**Module: CMP-6002A Machine Learning**

**Assignment: Classification with Decision Trees**

**Set by:**  Daniel Paredes-Soto (d.paredes-soto@uea.ac.uk)

**Date set:** Thursday 3 November 2022

**Value:** 50%

**Date due:** 15:00 Wednesday 14 December 2022 (Week 12)

**Returned by:** 7 February 2023

**Submission:** Blackboard

**Checked by:** Taoyang Wu (taoyang.wu@uea.ac.uk)

**Learning outcomes**

The students will learn how to implement Decision Trees and Ensemble Classifiers and the general issues related to the implementation and performance analysis of machine learning algorithms/classifiers. The coursework will improve transferable skills such as: programming in Python with conda environments; implementation of data science libraries for data manipulation; and analysis of data and results.

**Specification**

**Overview**

Implementing solutions for prediction of data using machine learning algorithms with modern tools in Python. Algorithms will be tested on public and well-established datasets.

**Description**

See attached as Part 1, 2 and 3.

**Relationship to formative assessment**

Lecture slides, Lab exercises and links to resources in slides provide the baseline to design experiments with machine learning algorithms applied to real data.

**Deliverables**

Create a folder named as your STUDENT\_ID with subfolders to store all files used, produced and Python script files containing your solutions for part 1, 2 and 3 of this coursework.

* data: datasets used. UCI Breast Cancer dataset with *categorial* features, UCI Breast Cancer dataset with *real value* features and the description files. Will be available on Blackboard.
* output: all output files (png, txt) generated from your code
* scripts: all your Python script files.
* documentation: the decision tree done by hand and the writing up pdf files. You can produce the decision tree workout on paper and scan or use digital tools to demonstrate how your decision tree was built.

Graphical user interface, application

Description automatically generated

Compress your STUDENT\_ID folder in a zip format and upload it to Blackboard.

|  |
| --- |
| **All your Python script files must include comments on top with the following information. You can include additional descriptive information such as dependencies, input and output.**  **# STUDENT\_ID: [your student id]**  **# Created on: [date created .py file]**  **# Last update: [date], [last modification to your code]**  **# Description: [The purpose of the script file]** |

|  |
| --- |
| **All plots must be formatted according to the data, title, axes ticks and labels.** |

|  |
| --- |
| **Note that some machine learning algorithms in sklearn (e.g., DecisionTreeClassifier) use some randomness. Hence, make sure you set random\_state to a fixed seed. Read the sklearn class documentation to know if the algorithm implements this.** |

**Resources**

The essential material to complete all tasks are in the lecture notes, lab sheets. Also, links to relevant materials (e.g., how to implement commonly used sklearn Classes) are included in the lecture/lab materials.

**Marking scheme**

Part 1. Decision trees by hand (10 marks)

Part 2. Implementation of Decision Trees Classifiers and Ensembles (50 marks)

Part 3. Optimisation and Evaluation of Classifiers (40 marks)

Total 100 marks.

**Plagiarism, collusion, and contract cheating**

The University takes academic integrity very seriously. You must not commit plagiarism, collusion, or contract cheating in your submitted work. Our Policy on Plagiarism, Collusion, and Contract Cheating explains:

* what is meant by the terms ‘plagiarism’, ‘collusion’, and ‘contract cheating’
* how to avoid plagiarism, collusion, and contract cheating
* using a proof reader
* what will happen if we suspect that you have breached the policy.

It is essential that you read this policy and you undertake (or refresh your memory of) our school’s training on this. You can find the policy and related guidance here: <https://my.uea.ac.uk/departments/learning-and-teaching/students/academic-cycle/regulations-and-discipline/plagiarism-awareness>

The policy allows us to make some rules specific to this assessment. Note that:

<In this assessment, working with others is *not* permitted. All aspects of your submission, including but not limited to: research, design, development and writing, must be your own work according to your own understanding of topics. Please pay careful attention to the definitions of contract cheating, plagiarism and collusion in the policy and ask your module organiser if you are unsure about anything.>\*

**Part 1. Building a Decision Trees by hand (10%)**

The aim of this exercise is to make sure you understand how basic classifiers work, and this will provide you skills to implement algorithms from the sklearn machine learning library.

