CS385 : Assignment 2

Topics covered: KNN Algorithm, train-and-test split, k-fold cross-validation, similarity measures

Deliverables:
In this assignment, you are given 2 functions in a2.py. In section 4 tasks, you'll find the purposes of those functions.
Your submission for this assignment should be an archive of two files, named `knn.py` and `yourusername_report.pdf`. knn.py should include functions listed in the following table:

Function name	Input	type	Output	type
dataNorm	X: the loaded dataset	2D array	X_norm: the normalized dataset	2D array
splitTT	X_norm: normalized dataset	2D array	X_split: a list contain two elements, the first is X_train in type of 2D array, the second is X_test in type of 2D array	list
	PercentTrain: expected portion of dataset for training, the domain is (0..1)	float		
splitCV	X_norm: normalized dataset	2D array	X_split: a list contain k elements, each element is in type of 2D array	list
	K: k-fold	int		
KNN	X_train: dataset for training	2D array	Accuracy	float
	X_test: dataset for testing	2D array		
	K: number of Nearest neighbors	int		
KNNManhattan	Same as KNN			
KNNMinkow	Same as KNN			

`yourusername_report.pdf` should follow the outline described in Section 3. You should put two files in a directory called `cs385_yourusername_2` and zipping the directory into a zip file called `cs385_yourusername_2.zip` etc. In addition to the two files, your zip file should contain a folder named `output`, where your output files are stored. Failure to follow conventions will result in penalties in marks.

Objectives:
- To get familiarized implementing your first machine learning algorithm, which is KNN from the scratch (i.e., without using any machine learning API).
- To apply KNN algorithm to classify the age of Abalone using Abalone dataset.
- To get familiarized on evaluating the performance of a machine learning algorithm.

1. KNN Algorithm

KNN is the simplest and yet powerful machine learning algorithm used for classification tasks. This is called as instance-based learning algorithm, because they simply store the training data in memory and

when a new query data is provided, a set of similar instances are retrieved from the training data and used to classify the new query instance. For more details on the KNN, refer to lecture notes. After implementing the algorithm, it should be evaluated using train-and-test split and k-fold cross validation method[1] as discussed in the class.

In this assignment, you will design the KNN algorithm to classify the age of the abalone using the abalone dataset provided by the UCI Machine Learning repository.

2. Dataset

Abalone dataset can be downloaded from this link[2]. The readme file in the dataset provides details on the dataset. Read this file before implementing your algorithm to check whether you need to do any data preprocessing steps. To give a brief introduction, there are totally 4,177 data observations in the dataset with 8 input attributes and 1 output variable. The input attributes are as follows:

1. Sex (Male (M), Female (F), or Infant (I))
2. Length
3. Diameter
4. Height
5. Whole weight
6. Shucked weight
7. Viscera weight
8. Shell weight
9. Rings (output)

3. Report Outline

The outline of the report is as follows.

- **a. Abstract:** Briefly mention the objective of this assignment and summarize the descriptions of the results/findings of this assignment in a concise manner.

- **b. Accuracy:** Report your findings of comparison on classification performance. You should follow the requirements in section 4.4.

- **c. Running time:** Report your findings of comparison on classification running time. You should follow the requirements in section 4.5.

- **d. Similarity measures:** Report your findings of performance and running time comparison on different similarity measures. You should follow the requirements in section 4.6.

4. Your tasks

- **a.** Use the function loadData() (which is given in a1.py) to load data from file. This command *X = loadData('abalone.data')* returns an array with size 4177*9. You can use *X.shape* to check its size. In this function, the values of the first attribute are already converted into floats. 'M'→0.333, 'F'→0.666 and 'I'→1.000.

[1] https://en.wikipedia.org/wiki/Cross-validation_(statistics)#k-fold_cross-validation

[2] https://archive.ics.uci.edu/ml/datasets/Abalone

b. Normalize the dataset. You are going to normalize the 8 input attributes by writing a function **dataNorm()**. This function takes in the array *X* and return the array *X_norm* (with 8 normalized input attributes plus 1 non-normalized output attribute). For each attribute, *max* is the maximal value and *min* is the minimal. The normalization equation is: (*data-min*)/(*max-min*). You can run *testNorm([X_norm])* (which is given in a1.py) to check the *mean* and *sum* for each attribute.

		Mean	Sum
Col1	Sex	0.47750066	1994.52023988
Col2	Length	0.60674608	2534.37837838
Col3	Diameter	0.59307774	2477.28571429
Col4	Height	0.12346584	515.71681416
Col5	Whole weight	0.29280756	1223.05719851
Col6	Shucked weight	0.24100033	1006.65837256
Col7	Viscera weight	0.23712127	990.45556287
Col8	Shell weight	0.2365031	987.87344295
Col9	Rings(output)	9.93368446	41493

c. Split the dataset into training and test set by (a) using train-and-test split and (b) using k-fold cross-validation method. Note the k-value here is different from the K-value in KNN algorithm. Choose the k value to be 5, 10 and 15.

Writing function **splitTT()**, it takes in the normalized dataset *X_norm* and the expected portion of train dataset *percentTrain*(e.g. 0.6), returns a list *X_split*=[*X_train,X_test*]. You can run *testNorm(X_split)* to check if the splitting is correct (e.g. no data loss in split). The output should be the same as the one in task2 (which is shown in the table).

Writing function **splitCV()**, it takes in the normalized dataset *X_norm* and *k*, returns a list *X_split*=[*X1,X2,…,XK*]. Again, you can run *testNorm(X_split)* to make sure the split is correct.

d. Implement the KNN algorithm by writing function **KNN()** as discussed in class. You can use Euclidean distance for similarity measure of any two samples.

KNN() takes in the training dataset *X_train*, test dataset *X_test*, and the number of nearest neighbors *K*, returns the accuracy of classification.

For each of the method (and folds) in task 3, compare the classification performance using **KNN()** for different values for K = 1, 5, 10, 15, 20 (here K is related to KNN algorithm) by filling the following **form** and giving the corresponding **plots** in your **report**.

Accuracy	Train-and test			Cross validation		
	0.7 - 0.3	0.6 – 0.4	0.5 - 0.5	5-fold	10-fold	15-fold
K=1						
K=5						
K=10						

K=15						
K=20						

e. Compare the computation time for the class prediction for each of the experiment in the task 3 by filling the following **form** and giving the corresponding **plots** in your **report**. You should also discuss on how the computation time varies for different experimental setups in your **report**.

Run-time	Train-and test			Cross validation		
	0.7 - 0.3	0.6 – 0.4	0.5 - 0.5	5-fold	10-fold	15-fold
K=1						
K=5						
K=10						
K=15						
K=20						

f. Repeat the task 4 and 5 with the following similarity measures and discuss the how the performance vary:

a. Manhattan distance (implement it in **KNNManhattan()**)
b. Minkowski distance with order 3 (implement it in **KNNMinkow()**)

5. Rubrics

This assignment is graded over total of 60 points. The breakdown is as follows:

- Data normalization is implemented correctly (10 points)
- Dataset is correctly split as mentioned in Section 4.3 (10 points)
- Correct implementation of KNN (10 points)
- Discussion on performance and computation measurements for each experimental setup presented in detail (10 points)
- Submission requirements must be met i.e., reasonable comments, proper format of the report (Not copy and pasted from the assignment specifications!), detailed explanation (using necessary equations, tables, figures, charts, etc.), correct documentation and file formats for the submission, etc. (20 points)