CITS5508 Machine Learning
Semester 1, 2023

Assignment 2

Assessed, worth 10%. Due: 5pm, Friday 05th May 2023

All work is to be done individually.

1 Submission

Name your Jupyter Notebook file as **assignment2.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.

2 Breast cancer wisconsin (diagnostic) dataset (30 marks)

In this labsheet, you are asked to train a few decision tree classifiers on the *Breast cancer wisconsin (diagnostic) dataset* available on Scikit-Learn and compare their performances.

This part of the assignment includes the following tasks:

1. Description about the *Breast cancer wisconsin (diagnostic) dataset* can be found in **Section 7.1.7** of the following web page:

   https://scikit-learn.org/stable/datasets/toy_dataset.html#breast-cancer-wisconsin-diagnostic-dataset

   There are two classes in the dataset:

   - *malignant* (212 instances, class value 0) and
   - *benign* (357 instances, class value 1).

   Follow the example code given on the web page:

   https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html#sklearn.datasets.load_breast_cancer

   to read in the dataset and separate it into a feature matrix `X` and a class vector `y`. Your feature matrix should have 569 (rows) × 30 (columns) and your class vector should have 569 elements.

2. Investigate whether some features can be dropped through some suitable visualisation of the data. For instance, if two features have a linear relationship, then it will be sufficient to keep only one of these features. If you use a different colour in the visualisation to show data points coming from each class, you will find that majorities of the points from the two classes are well separated. So we can assess that the classification task later on should not be too difficult.

   Determine from your visualisation what features can be dropped and write Python code to drop them from your feature matrix `X`.

   Split `X` and `y` into a training set and a test set using the split ratio 85/15 and `random_state=123`.

3. Investigate a few decision tree classifiers with different hyperparameter values as follows:

- Train a decision tree classifier using the default values for all the hyperparameters.
- Use the trained classifier to perform predictions on the training set and the test set. Compare the accuracy scores for the two sets.
- Do you think this classifier has an overfitting issue? Why?
- Follow the example code for Chapter 6 from the GitHub page for the textbook to display the decision tree built from the training process (like the one shown in Figure 6.1 of the textbook for the *iris* dataset). You will need to restrict the depth of the tree for the display (e.g., by setting `max_depth=3`); otherwise, the diagram will be too large. Study the diagram to see if it can help you to confirm whether the classifier has an overfitting issue.
- Construct a second decision tree classifier by setting the hyperparameter `max_depth=3`. Repeat all the steps above for this new classifier. Compare the accuracy scores for the training set and test sets with those from the previous classifier (note that your classification results may differ for different runs of the code).
- Construct a third decision tree classifier by setting the hyperparameter `min_samples_split=5`. Repeat all the steps above for this new classifier. Compare the accuracy scores for the training set and test set with those from all the previous classifiers.
- Construct a fourth decision tree classifier by setting the hyperparameter `min_samples_leaf=5`. Repeat all the steps above for this new classifier. Compare the accuracy scores for the training set and test set with those from all the previous classifiers.
- Use a 3-fold cross-validation and grid-search to find the best combination of these hyperparameters (`max_depth`, `min_samples_split` and `min_samples_leaf`). Report the results of the cross-validation. Compare the values of the best hyperparameters from the cross-validation with the ones you used before and comment on them.
- Use the best hyperparameters from the 3-fold cross-validation and report the training and test set results. Comments about the results and compare them with the previous classifiers. Think about what the dataset is about when commenting on the results. For the comparison, you can display the confusion matrices and performance metrics such as precision and recall for training and test sets.

3 Concrete Slump Test (20 marks)

The **Concrete Slump Test** dataset

https://archive.ics.uci.edu/ml/datasets/Concrete+Slump+Test

is a small dataset suitable for regression. For any new test instance, our aim is to use our trained model to predict its *28-day Compressive Strength* (Mpa) value. The *slump_test.data* dataset can be downloaded directly from the link below:

https://archive.ics.uci.edu/ml/machine-learning-databases/concrete/slump/

Although the name of the file ends with *.data*, it is actually a *csv* file.

NOTE: Save the downloaded file (*slump_test.data*) to the same directory with your Jupyter Notebook file. Do not rename or modify the file in any way.

This part of the assignment includes the following tasks:

- Firstly, inspect the data and perform data cleaning if needed. As described on the web page above. there are 7 input variables (i.e., feature columns) and 3 output variables. Our interest is the *28-day*

Compressive Strength output, so the other two output variables should be dropped. Perform some basic visualisation and determine whether any additional features should be removed also. You should write a Python function for this data cleaning step.

