SUPPORT VECTOR MACHINES

We have seen in task 1.2 how a linear model can be used to obtain a separating hyperplane for classification. A drawback of this approach is the fact that the least squares error does not capture what we expect from an optimal separation. This is illustrated in fig. 2.1. In this example the plane created by maximizing the distance to the nearest points of each class provides a better classification. One could equivalently look for the plane with the largest possible margin (in orthogonal direction) around it such that no data point is within the margin.

In this chapter we will see how this plane can be found and we will also treat the case of nonlinear separability. The resulting algorithm is known as Support Vector Machine (SVM) and it is one of the most famous algorithms in Machine Learning [4, 5].

2.1 OPTIMAL SEPARATING HYPERPLANES

Let $\Omega = \mathbb{R}^d$, $\Gamma = \{-1, 1\}$. Instead of performing a least squares fit $g$ for given data $D := \{(x_i, y_i) \in \Omega \times \Gamma \mid i = 1, \ldots, n\}$ and then using the hyperplane $g(t) = 0$ as a separator, we now determine the optimal margin hyperplane

$$f(t) = a_0 + \sum_{i=1}^{d} a_i t_i = t^\top \cdot a$$

(2.1)

![Figure 2.1: Optimal fit of a hyperplane to input data labelled with two classes. The optimization is w.r.t. least-squares (dashed) and maximum minimum distance to nearest point (solid).]
between two classes of points which can be separated linearly. To this end, we solve the constrained optimization problem

$$\max_{\alpha \in \mathbb{R}^{d+1}, \sum_{i=1}^{d} \alpha_i^2 = 1} \ M$$

subject to $y_i \left( \mathbf{x}_i^\top \cdot \alpha \right) \geq M$ for all $i = 1, \ldots, n$.

This problem can be recast into its so-called Wolfe dual form

$$\max_{\beta \in \mathbb{R}^n} \sum_{i=1}^{n} \beta_i - \frac{1}{2} \sum_{i,j=1}^{n} \beta_i \beta_j y_i y_j \langle x_i, x_j \rangle$$

subject to $0 \leq \beta_i \ \forall \ i = 1, \ldots, n$

and $\sum_{i=1}^{n} \beta_i y_i = 0$.

Now, $f$ is given by

$$f(t) = \sum_{i=1}^{n} \beta_i y_i \langle t, x_i \rangle + b.$$  \hspace{1cm} (2.2)

Note that we can easily switch between the representations (2.1) and (2.2) by setting $a_0 = b$ and

$$(a_1 \ldots a_d)^T = \sum_{i=1}^{n} \beta_i y_i x_i.$$  

Details can be found in [2, 4]. We still have to propose a suitable optimization algorithm to solve (OMH) and to determine the so-called bias $b$.

### 2.2 Support Vector Machines

By slightly altering the optimization problem above, we obtain a so-called support vector machine. To this end we add additional constraints to (OMH):

$$\max_{\beta \in \mathbb{R}^n} \sum_{i=1}^{n} \beta_i - \frac{1}{2} \sum_{i,j=1}^{n} \beta_i \beta_j y_i y_j \langle x_i, x_j \rangle$$

subject to $0 \leq \beta_i \leq C \ \forall \ i = 1, \ldots, n$

and $\sum_{i=1}^{n} \beta_i y_i = 0$.

for some constant $C > 0$. This can be interpreted as a so-called regularization. It allows us to obtain a model which possibly represents a better generalization for unseen test data than in the unregularized case $C = \infty$. More specifically, the choice of $C$ will introduce a trade-off between the minimization of the misclassification error and the maximization of the margin $M$ from above.
The so-called support vectors are the \( x_k \) for which \( \beta_k > 0, k \in \{1, \ldots, n\} \). The name hints at the fact that these are the necessary input data points, which span the vector \((a_1 \ldots a_d)^T\) that determines the hyperplane. To solve (SVM) we will use the sequential minimal optimization (SMO) algorithm. Note that for the linear SVM we considered so far, other solvers are more suitable. But the SMO algorithm can be easily adapted to the nonlinear SVM, which is introduced next.

