Design Report for k-NN Classification Algorithm Implementation

Name: Chris Dilger
Student ID: 101133703

Summary of Program

The program is an example of a foundational machine learning algorithm, variants of which have been used to classify Gastric Cancer, Wine Quality and Flower species (Li et al. 2012; Lichman 2013). In this implementation, the k-Nearest-Neighbour search will classify flowers to demonstrate practical applications. This implementation is general enough to give users the freedom to use a multitude of other datasets provided the input can be parsed by the CSV parser. This algorithm will be tested against the Iris dataset from the UCI Machine Learning Repository. In the case of Iris classification, a user in the field would be able to use this program to classify flowers of unknown species.

The complexity of the implementation is completely invisible to the end user. A user supplies data in a form analogous to CSV, which is then parsed by the k-NN search program. A user is then able to input key parameters which allow the k-NN to perform a classification on the object, returning a classification using the input data.

Example of input data

Input (iris.csv)

```
5.1,3.5,1.4,0.2,Iris-setosa
…
7.0,3.2,4.7,1.4,Iris-versicolor
…
6.3,3.3,6.0,2.5,Iris-virginica
…
```

Where the fields represent:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. classification

(Lichman 2013)

Given input data, say:

```
6.0, 2.7, 4.0, 1.3
```

we would expect an output:

```
Iris-versicolour
```

Data Dictionary

Description of the structures and custom data types used in the knn-search program.

Table 1: Classifier_List details

Field Name	Type	Notes
categories	my_string*	Stores the strings corresponding to each distinct category. This is an array type
num_categories	Int	Keep track of the number of categories in this list of categories

Table 2: Point details

Field Name	Type	Notes
dimension	float*	Keeps all of the floating point coordinates for the dimensions in each point. The number of dimensions in this array is dependent on the number of dimensions in the dataset.
category	int	Smallest possible way to represent the class of a point

Table 3: Dataset details

Field Name	Type	Notes
dimensionality	int	The number of dimensions the dataset is tracking. This would be 2 for a simple dataset with 2 variables, and for more complex datasets this could be 4, as for the Iris dataset.
num_points	int	Keep a record of the length of the array of points
Points	Point*	An array containing all of the points in the dataset

Table 5: Point_Neighbour_Relationship details

Field Name	Type	Notes
distance	float	The distance between the Comparision_Point to which this belongs, and the point this relationship is pointing to
neighbour_pointer	Point*	Reference to the neighbour it's pointing to

Table 6: Boolean enumeration details

Type	Enumerated Values	Notes
bool	false, true	Allows us to use true and false as it would be in other languages, as well as having it's own datatype

Overview of Program Structure

Data Structures

Of the types in the above data dictionary, attention needs to be drawn to some design choices that were made when designing the structs. Each has a brief overview which demonstrates how the struct is used in a broader program context, and how functions operate on or make use of the custom data types.

Point

Points of data in the kNN algorithm are one of the most fundamental units of computation. Each point contains classification information and a generic array of metrics the kNN uses to classify the points. In the Iris dataset, we have a relatively complex dataset with 4 dimensions, each corresponding to some physical measurement of the flower. This point is simply a representation of a point of n dimensions in a Euclidean space. This point has been kept as lightweight as possible because of the high number of operations involving points. The dimensionality of the point, is not stored with the point but with the Dataset type (see below) and the classification information is encoded with a single integer which is translated into strings by a set of classification functions (also below).

Dataset

A dataset holds a training dataset in memory at one time, containing all the points, number of points and the number of dimensions each point has. Each time a kNN classification is made, the training data must be supplied in the form of a Dataset. Dimensionality data is stored in one memory location that is a part of the Dataset, as well as the number of points. This ensures that points of different numbers of dimensions cannot be entered into the

dataset, as it would not make sense to compare a point in 3 dimensional Euclidean space with a point in 2 dimensional Euclidean space. When the accuracy of the kNN is measured, the Dataset has been designed so when a new dataset is created as a subset of the overall dataset, the points themselves don't have to be duplicated every time, and just the pointers are updated. This means that of the order of hundreds of datasets each with hundreds of points can be created with a light memory footprint.

Classifier List

A classifier list is a datatype that maps the integer category assigned to a point, with the string representation of the category specified in the data input file. When the CSV file is being parsed, the field containing the classification is passed to get_class_num, which if there is not already a class matching that string, will add a new class to the classifier list and return a new class integer for the point. This is an example of sequential and communicational coherence, as the data mapping from integer to string can be updated at the same time as returning a classification value for a new point of that class.

Comparison Point

A comparison point struct was created to store information about the nearest neighbours of the point being compared. A comparison point will have k nearest neighbours attached to it throughout the computation. The comparison point keeps a track of the distance to all of these nearest neighbours with a Point_Neighbour_Relationship, which has a reference to that nearest point, from which we can get the classification of each of those points. The knn_search function and the Mode Calculation sections describe this process in more detail. The comparison point was constructed to practice good information hiding, and only give functions the information about the implementation they need. Thus, instead of lumping all data into one point struct, another separate type was created.

