MLCC Laboratory 1: Local methods

This lab is about local methods for binary classification on synthetic data. The goal of the lab is to get familiar with the kNN algorithm and to get a practical grasp of what we have discussed in class. Follow the instructions below.

Think hard before you call the instructors or you look at the solution file!

1 Warm up - data generation

- **1.A** The function `MixGauss(means, sigmas, n)` generates dataset `X,Y` where the `X` is composed of mixed classes, each class being generated according to a Gaussian distribution with given mean and standard deviation. The points in the dataset `X` are enumerated from `0` to `n-1`, and `Y` represents the label of each point.
  Have a look at the code or, for a quick help, type "`help(MixGauss)`" in the python shell.
  Hint: If the command "`help(MixGauss)`" fails, this probably means that you have not set up correctly your current working directory.

- **1.B** Type on the python shell the commands:

```
X, Y = MixGauss([[0, 0], [1, 1]], [0.5, 0.25], 50)
fig, axs = plt.subplots(2, 2)
axs[0, 0].set_title("dataset 1")
axs[0, 0].scatter(X[:, 0], X[:, 1], s=70, c=Y, alpha=0.8)

plt.tight_layout()
plt.savefig('figure_1.png', dpi=100)
```

- **1.C** Now generate a more complex dataset following the instructions below. This dataset will be referred hereafter as training dataset:
  - Call `MixGauss` with appropriate parameters and produce a dataset with four classes: the classes must live in the 2D space and be centered on the corners of the unit square `(0,0), (0,1) (1,1), (1,0)`, all with standard deviation `0.3`. The number of points in the dataset is up to you. Use `Xtr, Ytr = MixGauss(....)`.
  - Use the python function `scatter` to plot the training dataset.
  - Manipulate the data so to obtain a 2-class problem where data on opposite corners share the same class. If you produced the data following the centers order provided above, you may do:

```
Ytr = 2*np.mod(Ytr, 2) - 1
```

- **1.D** Following the same procedure as above (section 1.C) generate a new set of data `Xts,Yts` with the same distribution, hereafter called test dataset.

2 Core - kNN classifier

The k-Nearest Neighbors algorithm (kNN) assigns to a test point the most frequent label among its k closest points/examples in the training set.

- **2.A** Have a look at the code of function kNNClassify (for a quick reference type "help(kNNClassify)" in the python shell).

- **2.B** Use kNNClassify on the previously generated 2-class data from section 1.D. Pick a "reasonable" k. Below we propose three ways of evaluating the quality of the prediction made by the kNN method. Try them and see the influence of the parameter k.

- **2.C1** [Evaluating the prediction] Plot the test data Xts twice. Once with its true labels Yts, and once with the predicted labels Ypred. A possible way is:

```
fig, axs = plt.subplots(2, 1)
axs[0].set_title('kNN prediction with k = 3')
axs[0].scatter(Xts[:, 0], Xts[:, 1], s=100, c=Yts, alpha=0.3, marker='o', edgecolor='black')
axs[0].scatter(Xts[:, 0], Xts[:, 1], s=30, c=Ypred, alpha=1, marker='^')
```

- **2.C2** [Evaluating the prediction] To compute the classification error percentage compare the estimated outputs with those previously generated:

```
error = np.mean(Ypred != Yts)
```

- **2.C3** [Evaluating the prediction] To visualize the separating function, use the routine separatingFkNN. You may use "help(separatingFkNN)" in the command prompt or look directly at the code.

3 Parameter selection - what is a good value for k?

So far we considered an arbitrary k. We now introduce different approaches for selecting it.

- **3.A** Perform a hold out cross validation procedure on the available training data. You may want to use the function holdoutCVkNN available (type "help(holdoutCVkNN)" in command prompt, you will find there a useful example of use). Plot the training and validation errors for the different values ok k.

- **3.B** Add noise to the data by randomly flipping the labels on the training set, and call it Ytr_noisy. You can use the function flipLabels to do that. How does the validation error behave now with respect to k?
  Note: Keep track of the best k, and the corresponding validation error for 3.D.

- **3.C** What happens with different values of perc (percentage of points held out) and rep (number of repetitions of the experiment)?

- **3.D** For now we have been using the training set to obtain a classifier. Now we want to evaluate its performance by applying it to an independent test set.
  - Consider the test dataset `Xts, Yts` generated in point 1.D. Add some noise to the dataset by randomly flipping some labels from `Yts`. You can use the function `flipLabels` to create the new `Xts, Yts_noisy`.
  - Take the best `k` you obtained by hold out cross validation in 3.B, and use it to get a prediction from `Xtr, Ytr_noisy, Xts`, as you did in part 2.
  - Evaluate the prediction with respect to `Yts_noisy` (as you did in 2.C2), and compare it to the validation error you had in 3.B.

4 If you have time - more experiments

- **4.A** Evaluate the results as the size of the training set grows: `n=10,20,50,100,300,...` (of course `k` needs to be chosen accordingly).
- **4.B** Generate more complex datasets with the `MixGauss` function, for instance by choosing larger variance on the data generation part.