Table 1 provides a dataset describing patients admitted to hospital presenting symptoms of meningitis, and their subsequent diagnosis.

Construct two Decision Trees using this data with the following splitting criteria:

1. Gini index
2. Chi-squared statistic

Note that the first Decision Tree will use the splitting criterion ‘Gini index’ only while the second one will use the ‘Chi-squared statistic’ only. Furthermore, the stopping criteria, for both DTs, should be to stop at a pure node (all instances in the same class) or if there are no more attributes to use for branching.



Table 1. Dataset of patients presenting symptoms of meningitis and their diagnosis.

Produce a pdf file (decision\_tree.pdf) and store it in the documentation folder.

**Part 2. Implementation of Decision Trees Classifiers and Ensembles (50%)**

This exercise involves implementations of Decision Trees classifiers, and coding functions to compute quality measures of attributes for splitting data. Please follow the python file and method naming specified in the instructions. Failure to follow this, may result in marks penalty.

**Functions to assess quality of attributes (10%)**

Create a Python script file and name it decision\_trees.py, where you will implement a range of functions to assess the quality of an attribute to split a node. Assume 2x2 contingency table when branching, where rows are the different attribute values and columns are the class counts. The attribute measure functions will receive a contingency table as an argument (in the format of your choice) and return the measure.

1. get\_information\_gain returns the information gain for the contingency table.
2. get\_gini return the Gini impurity measure for the contingency table.
3. get\_chi\_squared return the Chi-squared statistic for the contingency table.

The definition for information gain (IG), Gini (I) and Chi-squared can be found in the lecture slides on Decision Trees (week 2). Your methods should validate your input and execution of the mathematical formulas without crashing. Add comments to your methods to include a description and format of your input data and how did you handle undefined values. Call all the methods to find each measure for the attribute **Headache** in terms of **Diagnosis** on Table 1. Print each measure to the console in the form “measure\_{your-measure} = {value}” and save them in the headache\_splitting\_diagnosis.txt file in the output folder.

**Implementation of Decision Trees (20%)**

In this exercise you will implement a Decision Trees Classifier from the sklearn library using the entropy index measure as mechanism for attribute selection on real data. You will use the UCI Breast Cancer dataset with *categorical* features to model a binary class problem. Read the dataset description carefully to perform some transformations before implementing the classifier:

1. Remove all null, empty, “?”, unknown, and undefined values.
2. Slice the data and create the X, y data structures to follow the sklearn conventions for naming the Attributes and Class data.
3. You will research how to transform categorical values into a numerical representation. You can create your own dummy variables or use external tools such as pandas. This will allow the sklearn classifier to handle string values.
4. Apply the same principle on point 3 to the class column (y) to have a class column with values as 0 and 1s.

The decision tree will be created using 70% of your data and you will assess the performance of the classifier on the 30% remaining. Perform a random split using the random generator seed = 1. This will produce the same split always and will allow verification and validation of your code and results. Failure to set random\_state = 1 in all your sklearn algorithms relying on a random function may result in mark penalty.

The DecisionTreeClassifier in sklearn is implemented as a Class. You can modify the configuration to set, for instance, the quality measure to split data and the stop criterion. In this exercise you will use *entropy* as the quality measure and the branching stop criterion will be when reaching a *max\_depth* 3. The resulting tree will have a maximum depth previously defined. However, you will keep in mind that your tree also stops when a pure node is found or when there are no more attributes to use. This may result in some leaf nodes having a smaller depth level.

Create a decision tree classifier with the train set, then compare the balanced accuracy scores when predicting on the same train and the test set (unseen data). Print the balanced accuracy scores to the console in the form “dt\_balanced\_acc\_{train/test} = {value}” and save them in the dt\_balanced\_acc.txt file in the output folder.

Modify your program to test a range of max\_depth values to find the highest balanced accuracy.

1. Use entropy as the attribute quality measure.
2. max\_depth values ranging from 1 to 10.
3. Create a tree with the train set and predict on the test set. Use the same random split 70/30 above.
4. Print balanced scores on the test set to the console in the form “dt\_entropy\_max\_depth\_{value}\_balanced\_acc = {value}” and save them in the dt\_balanced\_acc\_scores.txt file in the output folder.
5. Store the balanced accuracy scores and plot them to visualise the scores. Save the plot as an image dt\_balanced\_acc\_scores.png in the output folder.
6. Present a graphical representation of the tree with the highest balanced accuracy. Save it as an image “dt\_entropy\_max\_depth\_{value}.png” in the output folder.

