Training and Testing
When evaluating the performance of a classifier, we cannot test it on the same data with which we trained it. If we did, even a lookup table could yield perfect performance. Instead, we are interested in the classifier’s ability to generalize, i.e., its performance on data that it has never “seen” before.

The are three common strategies: Holdout testing, k-fold cross validation, and leave-one-out testing.

Holdout Testing
• The available set of exemplars is divided into training and test sets.
• Subsequently, the classifier is trained with the training set once and evaluated with the test set once.
• Classification performance is calculated as the proportion of correctly classified exemplars in the test set.
• It is very efficient because it only requires one round of training and testing.
• However, it “wastes” a lot of exemplars by only using them for testing.

K-Fold Cross Validation
• The set of exemplars is divided into k subsets s₁, s₂, …, sₖ of approximately equal size.
• In the first of k training/testing cycles, s₁ is chosen as the test set and the union of the remaining subsets is used for training.
• Similarly, in the second cycle, s₂ is the test set and the other subsets form the training set, and so on.
• The average proportion of correct classifications across the k cycles is taken as the classification performance.
• The advantage of this method is that we can use a large share of the available exemplars for training.
• However, we have to run k training/testing cycles.

Leave-One-Out Testing
• This is an extreme case of k-fold cross-validation.
• If we have N exemplars, we perform N training/testing cycles, in each of which we train with all exemplars except one and then test performance on only that one exemplar that was left out.
• In each cycle, we leave out a different exemplar so that after the N cycles, each exemplar was chosen exactly once.
• The average proportion of correct classifications across the N cycles determines the classification performance.
• Computationally expensive but useful if only a small set of exemplars is available.

The MNIST Handwritten Digit Dataset
• 60,000 28x28-pixel images for training
• 10,000 28x28-pixel images for testing

Types of Classifier
We will take a look at three common types of classifier:
• k-Nearest Neighbor (kNN) classifier,
• Naïve Bayes (NB) classifier, and
• Artificial Neural Network (ANN) classifier.

These are just popular examples of classifiers that greatly differ from each other.
The k-Nearest Neighbor Classifier

- The k-Nearest Neighbor (kNN) classifier is arguably the simplest classifier that can still provide good results.
- It is a “lazy” learner – it simply stores all training exemplars.
- When classifying a new feature vector, it simply measures its Euclidean (or other) distance to each of the stored vectors.
- It finds the k nearest neighbors and determines which class has the most representatives among them.
- That class is taken as the classification result.

Typical values of k range from 3 to 10.
Instead of giving one “vote” to each neighbor, the votes can be weighted by the neighbors proximity to the input vector.
This classifier typically performs well if the feature space is low-dimensional and many training exemplars are available.
Training is extremely efficient but classification is not, because often the entire set of exemplars needs to be processed for each new classification.

The Naïve Bayes (NB) Classifier

- For two events A and B, Bayes’ theorem tells us the following:
  \[ P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \]
- In other words, if an event B occurs, and we know how likely that event is to occur and how likely it occurs if event A happens at the same time, then we can update our knowledge on the probability of A occurring.
- We can update the prior probability \( P(A) \) to the posterior probability \( P(A|B) \) after event B took place.

During training, we determine the probability distributions of all features for each of the classes.
When classifying an input, we keep track of the probabilities for this input to belong to each of the classes.
Typically, before analyzing the input vector, it is equally likely to belong to each class.
Then we look at the value of the first feature and update the probability for each class according to Bayes’ rule and the probability distributions obtained during training.

The Artificial Neural Network (ANN) Classifier

- NNs are able to learn by adapting their connectivity patterns so that the organism improves its behavior in terms of reaching certain (evolutionary) goals.
- The strength of a connection, or whether it is excitatory or inhibitory, depends on the state of a receiving neuron’s synapses.
- The NN achieves learning by appropriately adapting the states of its synapses.
Supervised Function Approximation

In supervised learning, we train an artificial NN (ANN) with a set of vector pairs, so-called **exemplars**. Each pair \((x, y)\) consists of an input vector \(x\) and a corresponding output vector \(y\).

Whenever the network receives input \(x\), we would like it to provide output \(y\).

The exemplars thus describe the function that we want to "teach" our network.

Besides **learning** the exemplars, we would like our network to **generalize**, that is, give plausible output for inputs that the network had not been trained with.