1.check list

F1	Successfully load the dataset (in indataset())	√
	Show number of data entries (in indataset())	√
	Show the number of classes (in indataset())	√
	Show the number of data entries for each classes (in indataset())	√
	Show the minimum and maximum values for each feature (in indataset())	√
	Show the train dataset and test dataset split (in indataset())	√
F2	Successfully call library functions to train (in sklearnknn())	√
	save a model (in sklearnknn())	√
F3	implementation of an KNN algorithm (in class selfknn() and selectk())	√
	saved model (in main())	√
F4	output the train and test errors for both models (in getAccuracy() and sklearnknn() and compare())	√
F5	query the models by changing the input (in plot() and main())	√

2.How to run

At first just run program, and waiting for the two KNN model finished. Then the console will ask you which index of testset you want to show. Just like this

Please enter 0~539 , and then press enter,

```
========================================
Enter test number for plot between 0-539
```

Then enter 1 or 2 to choose model

```
Please choose model to predict
1   is sklearnknn
2   is selfknn
```

you will get the pictures and predict.

3.Explain the every part

F1: through def indataset(), splitdataset(), and main(), import sklearn method, the program implement Optical recognition of handwritten digits dataset, and then split the dataset into X_train, X_test, y_train, y_test, which are mean that training dataset features, test dataset features, training dataset labels, testing dataset labels.

F2: through def sklearnknn(), main(), import sklearn, joblib method, the program implement

dataset training, predict, model save and load, then print training and testing accuracy by import sklearn method.

F3: through class selfknn(), selectk(), main(), import joblib and numpy, the program implement dataset training, predict, model save and load, then print training and testing accuracy, select best K by my own knn algorithm.

F4: def getAccuracy() and compare(), through compared with predict labels and test labels, count the equivalence time, add it and divide length of test. Then compare train accuracy and test accuracy, print it.

F5: through def plot(), main(), import matplotlib.pyplot, the program implement query the testset and predict it then draw the pictures for user.

Additional requirement1: through dump the two model, program results of functionalities f1, f4, and f5 have already by loading the saved models, without calling the training functionalities f2 and f3. You can confirm it through delete the def main() statement:

```
141     sklearnknn(X_train,X_test,y_train,y_test)  #import sklearnknn function
```

and ,

```
155     clf=selfknn() #instantiation class selfknn
156     #save and load knn model as selfknn.joblib
157     dump(clf, 'selfknn.joblib')
```

Additional requirement2:
F2: through def sklearnknn(), main(),the program implement dataset training(use KNeighborsClassifier and knn.fit), predict(in def compare()), model save(dump) and load, then print training and testing accuracy by import sklearn method and joblib

F3: the all knn algorithm is in def knnalgorithm().

- Euclidean Distance: At first, import training data and label, test data, then get training data length of the column of the matrix. Through np.tile() copy test data that it have the same number of rows with train data. Then subtract the train data matrix and square. Then add all so that matrix have only 1 column. Then sqrt the 1 column matrix. (through this method can reduce the compute time greatly compared with many for loop)

```
def knnalgorithm(self,X_test, X_train, y_train, k):
    #Euclidean Distance
    dataSetSize = X_train.shape[0]              #get trainset fe
    # use np.tile() for copy test features matrix up to trainset
    diffMat = np.tile(X_test,(dataSetSize,1)) - X_train
    sqDiffMat = diffMat ** 2 #square
    # add all elements to the left in row
    sqDistance = sqDiffMat.sum(axis=1)
    distance = sqDistance ** 0.5 #sqrt
    # use np.argsort() sorted index
```

- Sort the Euclidean Distance the back sorted index by np.argsort (it just back the index rather than real data)

```
sortedIndex = distance.argsort()
```

- Find the k-nearest neighbor: through for loop to find k nearest neighbor label as votelabel, and then count the labels appear as classcount[votelabel], then through .keys() sort it with descending order and return the first one.

```
classCount = {}  #create a list for label counts
for i in range(k):
    # use sortedIndex to get it labels
    voteLabel = y_train[sortedIndex[i]]
    # count the labels
    classCount[voteLabel] = classCount.get(voteLabel,0) + 1
# sort the counts and return most labels
sortedClassCount = sorted(classCount.keys(),reverse=True)
return sortedClassCount[0]
```

Select k in F3:

Through for loop in 1 to 10 to run class selfknn(), set a accuracy list, and add each accuracy in list, then choose the max test accuracy and find the k in this case. Then print the score.

4.The program result:

```
Show the dataset datails
************************    ********************
the number of data entries                   1797
classes                                        10
samples per class                            ~180
dimensionality                                 64
features                              MAX:16  MIN:1
************************    ********************

*******SklearnKNN START*******
Training Score: 0.9896579156722355
Testing Score: 0.9851851851851852
Traing accuracy>Test accuracy
SklearnKNN time: 0.578125
*******SelfKNN START************
When k= 1 Testing score MAX, which are
Training Score: 1.0
Testing Score: 0.9907407407407407
Traing accuracy>Test accuracy
SelfKNN time: 3.953125
****************************************
Enter test number for plot between 0-539

8
Please choose model to predict
1   is sklearnknn
2   is selfknn


1
```