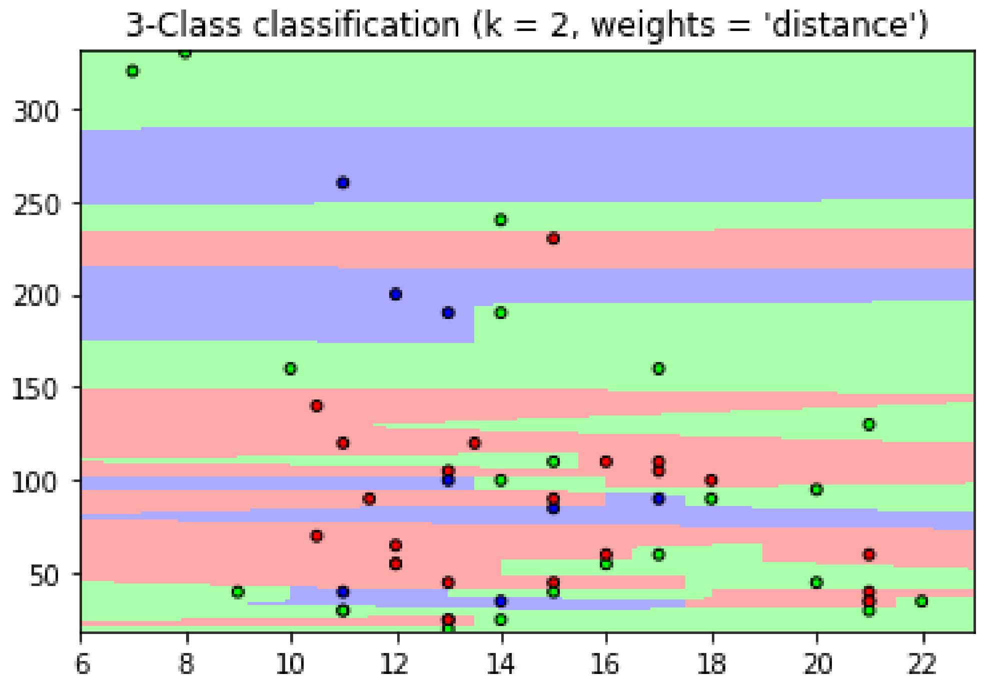

Figure 4: k-NN classification result with k = 2 (weighting = ‘ distance’)

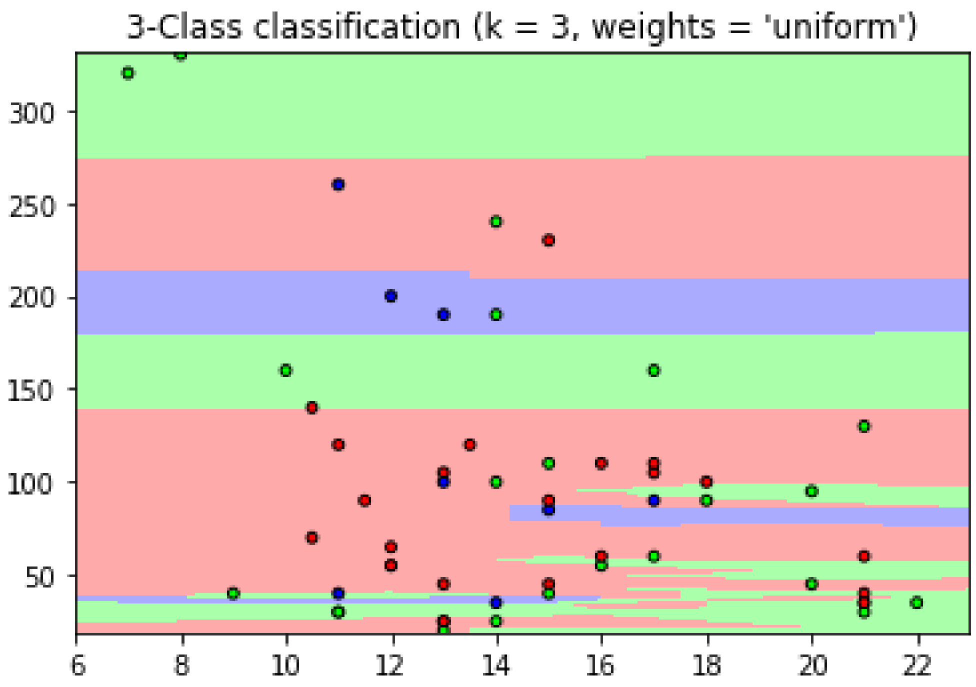

Figure 5: k-NN classification result with k = 3 (weighting = ‘ uniform’)

