Workshop 4: Classification

The aim of this week's workshop is to get to know the task of classification and to apply and compare two different classification methods.

You will use **Wisconsin Breast Cancer data set (https://scikit-learn.org/stable/datasets/toy_dataset.html#breast-cancer-dataset)** , which is included in scikit learn (follow the link and read about what the dataset is). To get you started we have provided a **skeleton notebook (https://myuni.adelaide.edu.au/courses/71744/files/10619134?wrap=1)** (https://myuni.adelaide.edu.au/courses/71744/files/10619134/download?download_frd=1) .

1. Familiarise yourself with the dataset by loading it using the code in the notebook and then use several different methods to display its properties, just like in Workshop 2. Pay particular attention to the class variable - this is often called the label or the target. The classifier aims to predict the label based on the remaining features in the data set. That makes it a supervised learning task.
   - Split the data into train and test set using an 80:20 ratio.
   - Then split the training part into a reduced training set and a validation set. We will use a fixed hold-out validation set for this workshop, but we could have done K-fold cross-validation in the same way as we did for regression.
   - **Question**: What elements should you put in a pre-processing pipeline and why?
   - Build an appropriate pre-processing pipeline, based on what you've seen in your investigations of the data above.
2. Stochastic Gradient Descent Classifier:
   - Build a new pipeline that includes your preprocessing and the Stochastic Gradient Descent classifier (which is a binary classifier that uses a linear model). You can find this in *sklearn.linear_model* called *SGDClassifier.* Use the parameter setting *loss='log'* (as this allows us to get probability outputs, whilst other loss functions, including the default, do not have anything except binary/integer outputs).
   - Apply the classifier to the validation set to get both binary class outputs and also probabilistic outputs using two separate calls, and store the outputs separately. We will use the binary class outputs for the next few steps, but will want the probabilities a little later.
   - Display the results graphically, along with the true labels, in such a way that it is easy to identify which ones are correct or incorrect.
   - Calculate the confusion matrix using *confusion_matrix* from *sklearn.metrics*
   - Display this confusion matrix using the *seaborn heatmap* function, with annotations on (i.e. *annot=True*)

- Calculate and display a normalised version of the confusion matrix, such that each row (True classes) sums to 1.0
- Calculate the accuracy of the classification using *accuracy_score* from *sklearn.metrics*
- Calculate the precision and recall using *sklearn's precision_score* and *recall_score*
- Calculate and display the Receiver-Operator-Characteristic (ROC) curve using *roc_curve* from *sklearn.metrics*
- **Questions**: How many distinct points are there in the ROC curve? Try calculating the ROC curve again using the probability outputs instead. Look at the thresholds and compare these to the predicted probability outputs from the classifier just at the points where the binary prediction is wrong.
- Calculate the Area Under the Curve (AUC) using the function *auc* from *sklearn.metrics*
- Calculate and display the Precision-Recall curve (using *precision_recall_curve*). This plots Precision vs Recall in the same way as an ROC curve, as this is an alternative way of looking at things, but where the *top right* corner represents the best result. These are particularly useful when you don't care about True Negatives, as TN doesn't feature in Precision or Recall.
- **Questions**: If this classifier would be used to make decisions in the hospital, which threshold would you choose? Is precision more important or recall? Do you think this classifier is good enough or does it need more research?

3. Decision tree classifier:

- Train a decision tree classifier on the same training data using the *DecisionTreeClassifier* from *sklearn.tree*. The default parameters are fine for this.
- Display the decision tree using the function *plot_tree* in *sklearn.tree*. Hint: To increase the resolution use *plt.rcParams['figure.dpi'] = 200* (if you did our standard *import matplotlib.pyplot as plt*)
- **Question**: What do each of the components (nodes, branches, thresholds) of the decision tree mean?
- Apply the classifier to the validation set. Note that you cannot get a probability output from a single decision tree.
- Calculate the confusion matrix, precision and recall. Display the confusion matrix.
- Calculate and plot the ROC curve.
- **Questions**: Why are there so few points in the ROC curve? Does it still show useful information?
- Calculate the AUC.
- **Questions**: How does the decision tree compare to the SGD linear model? List 2 pros and 2 cons of each approach.

4. Model selection

- **Question**: What do you think would be a good performance metric to use in this case, and why? Choose one to work with here.
- Compare the two models (pipelines) using your chosen performance metric, based on the results from the validation set.
- Take the chosen model and re-train it on the combination of training and validation datasets.
- Evaluate the chosen model on the test set. Compare the results to what you got from the validation data.
- **Question**: What would it mean if there was a big difference between the performance scores on the validation and test datasets?

5. Extension

- **Questions**: How stable do you think these results are?
- Try flipping the value in one element of a probability or class output (i.e. new_val = 1 - old_val) and see how much the results and curves change.