Prof. Dr. Jochen Garcke
Dr. Bastian Bohn

2 SUPPORT VECTOR MACHINES

Send your solutions to this chapter's tasks until May 7th.

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 [5, 6].

The assignment on GitHub classroom for this exercise is at `https://classroom.github.com/a/hEyvpX-Q`.

2.1 OPTIMAL SEPARATING HYPERPLANES

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

$$f(\mathbf{t}) = \alpha_0 + \sum_{i=1}^{d} \alpha_i \mathbf{t}_i = \hat{\mathbf{t}}^\top \cdot \boldsymbol{\alpha} \tag{2.1}$$

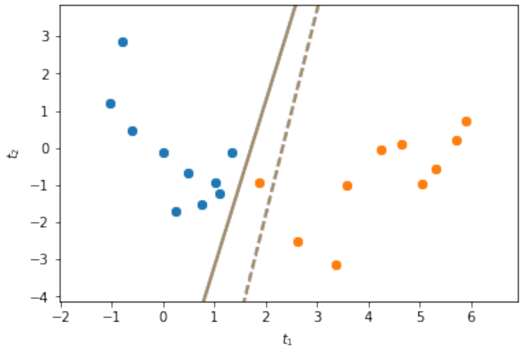

Figure 2.1: Optimal fit of a hyperplane to input data labelled with two classes. The optimization is w.r.t. least-squares (dashed) and maximum minimum distance to nearest point (solid).

between two classes of points which can be separated linearly. To this end, we solve the constrained optimization problem

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^{d+1},\sum_{i=1}^{d}\alpha_i^2=1} M$$
$$\text{subject to } \mathbf{y}_i\left(\hat{\mathbf{x}}_i{}^{\top}\cdot\boldsymbol{\alpha}\right)\geq M \text{ for all } i=1,\ldots,n.$$

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

$$\begin{aligned}\max_{\beta\in\mathbb{R}^n}\ &\sum_{i=1}^{n}\beta_i-\frac{1}{2}\sum_{i,j=1}^{n}\beta_i\beta_j\mathbf{y}_i\mathbf{y}_j\langle\mathbf{x}_i,\mathbf{x}_j\rangle_\Omega\\ \text{subject to } &0\leq\beta_i\quad\forall\, i=1,\ldots,n\\ \text{and } &\sum_{i=1}^{n}\beta_i\mathbf{y}_i=0.\end{aligned}\tag{OMH}$$

Now, f is given by

$$f(\mathbf{t})=\sum_{i=1}^{n}\beta_i\mathbf{y}_i\langle\mathbf{t},\mathbf{x}_i\rangle_\Omega+b.\tag{2.2}$$

Note that we can easily switch between the representations (2.1) and (2.2) by setting $\alpha_0=b$ and

$$(\alpha_1\ldots\alpha_d)^T=\sum_{i=1}^{n}\beta_i\mathbf{y}_i\mathbf{x}_i.$$

Details can be found in [2, 3, 5]. 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):

$$\begin{aligned}\max_{\beta\in\mathbb{R}^n}\ &\sum_{i=1}^{n}\beta_i-\frac{1}{2}\sum_{i,j=1}^{n}\beta_i\beta_j\mathbf{y}_i\mathbf{y}_j\langle\mathbf{x}_i,\mathbf{x}_j\rangle_\Omega\\ \text{subject to } &0\leq\beta_i\leq C\quad\forall\, i=1,\ldots,n\\ \text{and } &\sum_{i=1}^{n}\beta_i\mathbf{y}_i=0.\end{aligned}\tag{SVM}$$