- Perform a 80/20 random split on the dataset to form a training set and a test set. For your Voting regressor, use the following as the 3 base estimators with default hyperparameters: (i) a linear SVM regressor, (ii) a linear regressor (using the `LinearRegression` class), and (iii) a Stochastic Gradient Descent regressor.
- Train the base estimators and the Voting regressor on the training set and compare their predicted *28-day Compressive Strength* values for the test set. Report their RMSEs and illustrate the predicted values versus the ground truth values of all the test instances.
- Individually tune a few hyperparameters for each of these base estimators. You can use 3-fold cross-validation with Grid-Search to select the best hyperparameters. Then, repeat the previous task with the selected set of hyperparameters for each base estimator. Compare and comment on the results.

4 Abalone dataset (30 marks)

An *Abalone* dataset is available in the UCI Machine Learning Repository website below:

https://archive.ics.uci.edu/ml/datasets/Abalone

It has 4177 instances with 8 attributes and a column that describes the *age*, represented in terms of the *number of rings*, of the abalones. For any new test instance, we want our regressor to be able to predict this value. The dataset in csv format is in the *abalone.data* file which can be downloaded from

https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/

You should save the file *abalone.data* to the same directory with your Jupyter Notebook file. Do not rename or modify the file in any way.

This part of the assignment includes the following tasks:

- Read in the contents of the file and perform the usual data inspection and cleaning if needed. Perform appropriate data preprocessing to the data, e.g., you can either drop the text column or convert it into numerical values. Do an 85/15 random split to form a training set and a test set.
- Train a Decision Tree Regression using a 3-fold cross-validation to tune `max_depth`, letting all other hyperparameters in their default values. Make a plot of the training errors, cross-validation errors and test errors as a function `max_depth`. Comment on the results. What value of `max_depth` would you choose?
- Train a Decision Tree Regression using a 3-fold cross-validation to tune `min_samples_leaf`, using `max_depth` as selected in the previous task and letting all other hyperparameters in their default values. Make a plot of the training errors, cross-validation errors and test errors as a function `min_samples_leaf`. Comment on the results.
- Implement a Random Forest regressor with 500 estimators. You can use `max_depth`, and `min_samples_leaf` as selected before and manually experiment with other hyperparameters, such as `max_samples`, `max_features`, `bootstrap`, etc. Train your Random Forest regressor on the training set and test it on the test set. You can Report its RMSE for the predictions on the test set. Note: as the ring

values must be integers, the predicted results from your Random Forest regressor must be firstly rounded to the nearest integer before the RMSE computation.

- Use the *feature importances* obtained from the training process to trim the feature dimension of the data. In your Python code, you should retain only those features whose *importance* values are above 5% (i.e., 0.05)[1]. You can either write your own Python code or use the function `SelectFromModel` from the `sklearn.feature_selection` package to work out which feature(s) can be removed.

  Report what features were retained and what features were removed in the above process. What is the total feature importance value that is retained after your dimension reduction step?

- Repeat the training and prediction processes above on the reduced-dimensional data. We expect to see a slight increase of RMSE for the reduced-dimensional data. In many real applications, the feature dimension of the data may be reduced drastically with only a slight increase in the prediction error. We won't see this in this small dataset as the feature dimension is already quite small.

- Finally, compare the performance of the two Random Forest regressors. Illustrate in a diagram or two the prediction errors of all the test instances from one of the above models, e.g., you can try computing the average prediction error for each ring value.

  Do large or small (or both) ring values tend to have large average errors? Is a large prediction error for a ring value related to insufficient training instances for that ring value?

- Implement a Bagging regressor with 500 SVM regressors as the base estimators. Again, choose some reasonable hyperparameter values manually for the SVM regressors. For the Bagging regressor, you should use the same common hyperparameter values (e.g., *max_features*, *max_samples*, *bootstrap*, etc) as your Random Forest regressor. Train your Bagging regressor using the full-dimensional training set and test it using the full-dimensional test set. Report the RMSE of the predictions for the test set. Illustrate in a diagram the predicted ring values versus the ground truth ring values of all the test instances.

- Compare the performance of your first Random Forest regressor with the Bagging regressor.

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.

[1]If no features are removed with the 0.05 threshold value, then increase the threshold slightly. For the exercise in this project, we want to experiment with the effect of reducing feature dimension.

- 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.
- All your diagrams/plots should have proper axis labels and titles help the reader understand what you are plotting. For the confusion matrix, not showing the class names makes the confusion matrix completely useless.
- Before submitting your assignment, restart the kernel and rerun your code from beginning to end to ensure that your code will run when we mark it.

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

7 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