SEMTM0016: Week14 Lab WorkSheet: Decision tree and KNN classifiers

[Weiru Liu - based on materials originally created by Mohammad Golbabaee with thanks]

This lab: the aim is to build and evaluate classification models for the titanic dataset based on decision tree and KNN classifiers, and to fine tune their hyperparameters using cross-validation. During this lab it is highly recommended to check the sklearn documentation on class tree.DecisionTreeClassifier and neighbors.KNeighborsClassifier to fit decision trees and KNN in Python respectively, by simply typing in the name of function via scikit-learn website.

Load titanic dataset

1. Use Pandas's read_csv command to load the titanic.csv file in a table. From this table extract a binary (0-1) vector y of labels/outcomes indicating a passenger had survived or not. Create a feature matrix X whose columns are Pclass, Sex, Age, Siblings_SpousesAboard, Parents_ChildrenAboard, and Fare features. You need to convert the Sex varialble (currently is string) into a 0-1 numerical variable to include it in the feature matrix.
2. Split this data randomly into Train and Test sets with proportions 80%-20%.

Fit and Visualise decision trees for classification

1. Use sklearn commend tree.DecisionTreeClassifier to fit decision trees (or neighbors.KNeighborsClassifier.fit to fit KNN) to classify the train data with various max_leaf_nodes options (e.g. 4, 6, 8,10 ,50) (or with various number of neighbors option K=1, 2, 3, 4, ….). The number of leaves of a tree is equal to the number of splits plus one. `Visualise each tree model (e.g., named clf) using command tree.plot_tree`.
2. Fit a decision tree (or KNN) to the train data with a max number of splits e.g set split to 6 (=7 leaves) (or a chosen number of neighbors). Use this model to classify your train and test data. This can be done using the .predict attribute of the learned model (similar convention usually hold for many other sklearn's ML tools such as a classifier or regression model). For example, if your fitted tree model (or KNN model) was named clf then clf.predict(X) should give predictions (i.e. predicted class labels) for a feature matrix X.
3. Use sklearn's sklearn.metrics to compute the confusion matrix of your classifier evaluated on the train and test data.
4. Use information from confusion matrix to compute and print the classification accuracy, precision and recall, evaluated on the train and test data. You can also import other functions from sklearn.metrics (e.g., accuracy_score).

Cross validation

5. Split the train data further randomly into train & validation proportions 80%-20%, to perform a cross validation step.

6. Compute and plot the Validation Curves displaying the classification accuracies on the train & validation data against the model complexity, here, the tree's hyperparameter max_leaf_nodes ranging from 2 to 200, or KNN neighbors range from 1 to 50.
7. Can you see the over/underfitting cases in your validation curves? What's the best max_leaf_nodes number for the decision tree (or the best number of neighbors for the KNN)? Implement a grid search to find this optimal hyperparameter so that it maximises the validation accuracy.

Repeat Cross validation 50 or 100 times to get smoother results

You might also want to make better use of all of the data, perhaps the 20% we picked here for validation just happen to be particularly difficult examples? We can get a better estimate of how the entire dataset will perform if we perform multiple rounds of cross validation.

1. Therefore, in this part, repeat the random train-validation split and the associated cross validation procedure for 50 or 100 times. Average the Validation Curves over these 50 or 100 runs. Do you see smoother results?
2. Run a hyperparameter grid search on these averaged validation accuracies to determine the best number for the tree's max_leaf_nodes (or the best number of neighbors).
3. Use this optimal parameter to train a decision tree on the whole training data, the classification accuracies for the training and testing datasets.