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 $\mathbf{x}_k$ for which $\beta_k > 0, k \in \{1,\dots,n\}$. The name hints at the fact that these are the necessary input data points, which span the vector $(\alpha_1 \dots \alpha_d)^T$ 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 β_i, β_j and the bias b of $f(\cdot) = \sum_{l=1}^{n} \beta_l \mathbf{y}_l \langle \cdot, \mathbf{x}_l \rangle_\Omega + b$

```
Input: Indices $i, j \in \{1,\dots,n\}$.

$\delta \leftarrow \mathbf{y}_i \left( (f(\mathbf{x}_j) - \mathbf{y}_j) - (f(\mathbf{x}_i) - \mathbf{y}_i) \right)$
$s \leftarrow \mathbf{y}_i \cdot \mathbf{y}_j$
$\chi \leftarrow \langle \mathbf{x}_i, \mathbf{x}_i \rangle_\Omega + \langle \mathbf{x}_j, \mathbf{x}_j \rangle_\Omega - 2 \cdot \langle \mathbf{x}_i, \mathbf{x}_j \rangle_\Omega$
$\gamma \leftarrow s\beta_i + \beta_j$
if $s = 1$ then
    $L \leftarrow \max(0, \gamma - C)$
    $H \leftarrow \min(\gamma, C)$
else
    $L \leftarrow \max(0, -\gamma)$
    $H \leftarrow \min(C, C - \gamma)$
end if
if $\chi > 0$ then
    $\beta_i \leftarrow \min\left(\max\left(\beta_i + \frac{\delta}{\chi}, L\right), H\right)$
else if $\delta > 0$ then
    $\beta_i \leftarrow L$
else
    $\beta_i \leftarrow H$
end if
$\beta_j \leftarrow \gamma - s\beta_i$
Update function evaluations $f(\mathbf{x}_l), l = 1,\dots,n$
$b \leftarrow b - \frac{1}{2}(f(\mathbf{x}_i) - \mathbf{y}_i + f(\mathbf{x}_j) - \mathbf{y}_j)$
```

The SMO algorithm [4] works in an iterative manner. At first the values for $\boldsymbol{\beta}$ and b are initialized (e.g. as 0). In every iteration step we select two indices $i, j \in \{1,\dots,n\}$ and solve the quadratic optimization problem (SVM) by fixing all β_k for indices $k \in \{1,\dots,n\} \setminus \{i,j\}$. 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.*

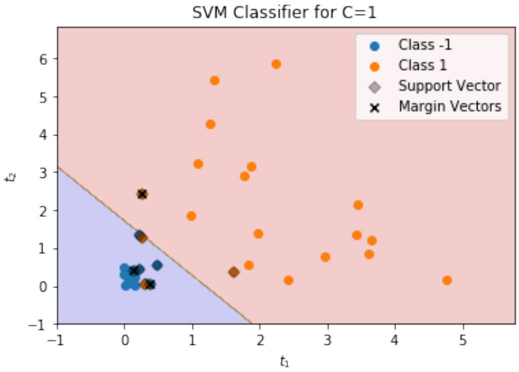

Figure 2.2: Support Vector Classifier for $C = 1$.

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* $[\mathbf{x}_i]_j \sim \exp(4)$ *for all* $i = 1, \dots, 20$ *and* $j = 1, 2$. *We assign the label* -1 *to these* $\mathbf{x}_i$. *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* $\boldsymbol{\beta} = 0$ *and* $b = 0$ *and - in each iteration step - randomly picks* $i, j \in \{1, \dots, n\}$ *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 mean* me *of* $f(\mathbf{x}_k) - \mathbf{y}_k$ *for all indices* $k \in \{1, \dots, n\}$ *for which* $0 < \beta_k < C$. *Then, set* $b \leftarrow b -$ me.

(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 $\mathbf{x}_k$ *in your scattered data plot.*

(d) Furthermore, also count the number of margin *defining vectors, i.e. the number of indices* $k \in \{1, \dots, n\}$ *for which* $C > \beta_k > 0$ *and mark the corresponding* $\mathbf{x}_k$ *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 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 [4, 5]. We will employ the *Karush-Kuhn-Tucker* conditions of the dual minimization problem:

$$\mathrm{KKT}_i := (C - \beta_i) \max(0, 1 - y_i f(\mathbf{x}_i)) + \beta_i \max(0, y_i f(\mathbf{x}_i) - 1)\,. \tag{2.3}$$

Task 2.5. (***Master students only***) *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* $i \in \{1, \dots, n\}$ *and check if* $\mathrm{KKT}_i > 0$*. 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* $j \in \{1, \dots, n\} \setminus \{i\}$*. Subsequently, run the* `OneStep` *function for the pair* (i, j)*. If* $\mathrm{KKT}_i = 0$ *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 $K : \Omega \times \Omega \to \mathbb{R}$

$$\langle \mathbf{t}, \mathbf{x} \rangle_\Omega \longrightarrow K(\mathbf{t}, \mathbf{x}). \tag{Kernel trick}$$

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

$$K(\mathbf{t}, \mathbf{x}) := \langle \phi(\mathbf{t}), \phi(\mathbf{x}) \rangle_V.$$