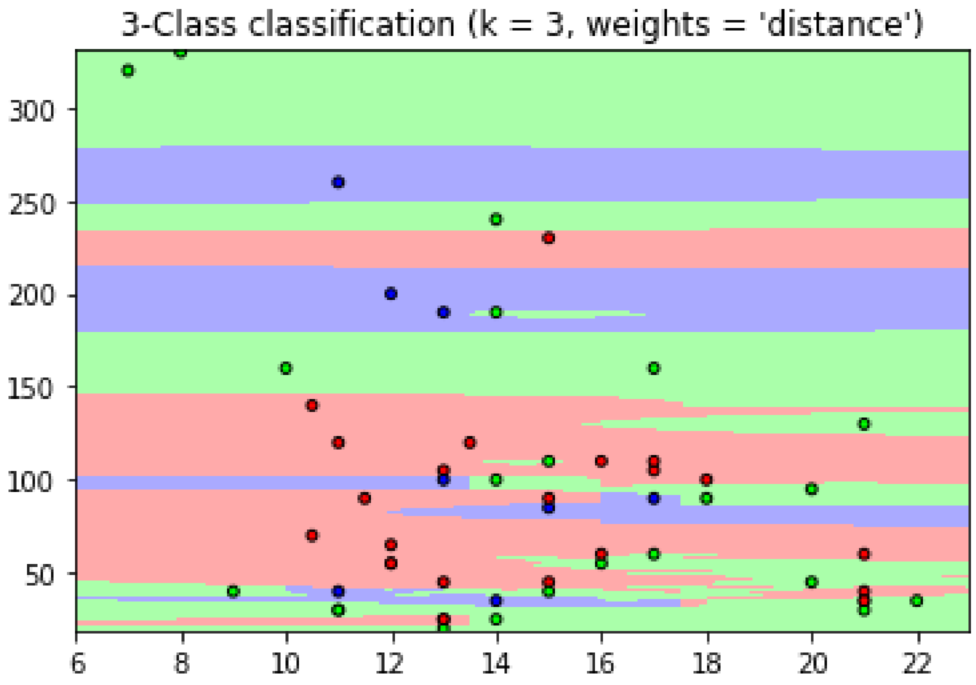

Figure 6: k-NN classification result with k = 3 (weighting = ‘ distance’)

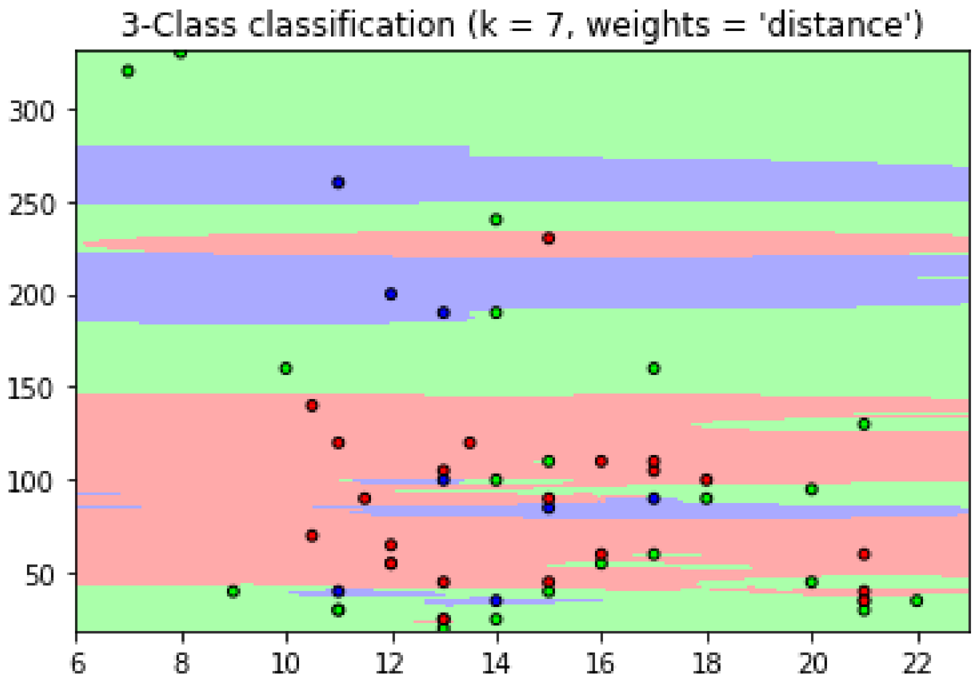

Figure 7: k-NN classification result with k = 7 (weighting = ‘ distance’)