Design Patterns in Practice – Mode Calculation

To calculate the mode of the categories a quicksort is performed on the k nearest neighbours, specified in a Comparison_Point list. Pointers to the points which are nearest neighbours are used to retrieve classification data. In order to classify a point, we need to know what the class of the majority of its k nearest neighbours are. Thus, a mode calculation is appropriate, the value occurring the most times will become the class of the new point. To do this, the function callback has been implemented to sort values based on integer size in the array of categories. The compare_int function is a function that is passed as an argument to another function, which is called within the function to determine the order of the elements. See the unit tests for the compare_int function documentation for how this works in practice. The qsort function is part of the standard library. Once the array is sorted, each value is counted and the resulting mode returned.

Euclidean distance

A function that will perform the core operation inside of the k-NN algorithm. From (Anon n.d.), the following algorithm will be implemented:

$$D_{ij}^2 = \sum_{v=1}^{n} \left(x_{vi} - x_{vj}\right)^2$$

(Anon n.d.)

This will involve some simple summation, some for looping and accessing members of arrays. Point typed variables are passed to the distance function, along with the dimensionality of the points. Since this function is cohesive and decoupled, it would be almost trivial to change the Euclidean distance function to a cosine, Chi square, and Minkowsky (Hu et al. 2016).

knn_search function

The kNN search function is the function that actually classifies the data. It is responsible for using the training dataset to find an appropriate class for a given Comparison_Point. See the structure chart for specific functions, however in general the kNN classification involves

1. Measuring the distance between the point to be classified and all of the other points
2. Picking a number, k of points
3. Finding the most common class of those points
4. Returning this classification

This is a very simplified overview of the implementation, which is much more complex when memory management, information hiding and other design considerations are taken into account. For an intuitive understanding of how the kNN works in practice, refer to the 1 dimensional unit tests for the knn_search function in the tests.c file.

Parsing CSV

Parsing inputs is not a trivial task, especially when multiple methods of input are considered. The CSV parser takes a filename, and will read the number of lines and fields before making a Dataset to hold all of the points expected. After the Dataset is initialised, a buffer reads in a line of the CSV file which is tokenised, separated by the “,” token. This tokenisation step in an earlier implementation unknowingly modified the buffer itself, as it was being passed by reference. This is a valuable example for when a pass by value is superior to a pass by reference approach, to avoid building coupled calling functions which are aware of the implementation details of the parse_point function.

User Input

Since the primary function of a kNN algorithm is to process huge datasets to create meaningful classifications, the user interface should hide as much of the implementation as possible. Thus, the success of the user interface can be measured in proportion to it’s simplicity and intuitive practice. Each input is validated and only a datafile and single point need to be supplied for the classification algorithm to return a result. The user is not required to input values to set the length of arrays, the program manages all this implementation without the user’s knowledge.

Experimental results for research

In order to facilitate the measurement of the accuracy of kNN algorithm for different k values, a benchmarking suite was developed to use the known parts of the training data to compare kNN classifications with the real data. Since the kNN algorithm is a so-called lazy learning algorithm, all operations on training data need to be repeated for every classification. Thus, there is little overhead introduced when a large number of datasets are produced to measure the accuracy of the classification model. An evaluate_knn function

returns the percentage classification accuracy. One point is removed, a new dataset created and the known classification is compared to the known classification. This is then repeated for every point in the dataset, to obtain a fraction of the points which were correctly classified. This allows us to know how accurate the kNN algorithm is for a given dataset, which is important when evaluating the effectiveness of the classification algorithm.