In this way, we transform our input data by the feature map and can apply our SVM algorithm on the image of ϕ 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* $\{\mathbf{t} \in \mathbb{R}^2 \mid \|\mathbf{t}\|_2 < 1\}$ *(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* $\{\mathbf{t} \in \mathbb{R}^2 \mid 1 < \|\mathbf{t}\|_2 < 2\}$ *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*

$$\phi(\mathbf{t}) := \left(t_1, t_2, t_1^2 + t_2^2\right).$$

Fit an SVM for $C = 10$ *to the transformed data. Plot the scattered data and the nonlinear separation curve in a* 2*d 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 ϕ. However, for many famous kernels such as the Gaussian

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

the corresponding vector space V can be infinite-dimensional and an explicit construction of ϕ 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* 2*d 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 σ 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 $k - 1$

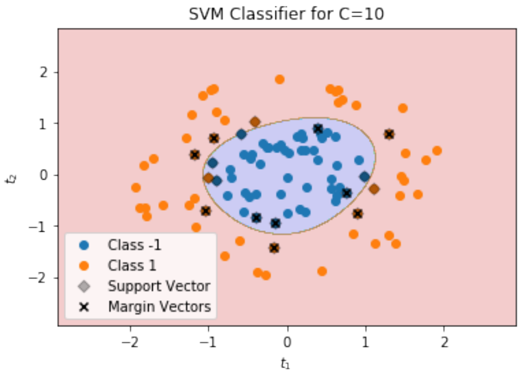

Figure 2.3: Support Vector Classifier with Gaussian kernel for $C = 10, \sigma = 1$.

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, σ) 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, 5] for example.

Algorithm 2.2 Abstract k-fold crossvalidation scheme

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

Randomly split $\mathcal{D}$ into $k$ parts $\mathcal{D}_1, \dots, \mathcal{D}_k$ of (almost) equal size.
for all $p \in P$ do
    for all $i = 1, \dots, k$ do
        Run learner with input data $\cup_{j \neq i} \mathcal{D}_j$ and parameters $p$.
        Evaluate resulting model on $\mathcal{D}_i$ 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 $p_{\text{best}} \leftarrow \arg\max_{p \in P} A^p$.
```

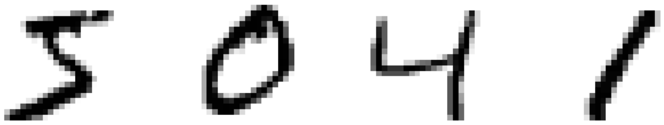

Figure 2.4: Four example images (28 × 28 pixels) from the MNIST data set (http://yann.lecun.com/exdb/mnist/).

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 Γ 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 Γ. To decide, in which class a data point $\mathbf{t}$ lies, each pairwise classifier is evaluated and the class $\gamma \in \Gamma$ to which $\mathbf{t}$ is assigned the most wins.

In this way, we can apply standard two-class algorithms to solve multi-class problems. We refer to [2, 5] 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/ or https://www.kaggle.com/datasets/hojjatk/mnist-dataset/data) consists of $70,000$ grey-scale images (28×28 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.

The data is in eCampus. You can also download and extract it by hand or use the following lines of code (be aware that the download routine might not work due to temporary unavailability of the URL). As you see, you might need to install the `urllib` library. To this end, just run `pip install urllib3` in your shell.

```
 #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)

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* $\gamma = \frac{1}{2\sigma^2} \in \{0.1, 0.01, 0.001\}$*. (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 $f(\mathbf{x}_i) \approx \mathbf{y}_i$ and the $\mathbf{y}_i$ 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 ε-insensitive loss function, see [5].

REFERENCES

[1] F. Cucker and D. Zhou. *Learning theory*. Cambridge Monographs on Applied and Computational Mathematics, 2007.

[2] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2009. URL: https://web.stanford.edu/~hastie/ElemStatLearn/download.html.

[3] Trevor Hastie, Robert Tibshirani, Gareth James, and Daniela Witten. *An Introduction to Statistical Learning (2nd ed.)* Springer, 2021. URL: https://www.statlearning.com/.

[4] John Platt. *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*. Tech. rep. 1998.

[5] B. Schölkopf and A. Smola. *Learning with Kernels – Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press – Cambridge, Massachusetts, 2002.

[6] V. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.