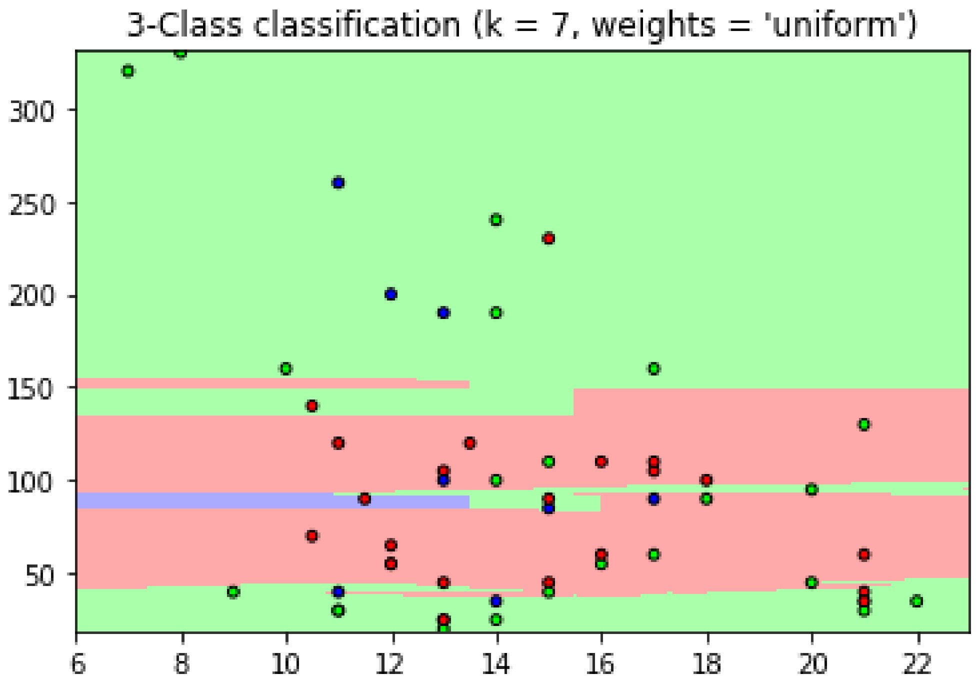

Figure 8: k-NN classification result with k = 7 (weighting = ‘ uniform’)

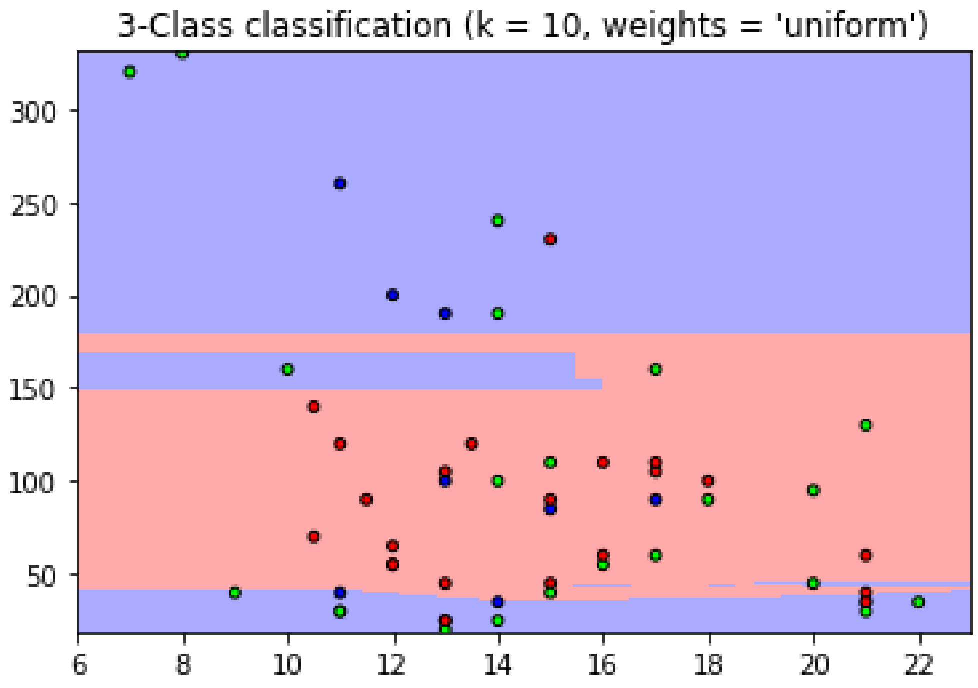

Figure 9: k-NN classification result with k = 10(weighting = ‘ uniform’)