2.2.1 Sequential minimal optimization

**Algorithm 2.1** OneStep algorithm to update the coefficients \( \beta_i, \beta_j \) and the bias \( b \) of \( f(\cdot) = \sum_{l=1}^n \beta_l y_l \langle \cdot, x_l \rangle_\Omega + b \)

**Input:** Indices \( i, j \in \{1, \ldots, n\} \).

\[
\delta \leftarrow y_i \left( (f(x_j) - y_j) - (f(x_i) - y_i) \right)
\]

\[
s \leftarrow y_i \cdot y_j
\]

\[
\chi \leftarrow \langle x_i, x_i \rangle_\Omega + \langle x_j, x_j \rangle_\Omega - 2 \cdot \langle x_i, x_j \rangle_\Omega
\]

\[
\gamma \leftarrow s\beta_i + \beta_j
\]

if \( s = 1 \)

\[
L \leftarrow \max(0, \gamma - C)
\]

\[
H \leftarrow \min(\gamma, C)
\]

else

\[
L \leftarrow \max(0, -\gamma)
\]

\[
H \leftarrow \min(C, C - \gamma)
\]

end if

if \( \chi > 0 \)

\[
\beta_i \leftarrow \min \left( \max \left( \beta_i + \frac{s}{\chi} L, H \right) \right)
\]

else if \( \delta > 0 \)

\[
\beta_i \leftarrow L
\]

else

\[
\beta_i \leftarrow H
\]

end if

\[
\beta_j \leftarrow \gamma - s\beta_i
\]

Update function evaluations \( f(x_l), l = 1, \ldots, n \)

\[
b \leftarrow b - \frac{1}{2} (f(x_i) - y_i + f(x_j) - y_j)
\]

The SMO algorithm [3] works in an iterative manner. At first the values for \( \beta \) and \( b \) are initialized (e.g. as 0). In every iteration step we select two indices \( i, j \in \{1, \ldots, n\} \) and solve the quadratic optimization problem (SVM) by fixing all \( \beta_k \) for indices \( k \in \{1, \ldots, n\} \setminus \{i, j\} \). Note, that this can be done exactly. To this end, one iterative step for the selected indices \( i, j \) can be found in algorithm 2.1.

**Task 2.1.** Implement the function OneStep from algorithm 2.1, which takes one iterative step of the SMO algorithm for two selected indices \( i \) and \( j \).
Task 2.2. To have a small data set on which we can test our algorithm, we draw 20 two-dimensional vectors according to an exponential distribution with $\lambda = 4$ in each of the coordinate directions, i.e. the $j$-th coordinate of the $i$-th vector is drawn i.i.d. according to $x_{ij} \sim \exp(4)$ for all $i = 1, \ldots, 20$ and $j = 1, 2$. We assign the label $-1$ to these $x_i$. Then, we draw 20 two-dimensional vectors according to $\exp(0.5)$ in the same way and assign the label 1 to them.

Task 2.3. Implement a function SMO which initializes $\beta = 0$ and $b = 0$ and - in each iteration step - randomly picks $i, j \in \{1, \ldots, n\}$ such that $i \neq j$ and calls OneStep with indices $i, j$ to perform an optimization.

(a) After the last iteration step, we need to compute a final estimate for $b$. To this end, calculate the median $\text{med}$ of $f(x_k) - y_k$ for all support vector indices $k$, i.e. all $k \in \{1, \ldots, n\}$ for which $\beta_k > 0$. Then, set $b \leftarrow b - \text{med}$.

(b) Run the SMO function with 10,000 iteration steps to compute a support vector classifier $f$ for the $n = 40$ data points from task 2.2. Compute the results for $C = 0.01$, $C = 1$ and $C = 100$. For each $C$, plot the scattered data and the hyperplane corresponding to $f = 0$. Compare your results to the separating hyperplane computed by a linear least squares algorithm.

(c) Count the number of support vectors. Mark the corresponding $x_k$ in your scattered data plot.

(d) Furthermore, also count the number of margin defining vectors, i.e. the number of indices $k \in \{1, \ldots, n\}$ for which $C > \beta_k > 0$ and mark the corresponding $x_k$ in the scattered data plot. An example for such a plot can be found in fig. 2.2.

