Department of Computing and Information Systems
The University of Melbourne
COMP30027, Machine Learning, 2017

Project 1: K-Nearest Neighbour Classification

Due:	Wednesday 29 March 11am
Submission:	Source code (in Python) and Report (as a PDF file) to Turnitin on the LMS
Marks:	The project will be marked out of 20, and will contribute 20% of your total mark.
Groups:	You may choose to form a group of 1 or 2. Groups of 2 will have higher expectations (elaborated below).

Overview

In this project, you will implement a classifier using the K-nearest neighbour algorithm, apply it to a provided data set, and evaluate the performance of the classifier using a meaningful machine learning evaluation framework.

Programming a K-NN classifier

You should implement your own K-Nearest Neighbour classifier using Python, using the following prototypical data types:

- An **attribute value** is a raw string (for nominal attributes) or floating point number (for continuous attributes), as specified in "Data" below
- An **instance** is a list of attribute values
- A **data set** is a 2-tuple, comprising a list of training instances (logically equivalent to a list of lists of attribute values), and a list of class labels (one string per training instance)
- A **training data set** is a data set where you will use the class labels as part of the model
- A **test data set** is a data set where you will use the class labels only in the evaluation phase

A minimal submission should comprise the following functions, which use the specified prototypes:

- `preprocess_data(filename)` should open the file given by the string `filename`, and return a data seta (with suitable class labels, as discussed in the "Data" section below) comprised of the instances in the file, one per line.
- `compare_instance(instance, instance, method)` should return a score based on calculating the similarity (**or** distance) between the two given instances, according to the

similarity (or distance) metric defined by the string `method`. You should specify the values that `method` can take in comments — according to the similarity (or distance) metric(s) that you have implemented.

- `get_neighbours(instance, training_data_set, k, method)` should return a list of `(class, score)` 2-tuples, for each of the k best neighbours (according to the similarity (or distance) metric `method` specified above) for the given instance from the test data set, based on all of the instances in the training data set.

- `predict_class(neighbours, method)` should return a predicted class label, according to the given neighbours defined by a list of `(class, score)` 2-tuples, and your chosen voting method. You should specify the values that `method` can take in comments — according to the voting method or methods that you have implemented.

- `evaluate(data_set, metric)` should return the calculated value of the evaluation metric, based on dividing the given data set into training and test splits using your preferred evaluation strategy (more below). You should specify the values that `metric` can take in comments — according to the evaluation metrics you have implemented; at least one valid value must be `"accuracy"`.

Your submission should run end-to-end (using suitable defaults where necessary) using the following function call:

```
>>> evaluate(preprocess_data("/path/to/abalone.data"), "accuracy")
```

Note that some of these components have pre-implemented versions, for example, similarity metrics in the `numpy` package or k-Nearest Neighbour and evaluation metrics in the `scikit-learn` package. **You should not use them in this project**; we are asking you to implement them yourself(ves) from first principles.

Groups

If your group only contains one member:

- You will need to preprocess the data, and build the core k-Nearest Neighbour classifier — you should examine the behaviour of the method as you vary k

- You should implement one (1) or more similarity (or distance) metrics, which you will use to find the nearest neighbours

- You should implement one (1) or more voting methods, to predict a class based on a list of nearest neighbours (when $k > 1$)

- You should implement two (2) or more evaluation metrics, at least one of which should be Accuracy

- You should aim to use a Hold-out strategy, which randomly partitions the data in training and test sets (and/or Cross–Validation, if you like)

If your group contains two members:

- You will need to preprocess the data, and build the core k-Nearest Neighbour classifier — you should examine the behaviour of the method as you vary k
- You should implement two (2) or more similarity (or distance) metrics, which you will use to find the nearest neighbours — you should examine how the behaviour of the method changes with different metrics
- You should implement two (2) or more voting methods, to predict a class based on a list of nearest neighbours (when $k > 1$) — you should examine how the behaviour of the method changes with different voting methods
- You should implement three (2) or more evaluation metrics, at least one of which should be Accuracy
- You should aim to use a Cross–validation strategy, which randomly partitions the data in training and test sets

Data

For this project, we will adopt one of the datasets available from the UCI machine learning repository (`https://archive.ics.uci.edu/ml/index.html`). Specifically, we will experiment with the `Abalone` dataset (`https://archive.ics.uci.edu/ml/datasets/Abalone`).

