CITS5508 Machine Learning
Semester 1, 2023

Assignment 1

Assessed, worth 10%. Due: 5pm, Thursday 06th April 2023

All work is to be done individually.

1 Outline

In this assignment, you will develop Python code for a small classification tasks and implement the k-NN algorithm for a regression task. You will also learn the *Markdown* syntax to put comments and explanation in your Jupyter Notebook file to improve the presentation of your work.

2 Submission

Name your Jupyter Notebook file as **assignment1.ipynb** and submit it to LMS before the due date and time shown above. You can submit your file multiple times. Only the latest version will be marked.

3 Part 1 - Exploring ML techniques for classification tasks (30 marks)

In this part, you will apply the concepts discussed in the lectures to perform a classification task using different ML techniques.

3.1 Dataset

For this part of the assignment, we will use the *training.csv* and *testing.csv* data files supplied on LMS for this assignment. These files were downloaded and slightly modified from the **Forest type mapping dataset** on the UCI Machine Learning website. Please look at the link below:

http://archive.ics.uci.edu/ml/datasets/Forest+type+mapping#

for the description about this dataset. Please do not download the original dataset from the UCI ML website above. You should use the two *csv* files supplied on LMS for this labsheet. You should save both *csv* files to the same directory with your *assignment1.ipynb* file.

The training set (*training.csv*) contains 325 instances instances of multivariate remote sensing data of some forest areas in Japan. There are 4 different forest types labelled in the first column (the column heading is '*class*'), as described in the link above. The test set (*testing.csv*) has the same format as *training.csv* and contains 198 test instances.

3.2 Tasks - Binary classification

Your tasks for this part of the assignments are:

1. Read in the contents of both csv files[1]. Inspect what the columns are by displaying the first few lines of the file.
2. To simplify the classification task, write Python code to remove all the columns whose names begin with `pred_minus_obs`. You should have only 9 features (`b1`, `b2`, $\cdots$, `b9`) left for both the training and test sets.
3. Use appropriate functions to display (visualise) the different features (or attributes/columns). Display some plots for visualising the data. Describe what you see in your markdown cells.
4. Write Python code to count the number of instances for each class label. Do you have an imbalanced training set?
5. Perform an appropriate feature scaling step before doing the classification. You can use `MinMaxScaler`, `StandardardScaler`, or any suitable scaling function in the `sklearn.preprocessing` package. You can also write your own feature scaling code if you prefer. Whichever way, ensure that your feature scaling is applied to both the training data and the test data.
6. Use the *Logistic Regression Classifier* implemented in `sklearn.linear_model` class to perform binary classification using examples from two classes: 's' ('Sugi' forest) and 'd' ('Mixed deciduous' forest). You will need to write Python code to have only examples from these two classes in your training and testing set.
7. (a) Plot the estimated probabilities and decision boundary (as in Figure 4.23 of the textbook) of your Logistic Regression Classifier considering two individual features. You can choose which ones they will be. Therefore, you will need to create a Logistic Regression model for each of these two features and inspect the decision boundaries. Hence, you should provide two plots.

   (b) Now plot the estimated probabilities and decision boundary for a model considering all features. To build your plot, you must use the score value of the linear part from the logistic regression model in the x-axis. Then, you choose randomly 10 instances of the testing set, add then to your plot, and verify if you have made a right or wrong decision (how would you classify the test instances) regarding the decision boundary of 50%. Comment about your results.

(Use all features (b1-b9) to perform tasks 8-11)

8. Plot precision versus recall and comments on the results. How does the performance measure behave? What threshold would you choose and why?
9. Use the *k-nearest neighbours* (k-NN) algorithm for the same binary classification task. Try different values of k. Which value did you choose? Why?
10. Compare the performances of the two classifiers (Logistic Regression and k-NN) and give a brief discussion about your experimental results. You should show the confusion matrices, ROC curves and accuracies of the two classifiers for the testing set. Remember to discuss which threshold for the Logistic Regression you used to get the results.
11. Use 3-fold cross-validation and discuss the generalisation capacity of the two classifiers.

[1] Since both files are in the same directory as your Jupyter Notebook file, you should be able to read each one of them without any path name. For example, `pd.read_csv('training.csv')` should work just fine.

3.3 Tasks - Multiclass classification (20 marks)

Now we will use all classes in the "Forest type mapping Data Set" and features b1-b9.

12. Use the *Support Vector Machine Classifier* implemented in the `sklearn.svm.SVC` class to perform multiclass classification using the *one-versus-one* strategy. You should look at the Scikit-learn API for this class and experiment with two hyperparameters. You should use grid search and 3-fold cross validation to find the optimal values for these two hyperparameters that maximise the classification accuracy.

    For other hyperparameters, you can manually set them to some reasonable values. Apart from the Python code, you should explain what you carried out in markdown cells, e.g., which two hyperparameters you have tried? What combination of the hyperparameter values gave the highest classification accuracy?

13. Use the *Softmax Regression* and the k-NN algorithm (try with some different values of k) on the same classification tasks and comment about the results of the three techniques (SVM, Softmax Regression and k-NN).