Store your Python code in the “decision\_trees\_implementation.py” file in your scripts folder.

**Implement an Ensemble Classifier (20%)**

The task is to handcraft an ensemble with a range of Decision Trees classifiers. You will diverse the classifiers by modifying two parameters: attribute selection and max depth. Predictions from the ensemble should return the majority vote of the ensemble. For example, classify a new test instance with each of the decision trees, count how many classifiers predict each class value, then return the class that receives the largest number of votes.

1. Use the UCI Breast Cancer dataset with *categorial* features from the previous exercise. Note that you have to pre-process your data before continue to point 2.
2. Generate a random split 75/25 with the random generator seed = 1.
3. Create five DecisionTreesClassifiers (sklearn) with the following parameters
   1. Alternate “gini” and “entropy” across the classifiers.
   2. Set the max\_depth to 5, 15, 20, 25, 30.

|  |  |  |
| --- | --- | --- |
| Classifier | Attribute selection | max\_depth |
| DT 1 | gini | 5 |
| DT 2 | entropy | 15 |
| DT 3 | gini | 20 |
| DT 4 | entropy | 25 |
| DT 5 | gini | 30 |

1. Train all the classifiers with the train set.
2. Predict every instance on the test set with the majority vote from your ensemble.
3. Print the balanced accuracy to the console in the form “ensemble\_balanced\_acc = {value}” and store it in the ensemble\_balanced\_accuracy.txt file in the output folder.

You can harness your ensemble using a Python list of classifiers, or the sklearn VotingClassifier Class.

Store your Python code in the “ensemble\_implementation.py” file in your scripts folder.

**Part 3. Optimisation and Evaluation of Classifiers (40%)**

In this exercise you will implement and evaluate a range of classifiers and select the best model for a given problem. You will optimise parameters of the algorithms with cross validation and report the median of the accuracy scores across the k-folds for every machine learning algorithm.

1. Use the UCI Breast Cancer dataset with *real value* features from the sklearn load\_breast\_cancer class.
2. Split data as 70/30 with the random generator seed = 1.
3. Use a 5-fold cross validation, ***without*** shuffling.

The parameter optimisation with cross validation must be done on the train set (train\_validation), and the accuracy will be estimated from predictions on the test set. Keep in mind that after finding the best parameters for the highest median accuracy, you must build the model on the train\_validation set before testing on the unseen data.

The following table shows the parameters and values to optimise the classifiers.

|  |  |
| --- | --- |
| Classifier | Parameters: Values |
| DecisionTreeClassifier | criterion: entropy  max\_depth: 2, 3, 5, 7, 10 |
| RandomForest | n\_estimators: 100, 200, 500  random\_state: 1 |
| KNN | k: 1, 11, 21, 31, 51  metric: euclidean, Manhattan |

In this exercise, the use of GridSearchCV is **not** permitted to optimise the classifiers. You will have to create and test *n* classifier models as the number of parameter value combinations. You can use GridSearchCV to verify your outputs only.

Summarise your exercise in a writing form. Describe the sequence of steps you followed to optimise the classifiers, and how you estimated the performance accuracy of the classifiers (test set). Include a table with the cv-median accuracy score for each model created. For instance, you will report five median accuracy scores for the optimisation of the DecisionTreeClassifier. Include a plot to visualise the distribution of median accuracy scores for the optimisation of each classifier. Compare the performance of the classifiers on unseen data (generalisation) with the accuracy achieved with the optimisation (train\_validation set).

The writing must not exceed two pages, you can include flow diagrams. Save your summary in a pdf file (classifiers\_evaluation.pdf) in the documentation folder.

Store your Python code in the “optimisation\_evaluation.py” file in your scripts folder.

**Check that your python scripts produce the same results always before uploading to Blackboard.**

See the Deliverables section to check specifications of script files, how to store and upload all files used and created in your solutions to Blackboard.