This dataset consists of 8 attributes, plus the attribute to be predicted (Rings = Age):

1. Sex / nominal / – / M, F, and I (infant)
2. Length / continuous / mm / Longest shell measurement
3. Diameter / continuous / mm / perpendicular to length
4. Height / continuous / mm / with meat in shell
5. Whole weight / continuous / grams / whole abalone
6. Shucked weight / continuous / grams / weight of meat
7. Viscera weight / continuous / grams / gut weight (after bleeding)
8. Shell weight / continuous / grams / after being dried
9. Rings / integer / – / +1.5 gives the age in years

In principle, the target variable (Rings) has 29 different values, making this a 29-class classification problem. However, many of these classes have very few instances and this may make precise classification difficult. Therefore we can explore two variants of this task, Abalone-2 and Abalone-3. (Challenge question: why did we choose the thresholds below?).

- **Abalone-2:** A two-class classification: `young` (Rings ≤ 10) and `old` (Rings ≥ 11)
- **Abalone-3:** A three-class classification: `very-young` (Rings ≤ 8), `middle-age` ($9 \leq$ Rings ≤ 10), and `old` (Rings ≥ 11)

Individuals may choose just one of these tasks. Groups should work with both tasks.

Task 3: Report

Groups of one member should write a short report of 600–1000 words that summarises your key insights.
Groups of 2 should submit a longer report, of 900–1,500 words.

The report should discuss:

1. Your choice of similarity metric(s).
2. A description of the validation framework you have adopted.
3. Your recommendations for this data set. What is
   (a) an effective representation ?
   (b) a good similarity metric ?
   (c) a good value of "k" ?

   Justify these choices based on your experiments.

Note that we will be looking for evidence that you have thought about the task and have determined reasons for the performance of the method.

While 1000 words might seem long at first glance, it isn't: being concise will be invaluable, and overlong reports will be penalised. You will not have space to discuss the technical elements of your implementation; you should focus primarily on the knowledge you have gained in your experiments. Spending more time on your report will allow you to use the word limit more effectively.

Submission

Submission will entail two parts, via links on the LMS that will be made available one week before the deadline.

1. A single Python file containing your code.
2. Your written report, as a single file in Portable Document Format (PDF).

If your report is submitted in any format other than PDF, we reserve the right to return it with a mark of 0.

Assessment

The mark breakdown for the project will be spit between the programming component, and the report.

12 Code

- 3 Manipulation of the data, including file input, generating the class labels, and suitable use of data structures
- 6 Method, including the core k-NN algorithm, similarity metric(s), and voting procedures
- 3 Evaluation, including partitioning the data, and calculating the evaluation metric(s)

8 Report

- 2 Clarity and Presentation.
- 2 Observations.
- 4 Well-justified recommendations.

Changes/Updates to the Project Specifications

If we require any (hopefully small-scale) changes or clarifications to the project specifications, they will be posted on the LMS. Any addendums will supersede information included in this document.

Academic Misconduct

You are encouraged to collaborate with your peers in terms of the conceptualisation and framing of the problem. However, re-use of code, excessive influence in development, or collusion in the report will be considered cheating.
If you are working in a group of two, we of course expect you to work together and share code with your team mate. However, we do not expect to see any sharing of code between groups and therefore between submissions.
We will be checking submissions for originality and will invoke the University's Academic Misconduct policy (`http://academichonesty.unimelb.edu.au/policy.html`) where inappropriate levels of collusion or plagiarism are deemed to have taken place.