4 Part 2 - Implementing the *k-nearest neighbours* (k-NN) algorithm to do regression (20 marks)

In this part, you will learn how to use the *k-nearest neighbours* (k-NN) algorithm to do regression. This is an *instance-based learning* method (see Figure 1-15 in the textbook for an example of k-NN classification). Suppose that the features $\mathbf{X}_i$, for $i = 1, \cdots, n$, of our training data are n-dimensional vectors and each feature vector has an associated y_i scalar value. Our objective is to predict the scalar y_{test} value for a given test instance $\mathbf{X}_{\text{test}} \in \mathbb{R}^n$. The way how the k-NN regression works can be summarized as follows:

1. finds the k nearest neighbours of the test instance $\mathbf{X}_{\text{test}}$ in the feature space ($\mathbb{R}^n$) . This step gives a set of k neighbours $\{\mathbf{X}_{(1)}, \mathbf{X}_{(2)}, \cdots, \mathbf{X}_{(k)}\}$, where k is a positive integer supplied by the user. From the training data, the associated scalar values $\{y_{(1)}, y_{(2)}, \cdots, y_{(k)}\}$ of these neighbours can be extracted.
2. computes y_{test} as the weighted sum of the y values of these neighbours, i.e., $y_{\text{test}} = \sum_{i=1}^{k} w_i y_{(i)}$.

In Step 1, we need to use a distance function that defines the *nearness* of points in the feature space. Distance functions that are commonly used include: Manhattan distance (ℓ_1 norm), Euclidean distance (ℓ_2 norm), and Minkowski distance (ℓ_p norm, for some integer $p > 2$).

In Step 2, the value of each weight w_i needs to be defined. The simplest formula is to set $w_i = 1/k, \forall i$, so y_{test} is just the simple average of $\{y_{(1)}, \cdots, y_{(k)}\}$. A more complex formula is to set w_i as the inverse of the distance or squared distance of each neighbour $\mathbf{X}_{(i)}$ to the test instance $\mathbf{X}_{\text{test}}$, i.e., let

$$w_i = \frac{1}{d(\mathbf{X}_{(i)}, \mathbf{X}_{\text{test}}) + \epsilon} \text{ or } w_i = \frac{1}{d^2(\mathbf{X}_{(i)}, \mathbf{X}_{\text{test}}) + \epsilon}$$

where $d(\cdot, \cdot)$ can be the Manhattan distance, Euclidean distance, Minkowski distance, or any user-defined distance between the two input items. The term ϵ in the denominator is a small constant to avoid the division-by-zero problem when $\mathbf{X}_{(i)}$ and $\mathbf{X}_{\text{test}}$ happen to be the same point and $d(\mathbf{X}_{(i)}, \mathbf{X}_{\text{test}})$ becomes zero. These more complex formulae allow neighbouring points that are closer to the test instance $\mathbf{X}_{\text{test}}$ to have larger weights. If the test instance $\mathbf{X}_{\text{test}}$ happens to coincide with one of the neighbours, then the weight for that neighbour would be 1 and the weights for all other neighbours would be 0.

After appropriately defining w_i, it is important to normalize all the weight values as follows:

$$w_i \leftarrow \frac{w_i}{\sum_{j=1}^{k} w_j}$$

so that they sum to 1.

The Scikit-Learn library includes the class `KNeighborsRegressor` that performs k-nearest neighbours regression. See

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html

The bottleneck of k-NN is in Step 1 – how to quickly find the k nearest neighbours. This is a computationally expensive process especially when the dimension n of the features is large and when the training set size is large also (having more training data is actually good for the regression task so we should not complain that training set is too large). The class `KNeighborsRegressor` includes the implementation of various algorithms (e.g., building a *ball-tree* or a *kd-tree* data structure) to speed up the neighbour search step.

Your task in this assignment is to apply the k-NN algorithm to the California Housing Prices dataset described in Chapter 2 to predict the house prices. You can use the parameter `n_neighbors` to set your desired k value. Your tasks for this part of the assignments are:

1. Randomly split the data into a training set (say 80%) and a test set (say 20%), apply the k-NN regressor, and evaluate how good the regressor performs by computing the *root mean square error* (RMSE) of the predicted house prices of the test set.

2. Experiment with a selection of columns (avoid those columns that contain text) from the dataset to form your feature vectors. The number of columns that you choose would become the dimension n of your feature vectors. Note that you would need to do some normalization so that your data is not dominated by some columns that have large magnitudes.

3. Try also setting the parameter `weights` to '`uniform`' (this is equivalent to setting $w_i = 1/k, \forall i$) and to '`distance`' (this is equivalent to setting $w_i = 1/(d(\mathbf{X}_{(i)}, \mathbf{X}_{\text{test}}) + \epsilon)$) and then compare the *root mean squared errors* (RMSEs) of their predictions.

Let $\{\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_n\}$ and $\{y_1, y_2, \cdots, y_n\}$ be the groundtruth instances and their corresponding ground truth house price values in the test set. Let $\{\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_n\}$ be the predicted house price values for these instances. Then the RMSE is given by:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}.$$

5 Presentation

For each of the tasks in your assignment, you should consider the following points to avoid losing marks when presenting your `ipynb` files:

- Present your `ipynb` file as a portfolio, with *Markdown* cells inserted at appropriate places to explain your code. See the following links if you are not familiar with

  Markdown:

- https://www.markdownguide.org/cheat-sheet/ (basic)
- https://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Typesetting%20Equations.html (more advanced)

- Divide your portfolio into suitable sections and subsections (with section and subsection numbers and meaningful headings) would make your portfolio easier to follow.
- Avoid having too many small *Markdown* cells that have only one short sentence. In addition to *Markdown* cells, some short comments can be put alongside the Python code.
- Use meaningful variable names.
- When printing out your results, provide some textual description so that the output is meaningful.
- Provide complete code for each of the tasks.
- Your code should run properly and efficiently.
- Provide comments about your code.
- Provide suitable visualisation of data and results.
- Provide comparisons that are meaningful and complete.
- Certify that the presentation of your Python notebook is good, and that you used the Markdown cells well.
- Results are presented and discussed.
- All the requirements stated in the specification of the tasks were covered.

6 Penalty on late submissions

See the URL below about late submission of assignments:
https://ipoint.uwa.edu.au/app/answers/detail/a_id/2711/~/consequences-for-late-assignment-submission