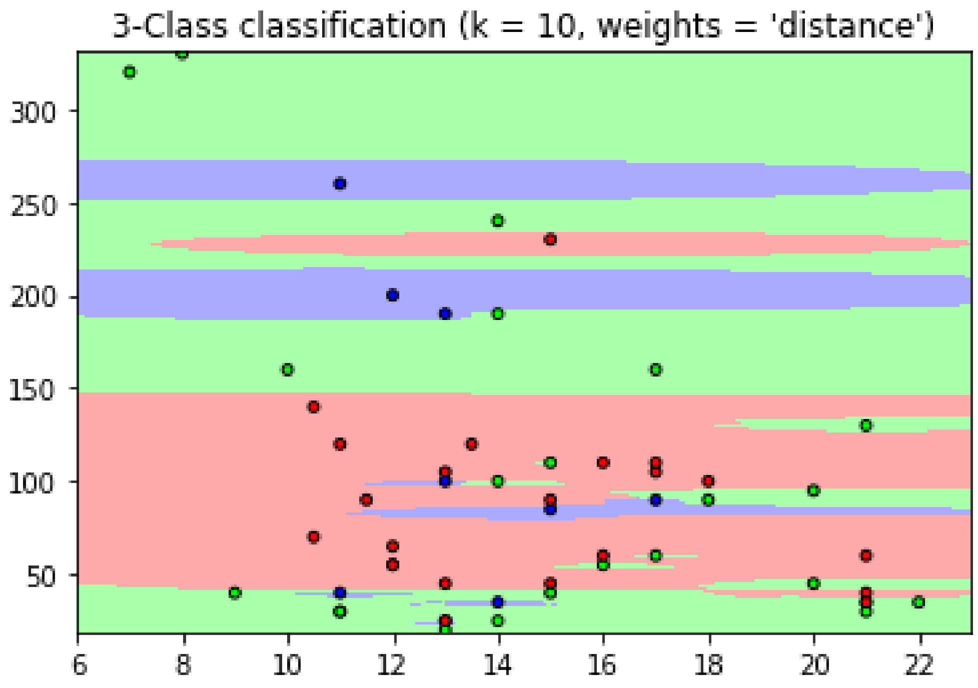

Figure 10: k-NN classification result with k = 10(weighting = ‘ distance’)

Conclusion:

In this project, k-NN classifier is used in order to classify variety of cereals by their manufacturer based on the data on their carbohydrate and potassium amount.From the results obtained,it can be said that the weighting method *distance* is the most precise one to use in this project because the accuracy of the classifier(with 96.29% accuracy), as shown in Table 2, does not change as *k* value changes.However as *k* increases,if the weighting method *uniform* is used,the accuracy of the classifier decreases down to about 59.26% from 96.29%.These accuracy results are also confirmed from confusion matrices shown in the results section by calculating true-positive,false-positive,false-negative,true-negative.

References:

[1] *sklearn.neighbors.KNeighborsClassifier*. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier

[2] *sklearn.metrics.confusion_matrix*. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

[3] *80 Cereals*. [Online]. Available: https://www.kaggle.com/crawford/80-cereals

Appendix A:

Python Code:

```
##################################################################################
#  Burak Koryan | burak@koryan.ca | http://koryan.ca
#  Project : Classification of different brands of cereals using k-NN #.classifier in Python
#  Date : May 10 2018
#################################################################################
```

# Step 1: Importing libraries

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets
import pandas as pd
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

#Step 2: Choosing the value of k and assigning X and Y axeses.

```
n_neighbors = 3
dataset = pd.read_csv('cereal.csv')
X = dataset.iloc[:, [8,10]].values
y = dataset.iloc[:, 1].values

# Step 3: Count how many 'K','G','P' are there in the data array
# in order to make new arrays for data and label.

j = 0
for i in range (0,77):
    if y[i] == 'K' or y[i] == 'G' or y[i] == 'P':
        j = j+1

new_data = np.zeros((j,2))
new_let = [0] * j
j = 0

for i in range (0,77):
    if y[i] == 'K' or y[i] == 'G' or y[i] == 'P':
        new_data[j] = X[i]
        new_let[j] = y[i]
        j = j+1
y_set = preprocessing.LabelEncoder()
y_fit = y_set.fit(new_let)
y_trans = y_set.transform(new_let)
#Step 4:
h = .02
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
```

# Step 5: k-NN classification,fitting,and predicting done here.

```
for weights in ['distance']:
    # we create an instance of Neighbours Cylassifier and fit the data.
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(new_data,y_trans)

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = new_data[:, 0].min() - 1, new_data[:, 0].max() + 1
    y_min, y_max = new_data[:, 1].min() - 1, new_data[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```

# Step 6: Plotting prediction results here

```
    # Plot also the training points
    plt.scatter(new_data[:, 0], new_data[:, 1], c=y_trans, cmap=cmap_bold,
                edgecolor='k', s=15)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title("3-Class classification (k = %i, weights = '%s')"
              % (n_neighbors, weights))
plt.show()
```

# Step 7 : Printing confusion matrix and calculating accuracy score here.

```
pred = clf.predict(new_data)
cm = confusion_matrix(y_trans,pred)
print accuracy_score(pred,y_trans)
print(cm)
```