Homework assignment 6:
Nearest Neighbour and Artificial Neural Networks

Objective: The objective of this exercise is to understand how k-nearest neighbor and neural networks can be used to solve classification problems.

Material: Pang-Ning Tan, Michael Steinbach, and Vipin Kumar, *Introduction to Data Mining*, section 5.2-5.4.

Important: When handing in your homework:

- Provide clear and complete answers to the questions below (in a separate file, not hidden somewhere in your source code), and make sure to explain your answers / motivate your choices.
- Source code, output graphs, derivations, etc., should be included, and zipped together with your answers.
- Hand-in: upload to Blackboard.
- Include name, student number, assignment (especially in filenames)
- When working in pairs both of you should upload the assignment, and report the name of your partner in the upload comments.
- For problems or questions: use the BB discussion board or email.

6.1 K-nearest neighbor classification

In this exercise we will use the k-nearest neighbors (KNN) method for classification. First, we will consider the four synthetic data sets `synth1`, `synth2`, `synth3` and `synth4` we used in earlier assignments.

6.1.1 For each of the four synthetic data sets, do the following. Load the dataset into Python and examine it by making a scatter plot. Classify the test data `X_test` using a k-nearest neighbor classifier. Choose a suitable distance measure (you should consider the distance measures `euclidean` and `cityblock`) Choose a suitable number of neighbors. Examine the accuracy and error rate.

Hints:

- The Python class `KNeighborsClassifier` from the `sklearn neighbors` module can be used to perform k-nearest neighbors classification.
- To generate a confusion matrix, you can use the function `confusion_matrix()` from the module `sklearn.metrics`. You can use the function `imshow()` to plot the confusion matrix.

Which distance measures worked best for the four problems? Can you explain why? How many neighbors were needed for the four problems? Can you give an example of when it would be good to use a large/small number of neighbors? Consider e.g. when clusters are well separated versus when they are overlapping.

6.1.2 In general we can use cross-validation to select the optimal distance metric and number of nearest neighbors k although this can be computationally expensive. We now return to the Iris data that we have considered in previous exercises, and will attempt to classify the Iris flowers using KNN. Load the Iris data into Python. Use leave-one-out cross-validation to estimate the number of neighbors, k, for the k-nearest neighbor classifier. Plot the cross-validated average classification error as a function of k for $k = 1, \ldots, 40$.

Hints:

- Use `LeaveOneOut` cross-validation from the module `sklearn.cross_validation`.
- As before, use the `KNeighborsClassifier` class for k-nearest neighbor classification.

6.1.3 KNN can also be used for regression by predicting the output of an observation as the average of the output values of its nearest neighbors. Predict the alcohol content of wine in the `Wine` data with KNN regression, using the other 10 attributes as predictors (ie. 'nearest' is based on those 10 attributes). Plot the error as a function of the number of nearest neighbors k for $k = 1, \ldots, 40$. What is the optimal value for the number of nearest neighbors?

Hints:

- Perform the regression on all observations, you don't need to do cross-validation here.
- Use the `KNeighborsClassifier` class to find the nearest neighbors.
- Predict each observation as the mean of the alcohol content of its nearest neighbors. Make sure you don't include the value for the observation itself.
- Compute the error for each value of k as the mean of the squared differences between the predicted and the observed alcohol content.

6.2 Artificial Neural Networks

In this part of the exercise we will use neural networks to classify the `xor` data. We will consider networks with an input layer, one layer of hidden units and an output layer. For this exercise we shall use a simple package offering neural networks functionality in Python, called Neurolab. You can find the source code, installation hints, documentation, and examples at `https://code.google.com/p/neurolab/`

6.2.1 Load the data into Python and make a scatter plot of the two attributes in X, coloring the points according to the class label y. How are X and y related?

6.2.2 Fit a neural network with one hidden unit using X as inputs and y as target, classifying observations as 1 if the predicted value is greater than 0.5 and 0 otherwise. Use 10-fold cross-validation to estimate the classification error. Plot the decision boundaries of one the networks fitted in the cross-validation using the `dbplot` function and explain why the network performs so poorly.

Hints:

- Use $neurolab.net.newff([[0, 1], [0, 1]], [H, 1], [nl.trans.TanSig(), nl.trans.TanSig()])$ with H the number of hidden units to create the network; see the Neurolab site on how to then fit it.
- To plot the decision boundaries of a network `nw` you can use the code provided in `decision_boundaries.py` to do so. NOTE: this does create the full decision boundary plot, only the contour lines and shading. You will need to overlay this with a color-coded scatter-plot like in 6.2.1 yourself. The plot should show the colored data points, with the colored contours separated by white lines indicating how points in different areas of the plot are classified.
- Sometimes the network fitting algorithm doesn't converge, and you will need to rerun it (or plot a network fitted on a different fold of the cross-validation).

6.2.3 Repeat 6.2.2, but use two hidden units instead of one. Does the classification performance improve? Can you explain why?

6.2.4 Repeat 6.2.2 with 10 hidden units. What happens to the decision boundaries of the learned neural networks? What are the benefits and drawbacks of including many hidden units in the network?