COMP5850/8260 – AI Systems Class 2

We are exploring classification on the adult dataset, which is designed to predict high income earners (>50k$) from US census information such as education, hours of work per week, etc. While approaching the exercises below it could be useful to keep the following pages open in your browser:
sklearn API https://scikit-learn.org/stable/modules/classes.html
pandas getting started tutorial
https://pandas.pydata.org/docs/getting_started/intro_tutorials/index.html
pyplot tutorial https://matplotlib.org/stable/tutorials/introductory/pyplot.html

Initial exploration

1. Load the **adult** dataset **(version 2)** from openml.org using the `sklearn.datasets.fetch_openml` function. Have a look at the documentation for `fetch_openml` if you are not familiar with how to use it, check what is the returned type and make sure you understand how to access the input and target feature of the dataset. Print the type of each feature to see if they are categorical or numeric (hint you can use `pandas.Dataframe.info` to get information about columns in the `DataFrame`). Do you notice any feature with missing values? Print out the size of the dataset (number of instances) and the distribution of the target across the two classes using `pandas.Series.value_counts`.

2. Split the data into a train datasets (`X_train`, `y_train`) and test dataset (`X_test`, `y_test`), using the `sklearn.model_selection.train_test_split` function. Verify the size of each dataset looking at the `shape` attribute.

3. Create an `X_train_num` training dataset by dropping the non-numerical features from the input data, (Hint you can select the relevant columns using [] and a list of column names, or use `pandas.Dataframe.drop` to drop the categorical columns or use `pandas.DataFrame.select_dtypes`).

Decision Trees

4. Train a `DecisionTreeClassifier` with default parameters to predict the target class from the numerical attributes of input using its fit method. Compute the accuracy of the classifier over the training data and the test data. Hint: you can use the `predict` method of the classifier to obtain the predicted labels from the train and test inputs and use `sklearn.metrics.accuracy_score` to compute the accuracy. Does the decision tree seem to overfit? Why? Print out the depth and the number of leaves for the tree.
5. You can limit the complexity of the tree in different ways; one way is to limit the size of the tree by specifying the maximum depth of the tree (`max_depth`) as a

parameter to `DecisionTreeClassifier`. Find from the sklearn documentation which other parameters you can use to limit the complexity of the tree.

6. Plot the training and testing accuracy vs the maximum tree depth. Have a look at the material from Programming for AI to refresh how to use matplotlib or you can check https://matplotlib.org/stable/tutorials/introductory/pyplot.html What do you notice about the two trends? What would be the best depth to use ? If you select the ideal tree depth identified from the previous point to train your model, you are committing one of the capital sins of ML, i.e. you are using information from the testing dataset to optimise your model (called data snooping or data leakage). The test set should be used just to compute the final performance of your model, ideally should be used as little as possible, and certainly not to tune any of the model (hyper)parameters. A possible correct way to tune hyper-parameters is to repeat the above process splitting the training dataset into a (smaller) training set and a validation set, however this would waste some training data and would make the model performance score (and thus the parameter selection) dependent on the specific train-validation split. A better solution is to use k-fold cross-validation by splitting the training set in k partitions and validating a model on each, trained on the rest.
7. Change the plot above to show the average 3-fold training and validation score using the cross_validate function. Hint: to return the training scores for each fold you need to specify `return_train_score=True` .
8. Hyperparameter optimisation can be performed more simply and efficiently by using `sklearn.model_selection.GridSearchCV`. Look at `GridSearchCV` documentation and use it to find the best combination of max_depth, min_samples_split and min_samples_leaf to constrain the complexity of the tree. Keep each parameter to max 3-4 choices or the computation would take a long amount of time. Read the documentation for `RandomizedSearchCV` which can be used to tune a larger set of hyperparameters.

Encoding and Pipelines

9. Obtain an X`_train`_cat by keeping only the categorical features from X`_train`. Sklearn's DecisionTreeClassifier implementation does not natively support learning from categorical features. Use sklearn.preprocessing.OneHotEncoder to transform X`_train`_cat into X_train_enc (using the fit and the transform method) and train a DecisionTreeClassifier using it. Compare its test and training performance with the decision tree computed using the numerical features.
10. Did X`_train`_cat contain any missing values? Can you figure out what happened to the missing values during the encoding by looking at the output of the get_feature_names_out() method of the encoder?
11. An alternative way to deal with missing values is to drop the instances (or features) with missing values or to fill the missing values using an Imputer. Use SimpleImputer to fill in the missing values in X`_train`_cat using the most frequent value of the missing attributes before encoding. How do you expect the set of encoded features and the classification performance at the previous points would change? Verify if that is the case.

12. Automate the use of SimpleImputer and OneHotEncoder by creating a Pipeline. Create a Pipeline with a SimpleImputer, OneHotEncoder and a DecisionTreeClassifier. Train and verify the accuracy of the pipeline.
13. It would be nice see if a Decision Tree classifier would be able to reach a higher accuracy by considering both categorical and numerical features. To do so we could start from the full dataset X_train and apply a transformer to only a subset of the attributes (DataFrame columns). Use a ColumnTransformer to apply the categorical pre-processing pipeline (composed by an Imputer and one OneHotEncoder) to the categorical attributes of the data, leaving the numerical attribute unaltered (hint: have a look at the remainder parameter of ColumnTransformer).
14. Define a Pipeline composed of the ColumnTransformer above and a DecisionTreeClassifier and train and verify the accuracy of the Pipeline. Does the use of both types of features improve the accuracy?

Ensembles

15. Train a RandomForestClassifier on the full dataset, modifying the pipeline of the previous point. Compare the training and testing accuracy with the previous cases: does the classifier overfit the data? How does the accuracy compare with the previous DecisionTrees? Print out the size of each tree in the ensemble.
16. Train an AdaBoostClassifier on the dataset. Compare train and test accuracy with random forest print out the size of each tree in the ensemble. Compare the performance of the classifier with a RandomForestClassifier that uses the same number of trees and max_depth the AdaBoostClassifier.
17. Train a GradientBoostingClassifier on the dataset and compare the performance with the above.