What influence does the parameter $C$ have on the number of the support vectors and on the position of the separating hyperplane?

Now let us check how our classifiers perform if we evaluate them on some test data.
**Task 2.4.** Draw 2,000 test data points according to the distributions from *task 2.2* (1,000 points for class \(-1\) and 1,000 points for class 1). Evaluate the accuracy (percentage of correctly classified data points) for the LLS and SVM models calculated in *task 2.3*.

The random picks of \(i, j\) in the SMO algorithm can be very ineffective for large data sets. Therefore, we have to come up with a better heuristic to choose appropriate indices in each step of the SMO algorithm. There exist many heuristics to choose suitable indices in each step. We refer the interested reader to [3, 4]. We will employ the Karush-Kuhn-Tucker conditions of the dual minimization problem:

\[
\text{KKT}_i := (C - \beta_i) \max (0, 1 - y_i f(x_i)) + \beta_i \max (0, y_i f(x_i) - 1). \tag{2.3}
\]

**Task 2.5.** Repeat *task 2.3* and *task 2.4* but instead of drawing the indices \(i, j\) for each SMO-step randomly, write an outer loop which iterates over all \(i \in \{1, \ldots, n\}\) and check if \(\text{KKT}_i > 0\). If this is the case, randomly pick a \(j \neq i\) for which \(0 < \beta_j < C\). If no such \(j\) exists, randomly pick a \(j \in \{1, \ldots, n\}\ \backslash \{i\}\). Subsequently, run the OneStep function for the pair \((i, j)\). If \(\text{KKT}_i = 0\) for each \(i\) or if the maximum number of OneStep calls (10,000) is reached, the algorithm terminates. Compare the results achieved with this heuristic with the results achieved by randomly picking \(i\) and \(j\). How do their runtimes compare?

### 2.3 Nonlinearity – Feature Maps and Kernels

A major drawback of the linear least squares approach and the support vector machines above is the fact that the resulting functions are linear. However, in cases where the distribution of the input data is such that a linear hyperplane is not a suitable to classify the data, it is advantageous to consider nonlinear approaches. We already learned about a very simple nonlinear algorithm: \(k\)-nearest neighbors. Here, the separation is done by a nonlinear function. Next, we will learn about the nonlinear SVM.

#### 2.3.1 Nonlinear SVM

The main reason for the huge success of SVMs in machine learning is due to the fact that we can slightly alter (SVM) and (2.2) to obtain nonlinear classifiers. To this end, we consider the so-called kernel trick: We change the scalar products to an evaluation of a kernel function \(K : \Omega \times \Omega \rightarrow \mathbb{R}\):

\[
\langle t, x \rangle_\Omega \rightarrow K(t, x). \tag{Kernel trick}
\]

In machine learning this is usually done by using a nonlinear feature map \(\phi : \Omega \subset \mathbb{R}^d \rightarrow V\) into a Hilbert space \(V\) (usually with higher dimension than \(d\)) and defining

\[
K(t, x) := \langle \phi(t), \phi(x) \rangle_V.
\]
In this way, we transform our input data by the feature map and can apply our SVM algorithm on the image of $\phi$ by using the scalar product in $V$. Let us have a look at a simple example.

**Task 2.6.** Generate 50 uniformly distributed i.i.d. points which lie in $\{t \in \mathbb{R}^2 \mid \|t\|_2 < 1\}$ (e.g. by drawing uniformly distributed points in $(-1,1)^2$ until 50 of them are within the unit sphere) and label them by $-1$. Now generate 50 data points, which are uniformly distributed in $\{t \in \mathbb{R}^2 \mid 1 < \|t\|_2 < 2\}$ and label them by 1.

(a) Fit a linear SVM for $C = 10$ to the data and plot the scattered data as well as the separating hyperplane.

(b) Transform the data by the feature map $\phi : \mathbb{R}^2 \to \mathbb{R}^3$ defined by

$$
\phi(t) := (t_1, t_2, t_1^2 + t_2^2)
$$

Fit an SVM for $C = 10$ to the transformed data. Plot the scattered data and the nonlinear separation curve in a 2d plot (i.e. in the same way as in (a)). What does the feature map do and why does it work so well?

