Due: Monday, 17 November 2025, 11:59 PM

- Task 1: Obtain a data set
- Task 2: Build a kNN classifier
- Task 3: Test the kNN classifiers

Task 1: Obtain a data set

LARGE OPTION

Download the mnist data set. See https://yann.lecun.org/exdb/mnist (browser will complain, but it is a legitimate site).

Python solution: The machine learning library `keras`, part of `tensorflow`, contains ready-made functions. Just do the following:

```
from tensorflow.keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

The data represent 70 000 handwritten digits in 28x28 greyscale images, already split into a 60K-image training set and a 10K-image test set, and are a standard benchmark for machine learning tasks.

View random examples of the data with the following code:

```
nr = 6
nc = 6
N = X_train.shape[0]
fig1, ax = plt.subplots(nrows=nr, ncols=nc,layout="tight", figsize=(7,8))
fig1.tight_layout()
for i in range(nr):
  for j in range(nc):
    ax[i,j].set_yticks([])
    ax[i,j].set_xticks([])
    l = np.random.randint(N)
    ax[i,j].imshow(np.reshape(X_train[l,:],(28,28)),cmap='Greys')
plt.show()
```

SMALLER OPTION

Download the wine data set from the UCI machine learning repository. The data represent 3 types of wine (3 classes) using 13 chemical descriptors, all continuous. There are 178 observations and you should split the data into a training set and a test set yourself.

Python solution: Similarly to the MNIST case, you can obtain the Wine dataset from the `scikit-learn` library:

```
from sklearn.datasets import load_wine
wine = load_wine()
X, y = wine.data, wine.target
```

Remark for the smaller option (and general cases)

Observe that the MNIST data set has a special structure in that **all variables in each observation are homogeneous**. They all represent pixel values, and therefore they lie in the same range.

In more general cases, it is not necessarily so. Different variables may lie in different ranges. When using methods based on distance (like knn), the largest variables may dominate the summation used to compute distances even if not really important, and the smallest ones will have a negligible influence on the total. Think for example of length and width of a road, in meters.

It is therefore necessary to bring everything on the same scale. This is done with the following **normalisation** formula:

```
new_val = (orig_val  – min_orig_val) / (max_orig_val – min_orig_val)
```

This transforms all ranges into the interval [0, 1].

Note that, although this operation in python/numpy requires just two lines of code (or even one when using list comprehensions!), the scikit-learn library provides a **normalize** function to this purpose.

Task 2: Build a kNN classifier

Here you have to create a program that provides:

- A data structure (e.g. a python class) that stores the model parameters, including *k*
- A function *fit(X_train, y_train)* (e.g. a class member function) that trains the model
- A function *predict(X_test)* (e.g. another class member) that outputs the predictions for a new data set as a vector *y_pred*
- A function *test(y_test, y_pred)* (e.g. yet another class member) that outputs the fraction of correct classifications (*accuracy*)
- See the first assignment for suggestions (same program structure). When creating your own instance of the model, the class constructor should take *k* as an argument.

Hint: Remember that in python, to access elements of a class, you need to indicate self as the first argument of class function members, and class elements are accessed with the "dot" notation as for instance in *self.k*. When calling a member function, however, the first argument *self* should not be indicated.

Class names conventionally have capitalized initials, as in *Knn*.

Hint: *numpy* provides the function *mode()*.

Hint: Use the slides to implement the classifier. Use the previously provided codes for inspiration on how to organise your code.

Task 3: Test the kNN classifier

Here you have to create another program that:

- Imports your previous program, e.g. *import myknn*
- Runs several experiments; note that for experimenting with many values of *k* you have to create different instances of your knn model, since *k* is fixed at creation time (this is not a necessity, but complies with the convention in scikit-learn)

Hint: Remember that in python, to access functions or classes from an imported module, the "dot" notation is used: e.g. myknn.Knn(k)

Run experiments on various combinations of:

- the Wine data set
- different subsets of the MNIST data set, e.g. 6 subsets of size 10000 for training, 5 subsets of size 2000 for testing
- several values of *k*, e.g., k=1,2,3,4,5,10,15,20,30,40,50 (you can use these, or a subset of these, or add more numbers. Check what happens when *k* is divisible by the number of classes.

Provide data or graphs for any combination of these parameters.

For each *k*, estimate the most probable value of accuracy on the MNIST recognition problem based on the results of your multiple experiments. Also provide an estimate of variability (spread). Choose the type of centre and spread indicators that are appropriate to the case.

Add submission

Submission status

Attempt number	This is attempt 1.
Submission status	No submissions have been made yet
Grading status	Not graded
Time remaining	13 days 7 hours remaining

?