Supervised vs. Unsupervised Learning

- **Task:** Apply some machine learning method to data from a given source
- **The source has a characteristic distribution, but we don’t know what it is.**

Supervised Learning:
- Use training data to infer model
- apply model to test data
- e.g. Maximum likelihood, Perceptron, SVM

Unsupervised Learning:
- No training data
- Model inference and application both rely on test data exclusively
  - e.g. k-means

We can test our model.

Supervised Learning:

- Fish-example from lecture
- Supervised Learning:
  - One bag with salmon
  - One bag with sea bass
  => build model
  - One mixed bag
  => please classify
- Unsupervised Learning:
  - One mixed bag
  => build model
  - Another mixed bag
  => please classify

We can not test our model.
Supervised Example: classification

1. Choose a classifier model (e.g. family of functions)
2. Use training data (pre-classified reference data) to choose a specific classifier (e.g. compute parameter)
3. Using the classifier on new data (test data): Apply classifier function to data value -> Result: Class label

Perceptron

- Idea: Separate data points from two classes \( C_1, C_2 \) by a linear function (hyperplane)
- Identify hyperplane by normal vector \( a \) (in generalized coordinates):
  \[
  a^T y_i > 0 \quad \text{for} \quad i \in C_2
  
  a^T y_i < 0 \quad \text{for} \quad i \in C_2
  \]
- Solution is not unique: possible normal vectors define a conic region (= intersection of \( n \) half spaces):
- For training: "normalization" of data points by mirroring points from \( C_2 \) at the origin:
  \[ y_i \rightarrow -y_i \quad \text{for} \quad i \in C_2 \]
  • Linear separation remains!
  • New requirement:
    \[
    a^T y_i > 0 \quad \forall \quad i
    \]

Perceptron Algorithm

- Sequentially add "normalized" misclassified data points \( y_i \) to \( a \)
- Moves \( a \) in the direction of the wider opening of the solution cone
- Terminates if \( a \) lies in the solution cone

Aufgabe 3

Perceptron to SVM

- Introduce a margin \( a^T y_i \geq b \) to reduce the cone of possible solutions (and move the separating hyperplane more to the middle):
SVM

- By definition, SVM is the maximum margin classifier defined in terms of the support vector approach.
- Real-world SVM implementations usually combine three techniques:
  1. Maximum margin classifier (this is where convex optimization comes in).
  2. Soft margin technique (slack variables).

Maximum Margin Classifiers

- Background on maximum margin classifiers: Generalization error.
- V. Vapnik: Theory of “Structural risk minimization”
- Approach: Classification error consists of two parts.
  
  \[ \text{classification error} = \text{training error} + \text{generalization error} \]

- Flexible classifiers: small training error, but possibly large generalization error (overfitting).
- Badly trained does not generalize well

Maximum Margin Classifiers

To keep the generalization error low (prevent from overfitting):

1. Use linear classifier. More complex/flexible classifiers (curved surfaces etc) are more likely to overfit.
2. When placing the hyperplane, choose the position which maximizes the margin.

Note: The only conceptual difference between the basic SVM and the perceptron is the rule for positioning the hyperplane.

Support Vectors

- Support vector idea: The classifier hyperplane is determined only by the points which are closest to it (the support vectors).
- Background: Region around class boundary is critical for classification. What happens “in the back” of the classes does not matter (classification = density estimation).
- In theory, you may have millions of data points, but only three support vectors.

Aufgabe 2

- personal opinion:
  - very nice task
  - good for low level understanding
- a): simply draw a picture
- b) – d): this is not SVM, but LSQ
  - used to illustrate the difference to SVM
- e) + f): SVM

Soft Margin

- First problem of the SVM classifier: It requires data to be linearly separable.
- Solution: Soft margin approach.
  - We allow training points on the wrong side of the classifier, but such points produce extra costs.
- The margin maximization is rewritten as a minimization problem.
- The two problems are combined.
Soft Margin

\[
\begin{align*}
\text{minimize } & \quad \frac{1}{2}w^T w + C \sum_{i=1}^{n} \xi_i \\
\text{subject to } & \quad z_i(w^T x_i + b) \geq 1 - \xi_i \\
& \quad \xi_i \geq 0
\end{align*}
\]

- There are many ways to combine two cost terms into a cost function: Every expression that becomes larger as either of the two terms increases is valid.
- Note: Slack variables apply only to training data. Classification of test points depends only on which side of the hyperplane they are on.

Optimization Theory

- The theory about convex optimization (KKT conditions etc) only comes in because the maximum margin problem (with and without slack variables) can be cast in terms of a quadratic optimization problem.
- In machine learning, the convex optimization part is usually used as a black box. To program SVMs, we use libraries or matlab functions.

Kernel Trick

Observation: Imagine some high-dimensional vector space
- Projection of a linear object (say, a hyperplane) onto a linear subspace is again linear.
- Projection of the same object onto a non-linear subset (e.g. a curved surface) is non-linear.
- If we regard our data space as a non-linear subset of some high-dimensional space, we can “simulate” non-linear operations in data space by doing linear operations in the high-dimensional space.

Kernel trick: Motivation

- Problem with linear classifiers: Not very flexible.
- Two types of errors:
  1. Classes overlap.
  2. Boundary between classes is obviously non-linear.
- The use of slack variables should be reserved for case (1).
- Consequence: We would like to be able to use “curved” classifiers.
- Curved classifiers: Nonlinear objects! Much more difficult mathematics.

Classifier still linear, but space transformed
Kernel Trick

- The key operation of our linear methods is the scalar product:
  - Computes projections (needed for classification).
  - Computes orientations (determines orthogonality/direction of hyperplanes).

- Check the slides on optimization for SVMs: The only vector operation that appears is the scalar product.

- So: If we can represent the scalar product in \( \mathbb{R}^H \) as a function in \( \mathbb{R}^L \), we can simulate linear classification in \( \mathbb{R}^H \) without actually leaving \( \mathbb{R}^L \).

=> we compute the scalar product of \( \mathbb{R}^L \) vectors in \( \mathbb{R}^H \).

Kernel Trick

- Answer: Mercer’s theorem.
  - For a function \( k(x, y) \), what do we have to require such that there is some space \( \mathbb{R}^H \) in which \( k \) is a scalar product?

  Meaning: When do we have
  \[
  k(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathbb{R}^H}
  \]

  \( \phi : \mathbb{R}^L \rightarrow \mathbb{R}^H \)

  \( k(x, y) \) is a scalar product in \( \mathbb{R}^H \).

- The Mercer condition is the functional analogy of positive definiteness for matrices (prev. slide)

  We can use all these functions instead of the scalar product.

Why do SVM work so well?

- First of all: It’s an unsolved research problem.

- Some claims: Because of...
  1. the maximum margin classifier.
  2. the compression property (solution represented by small subset of points, the support vectors).
  3. certain (still largely unproved) convergence properties of the kernel operator eigenvalues.

- Three very simple reason why SVMs are so popular:
  1. Of proven merit.
  2. Lots of experience, literature etc exists.
  3. Several easy-to-use, freely accessible, well-tested implementations are available (libsvm, svmlite etc.)