One of the most important theorems for kernel learning algorithms such as the nonlinear SVM is Mercer’s theorem: It tells us that for each continuous, symmetric and non-negative definite kernel function $K$ there exists a corresponding feature map $\phi$. However, for many famous kernels such as the Gaussian

$$
K_\sigma(t, x) := \exp \left( -\frac{\|t - x\|_{\mathbb{R}^d}^2}{2\sigma^2} \right)
$$

the corresponding vector space $V$ can be infinite-dimensional and an explicit construction of $\phi$ can be infeasible to compute. In these cases it makes much more sense to work directly with the kernel $K$.

**Task 2.7.** Change your SMO code and your function evaluation of $f$ from (2.2) such that it allows to use a kernel function instead of the scalar product of the input data, i.e. substitute all scalar products by the evaluation of the kernel function. Perform a SVM classification ($C = 10$) with Gaussian kernel ($\sigma = 1$) for the data from task 2.6. Plot the scattered data and the nonlinear separation curve in a 2d plot. The result should look similar to fig. 2.3.

2.3.2 $k$-fold crossvalidation

In practical applications the choices of the regularization parameter $C$ as well as optional kernel parameters, such as $\sigma$ for the Gaussian kernel, play an important role. The most common technique to determine these so-called hyperparameters is crossvalidation. Here, the training data set is randomly split into $k$ parts/folds of approximately equal size. One fold is taken as test (or evaluation) data while the remaining $k - 1$
Figure 2.3: Support Vector Classifier with Gaussian kernel for $C = 10, \sigma = 1$.

folds serve as input data for our algorithm. Subsequently, we take a different fold as evaluation data and the rest as input data and repeat the process $k$ times until each fold has been used as evaluation data once. The (arithmetic) average of the $k$ accuracies calculated on the evaluation data serves as our quality measure. This process is called $k$-fold crossvalidation.

Now, to determine the best choice of hyperparameters, we choose small candidate sets, e.g. $C \in \{0.01, 0.1, 1, 10, 100\}, \sigma \in \{1, 10, 100\}$ and run a $k$-fold crossvalidation for all possible combinations of parameter pairs. The pair $(C, \sigma)$ with the best average accuracies in the crossvalidation process is the winner. The corresponding pseudocode can be found in algorithm 2.2. Subsequently, the winning parameter set is usually taken to learn an SVM on the whole training data, i.e. all $k$ folds. The resulting model is then evaluated on the true test data, which has not been touched during the crossvalidation process. More details on this approach are given in [2, 4] for example.

Algorithm 2.2 Abstract $k$-fold crossvalidation scheme

**Input:** $k \in \mathbb{N}$, training data $D$, possible combinations of hyperparameters $P$.

Randomly split $D$ into $k$ parts $D_1, \ldots, D_k$ of (almost) equal size.

**for** all $p \in P$ **do**

**for** all $i = 1, \ldots, k$ **do**

Run learner with input data $\bigcup_{j \neq i} D_j$ and parameters $p$.

Evaluate resulting model on $D_i$ and store accuracy $A_i$.

**end for**

Average over the accuracies: $A^p \leftarrow \frac{1}{k} \sum_{i=1}^{k} A_i$.

**end for**

Determine $p_{best} \leftarrow \arg \max_{p \in P} A^p$.
2.4 Application to Real World Data

We will now apply a support vector machine to a real-world classification problem.

2.4.1 Multi-class Learning

Up to now, we always considered classification problems, where our label set $\Gamma$ was of size two, i.e. we just had two different classes. In real-world applications one often encounters so-called multi-class classification problems, where $|\Gamma| > 2$. In this case, a very common idea is to use $\frac{|\Gamma|(|\Gamma|-1)}{2}$ pairwise classifiers, i.e. classifiers to discern between each possible pair $\gamma_1 \neq \gamma_2$ of classes in $\Gamma$. To decide, in which class a data point $t$ lies, each pairwise classifier is evaluated and the class $\gamma \in \Gamma$ to which $t$ is assigned the most wins.