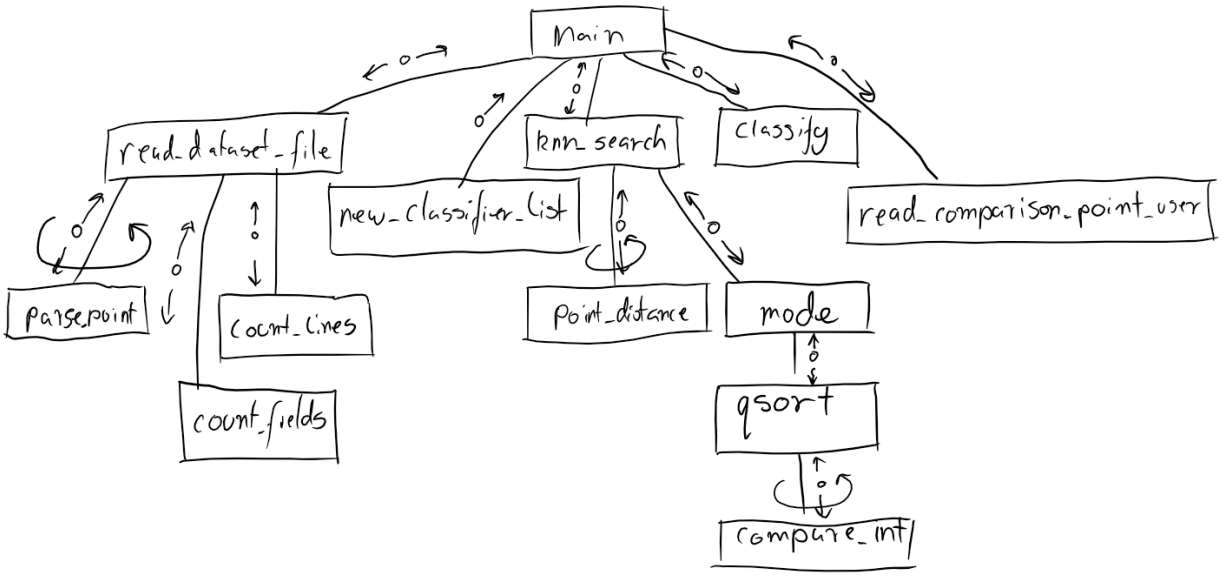
Unit tests

A unit testing library, greatest.h (Vokes 2017) provides a minimalistic framework with which Test Driven Development is implemented. This allows unit tests to be run to ensure that the minimum and correct functionality is achieved at a functional level. See the short video for a practical approach to Unit Testing.

Unit testing is a software development process which delivers fast feedback when implementing functionality, encourages good encapsulation and abstraction by explicitly specifying functional requirements and ensures changes to the codebase don't inadvertently introduce unnoticed changes. (Gary Bernhardt n.d.) This means the approach to writing functions is completely different to the more organic approach demonstrated in the course materials. When requirements can be specified in a structure chart and planning documentation like data dictionaries and other outlines are created, TDD gives an assurance that the code delivered is the code the design specified. The functional decomposition of this program was heavily influenced by the Test Driven Design development process.

While TDD is an effective methodology, there are places in the program in which TDD was not used for code clarity reasons. Whilst all code should be unit tested, most of the code dealing with IO or user input is not unit tested, because the input is arbitrary. Though as (Roland Cuellar 2006, p. 7) states, "If you can't write a test for what you are about to code, then you shouldn't be coding it." In the future custom IO functions would be written to support unit testing.

Structure Chart



References

Anon n.d., 'A Complete Guide to K-Nearest-Neighbors with Applications in Python and R', viewed 8 May, 2017a, <https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/>.

Anon n.d., 'c++ - how to completely disable assertion - Stack Overflow', viewed 25 May, 2017b, <https://stackoverflow.com/questions/5354314/how-to-completely-disable-assertion>.

Anon n.d., 'INFO: strtok(): C Function -- Documentation Supplement', viewed 31 May, 2017c, <https://support.microsoft.com/en-us/help/51327/info-strtok-c-function----documentation-supplement>.

Anon n.d., '"n"-Dimensional Euclidean Distance', viewed 8 May, 2017d, <https://hlab.stanford.edu/brian/euclidean_distance_in.html>.

Anon n.d., '"Re: Regression testing in OpenBSD" - MARC', viewed 25 May, 2017e, <http://marc.info/?l=openbsd-ports&m=139474670315494>.

Datar, M, Immorlica, N, Indyk, P & Mirrokni, VS 2004, 'Locality-sensitive hashing scheme based on p-stable distributions', *Proceedings of the twentieth annual symposium on Computational geometry*, ACM, pp. 253–262, viewed 24 May, 2017, <http://dl.acm.org/citation.cfm?id=997857>.

Dong, W, Moses, C & Li, K 2011, 'Efficient k-nearest neighbor graph construction for generic similarity measures', *Proceedings of the 20th international conference on World wide web*, ACM, pp. 577–586, viewed 19 May, 2017, <http://dl.acm.org/citation.cfm?id=1963487>.

Gary Bernhardt n.d., 'Destroy All Software', viewed 5 June, 2017, <https://www.destroyallsoftware.com/screencasts>.

Hu, L-Y, Huang, M-W, Ke, S-W & Tsai, C-F 2016, 'The distance function effect on k-nearest neighbor classification for medical datasets', *SpringerPlus*, vol. 5, no. 1, p. 1304.

Li, C, Zhang, Shuheng, Zhang, H, Pang, L, Lam, K, Hui, C & Zhang, Su 2012, 'Using the K-Nearest Neighbor Algorithm for the Classification of Lymph Node Metastasis in Gastric Cancer', *Computational and Mathematical Methods in Medicine*, vol. 2012, p. e876545.

Lichman, M 2013, *UCI Machine Learning Repository*, University of California, Irvine, School of Information and Computer Sciences, viewed <http://archive.ics.uci.edu/ml>.

Roland Cuellar 2006, *Test Driven Requirements*, viewed 6 June, 2017, <http://clearspecs.com/joomla15/downloads/ClearSpecs37V01_Test-Driven%20Requirements.pdf>.

Vokes, S 2017, *greatest: A C testing library in 1 file. No dependencies, no dynamic allocation. ISC licensed*, C, viewed <https://github.com/silentbicycle/greatest>.