Week 4: Non-Linear Models

Anikó Ekárt

Module CS4730: Machine learning

Learning outcomes

In this unit we studied non-linear classification and regression models and the cross-validation method for model selection. In this practical, we shall deepen our understanding by applying the multi-layer perceptron and support vector machines with kernel functions and interpreting and comparing the results.

Instructions

Download the dataset `haberman.csv`. [1]

We shall build non-linear classification models for this dataset, then evaluate and compare them with each other and also the linear models previously built in Week 3.

You are encouraged to use Python Jupyter Notebooks for your work.

Task 1

This task focusses on applying a multi-layer perceptron using scikit-learns `MLPRegressor` using one hidden layer.

Divide the dataset into training and testing sets, by using the data for years 1958-1965 (229 instances) for training and 1966-1969 (77 instances) for testing.

Apply a multi-layer perceptron using scikit-learns `MLPRegressor` using one hidden layer, with different numbers of nodes, on the training data. You can use trial and error for the number of nodes or hill-climbing.

Record the performance on training and testing data, using your preferred measures (accuracy, f1-measure for example) for each experiment.

Discussion. *What is the best number of nodes? How did you decide? How does the best MLP solution compare to the linear solution of week 3?*

[1]Donated to UCI repository by Tjen-Sien Lim, used by Haberman, S. J. (1976). Generalized Residuals for Log-Linear Models, Proceedings of the 9th International Biometrics Conference, Boston, pp. 104-122.

Task 2

In this task, prepare the dataset for cross-validation using 5 folds.

Explore the use of `sklearn.model_selection.cross_validate` for this purpose.

Task 3

Use your Multi-Layer Perceptron method from Task 1 with one hidden layer of various numbers of nodes. Instead of basing your decision for number of nodes on the measures recorded on unseen test data, use cross-validation for **accuracy**.

Discussion. *Based on cross-validation, what is the best number of nodes? How does it compare to the solution you found using simple training-testing set separation?*

Task 4

Apply SVM with different kernel functions, using `sklearn.svm.SVC` with the `kernel` parameter, for example set to polynomial, RBF or sigmoid. When exploring one kernel function, study the parameters and what the expected influence of different values is. For example, for RBF, intuitively the `gamma` parameter defines how far the influence of a single training example reaches, with low values meaning far and high values meaning close. The C parameter (regularization parameter) trades off correct classification of training examples against maximization of the decision functions margin. A larger *C* leads to a smaller margin, if the decision function is better at classifying all training points correctly. A lower *C* will encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy.

Record the performance on training and testing data, using your preferred measures (accuracy, f1-measure for example).

Discussion. *For each solution, compare the results on training and testing data. What is the best model? How does the best model compare to the linear SVC found in week 3?*