In this way, we can apply standard two-class algorithms to solve multi-class problems. We refer to [2, 4] for more details on different approaches to multi-class problems.

2.4.2 The MNIST Data Set

The MNIST data set (http://yann.lecun.com/exdb/mnist/) consists of 70,000 grey-scale images ($28 \times 28$ pixels) of handwritten digits. Four exemplary images can be found in fig. 2.4. Our goal will be to construct an algorithm which is able to identify the correct digit from an image of the handwritten one.

You can either download and extract it by hand or use the following lines of code. As you see, you might need to install the urllib library. To this end, just run `pip install urllib3` in your shell.

```python
# Load MNIST Data
import os
import gzip
from urllib.request import urlretrieve

def download(filename, source='http://yann.lecun.com/exdb/mnist/ '):
    print("Downloading %s" % filename)
    urlretrieve(source + filename, filename)
```

Send anonymous feedback for this page.
def load_mnist_images(filename):
    if not os.path.exists(filename):
        download(filename)
    with gzip.open(filename, 'rb') as f:
        data = np.frombuffer(f.read(), np.uint8, offset =16)
    data = data.reshape(-1, 28, 28)
    return data / np.float32(256)

def load_mnist_labels(filename):
    if not os.path.exists(filename):
        download(filename)
    with gzip.open(filename, 'rb') as f:
        data = np.frombuffer(f.read(), np.uint8, offset =8)
    return data

X_train = load_mnist_images('train-images-idx3-ubyte.gz')
y_train = load_mnist_labels('train-labels-idx1-ubyte.gz')
X_test = load_mnist_images('t10k-images-idx3-ubyte.gz')
y_test = load_mnist_labels('t10k-labels-idx1-ubyte.gz')

2.4.3 Scikit-Learn – A neat machine learning library in python

For the sake of understanding the basic programming and machine learning paradigms, we did (and will) implement the learning algorithms on our own. However, we will also learn how to use important python machine learning libraries such as scikit-learn (http://scikit-learn.org). This is an efficient and easy-to-use library in which we can find variants of all algorithms we have learned about so far (LLS, k-NN, SVM) and many more.

Task 2.8. Make yourself familiar with the SVC function in scikit-learn, which implements a support vector classifier.

(a) Choose a random subset of size 500 from the MNIST training data and use this as your new training data set for crossvalidation. Perform a 5-fold crossvalidation SVM to determine the optimal parameters among $C \in \{1, 10, 100\}$ and $\gamma = \frac{1}{\sigma^2} \in \{0.1, 0.01, 0.001\}$. (Hint: You can use the scikit-learn function GridSearchCV.)

(b) Use the determined optimal parameters to learn a support vector classifier on a random 2,000 point subset of the MNIST training data and evaluate the confusion matrix and the accuracy on the whole MNIST test data set. (Hint: You can use the scikit-learn module metrics.)

Is our approach of picking a different training set in step (b) – and learning with the optimal parameters from (a) – valid? Are there potential pitfalls?
2.5 WHAT WE DID NOT COVER...

**NON-NUMERICAL DATA** Note that the feature map approach allows us to also classify data which does not reside in an Euclidean space by building appropriate feature maps that assign a value in $V$ to each element of the input data. This is often very useful when it comes to practical applications where data is not directly given as numerical values or vectors.

**KERNEL CHOICE** The kernel can also be chosen by crossvalidation over a finite set of fixed kernel functions for instance. However, if we have some a priori problem knowledge (such as smoothness of the “true” separation function), we can exploit this in order to choose an appropriate kernel, see also [1].

**REGRESSION** The linear least squares and the $k$-nearest neighbors algorithms also apply to the regression case, where we look for a function $f$ such that $f(x_i) \approx y_i$ and the $y_i$ can take arbitrary values in $\mathbb{R}$ – instead of only discrete ones as in classification. However, for support vector machines this is not so straightforward since our optimization problem (SVM) originated from the optimal margin hyperplane formulation. Nevertheless, there also exists a support vector machines regression algorithm based on the minimization of the so-called $\varepsilon$-insensitive loss function, see [4].

**REFERENCES**


