KNN

Data Preprocessing

Training Data

```
import csv
import numpy as np
import pandas as pd

train = pd.DataFrame()
test = pd.DataFrame()

train = pd.read_csv('trainData.csv',  dtype={'Color': str,'Radius
(cm)':float,'Weight (grams)':float})

train = train.drop_duplicates()
train = train.replace(0,np.nan)
train = train.dropna() # drop all nan entiries

newCols = pd.get_dummies(train.iloc[:,0])
train = pd.concat([newCols, train], axis=1)
train = train.drop(train.columns[3], axis = 1)
min_radius = min(train.iloc[:,3])
max_radius = max(train.iloc[:,3])
train['Radius (cm)'] = list(map(lambda x: (float(x)-
min_radius)/(max_radius-min_radius), train.iloc[:,3])
min_weight = min(train.iloc[:,4])
max_weight = max(train.iloc[:,4])
train['Weight (grams)'] = list(map(lambda x: (float(x)-
min_weight)/(max_weight-min_weight), train.iloc[:,4]))

train
```

Test Data

```
test = pd.DataFrame()
test = pd.read_csv('testData.csv',  dtype={'Color': str,'Radius
(cm)':float,'Weight (grams)':float})



test= test.drop_duplicates()

newCols=pd.get_dummies(test.iloc[:,0])
test = pd.concat([newCols, test], axis=1)
test= test.drop(test.columns[3], axis = 1)

test['Radius (cm)'] = list(map(lambda x: (float(x)-min_radius)/(max_radius-
min_radius), test.iloc[:,3]))
```

```
test['Weight (grams)'] = list(map(lambda x: (float(x)-
min_weight)/(max_weight-min_weight), test.iloc[:,4]))

test
```

kNN Classification with Manual K

```
from math import sqrt

def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(len(row1)-1):
        distance += (row1[i] - row2[i])**2
    return sqrt(distance)

def get_neighbors(train, test_row, K):
    distances = list()

    for index, train_row in train.iterrows():
        dist = euclidean_distance(train_row,test_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    neighbors.clear()
    for i in range(K):
        neighbors.append(distances[i][0])
    return neighbors

def predict_classification(train, test_row, K):
    neighbors = get_neighbors(train, test_row, K)
    output_values = [row[-1] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction

def my_main_method(K):
    rows_list=[]
    for index, rw in test.iterrows():
        prediction = predict_classification(train, rw, K)
        rw[5]=prediction
        rows_list.append(rw)
    prediction_result = pd.DataFrame(rows_list)
    return prediction_result


try:
    K=int(input("Enter Value for K:"))
except ValueError:
    print("Error!!")
else:
    prediction_result = my_main_method(K)
prediction_result



from sklearn.metrics import precision_score
def calculate_Precision(y_true, y_pred):
```

```
    return precision_score(y_true, y_pred, average='micro')

y_true = prediction_result['Original (class)'].tolist()
y_pred = prediction_result['Predicted (class)'].tolist()

p=calculate_Precision(y_true, y_pred)
print('precision = %.1f when K = %.d' % (p * 100, K))
```

kNN Classification with Automatic K Estimation

```
K=1
max_K=len(train)
program_precision=0

try:
    Min_Precision=float(input("Enter minimum precision required:"))
    Min_Precision = Min_Precision/100
except ValueError:
    print("Error!!")
else:
    while program_precision < Min_Precision:
        prediction_result = my_main_method(K)

        y_true = prediction_result['Original (class)'].tolist()
        y_pred = prediction_result['Predicted (class)'].tolist()

        program_precision = calculate_Precision(y_true, y_pred)
        if program_precision < Min_Precision:
            K=K+1

            if K>max_K: # in order to avoid infinite loop
                print('precision = %.1f when K = %.d' % (program_precision
* 100, K))
                print('')
                print('****************************************')
                print('***********   Max K Reached   ***********')
                print('****************************************')
                break

            prediction_result = my_main_method(K)
            y_true = prediction_result['Original (class)'].tolist()
            y_pred = prediction_result['Predicted (class)'].tolist()

            program_precision = calculate_Precision(y_true, y_pred)
        print('precision = %.1f when K = %.d' % (program_precision * 100,
K))
```