Machine Learning

Notes Part 3, HSG-MiQEF, Spring 2022

David Preinerstorfer

Last updated on 2022-03-08 16:10:31

Contents

Goals:

- Explore more complex classification algorithms. In contrast to the methods from last week, which were obtained by putting assumptions on the distribution of $(Y, \boldsymbol{X})$, the methods we take a look at today are guided by "visual principles".
- The focus in this notes is on **nearest neighbor methods**, **separating hyperplanes** and **support vector machines**.

This lecture draws on material from Chapter 4 of Richter [2019] and Chapters 4, 12 and 13 of Hastie et al. [2009].

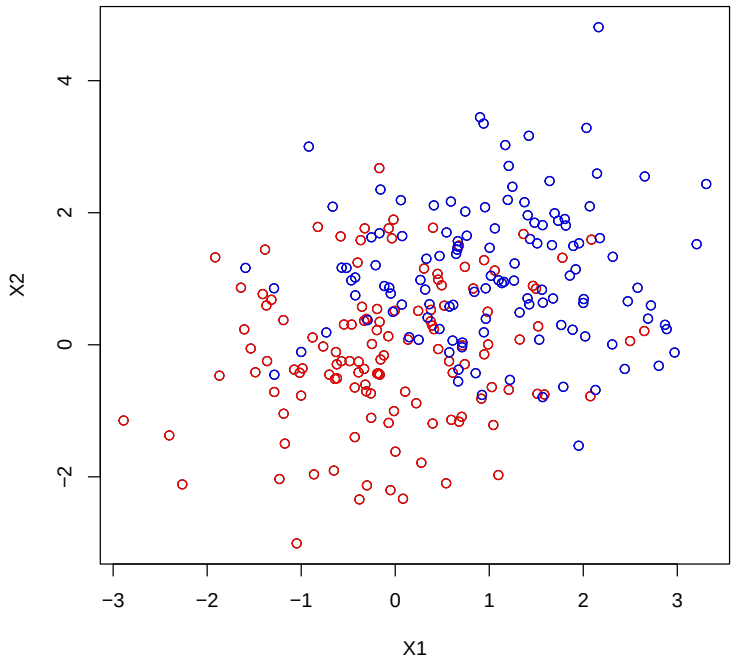

Figure 1: Data from last week, generated in the context of a classification problem satisfying the assumptions imposed for discriminant analysis.

1 Nearest neighbor classification

A simple, intuitive and assumption-free heuristic to obtain a classifier is the following (which is not derived from optimality considerations and modeling assumptions, but essentially from visual considerations): Fix a natural number k. For the input vector $\boldsymbol{x}$ we then obtain a classification rule as follows:

1. Find those k observations in the training sample, the feature vectors of which are closest to $\boldsymbol{x}$.
2. Set $\hat{f}_n(\boldsymbol{x})$ equal to that class, which appears most frequently among the k data points obtained from the first step; i.e., classify $\boldsymbol{x}$ according to "majority vote".

This method is called k-**nearest neighbor classifier** (kNN classifier). Here the term "closest" needs to be defined formally, before one can go on to implement this kind of algorithm on a computer. Typically, "closest" is defined via a measure of distance, such as the Euclidean distance. But one can also use other distance measures, based on different norms, for example. Popular norms are so-called p-norms:

$$\|\boldsymbol{x}\|_p = (\sum_{i=1}^{K} |x_i|^p)^{1/p},$$

for $p \geq 1$, and where we *define*

$$\|\boldsymbol{x}\|_\infty := \max_{i=1,\ldots,K} |x_i|.$$

In Figure 2 it is illustrated how the sets where different p-norms equal 1 compare to each other. Note the different type of geometry inherent to the different p-norms. The formula $(\sum_{i=1}^{K} |x_i|^p)^{1/p}$ can also be used for $p \in (0, 1)$, even though this does not define a norm.

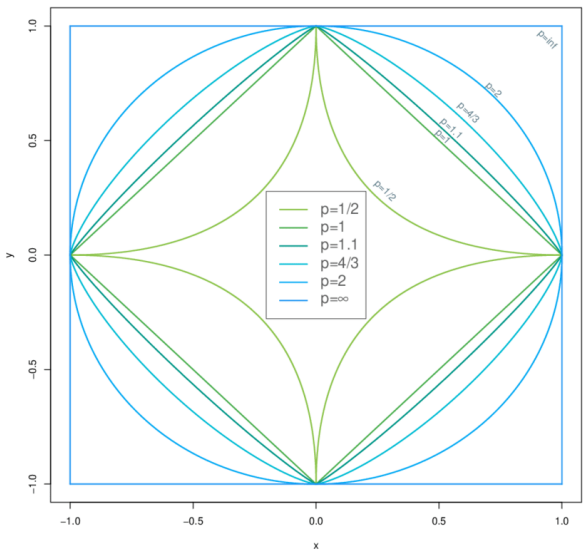

Figure 2: P-norm balls for different values of p.

The value of k is a tuning parameter in the kNN classifier which can be fixed in advance (e.g., $k = \lfloor \sqrt{n} \rfloor$, where n is the size of the training sample); it can also be determined via validation (e.g., by cross-validation).

In any case, the above algorithm is implemented in the `R` package **class** through the function `knn`, which can be used via the command `knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)`.

Note that while the algorithm is easy to formulate, it is not so easy to compute if there are many features, i.e., in the high-dimensional case (you will have the opportunity to think about that in an exercise)! To see what the algorithm does on our datasets, you can find in Figure 3 the result of running it on our (simple) data example from last week (that satisfies the assumptions underlying LDA and QDA).

```
library(class)
class_knn <- knn(X, G_lda, k = floor(sqrt(length(y))), y)
plot(X, xlab = "X1", ylab = "X2")
points(G_lda[class_knn == 1,], pch = 46, col = "lightblue", lwd = 1)
points(X[y == 0,], col = "red")
points(X[y == 1,], col = "blue")
```

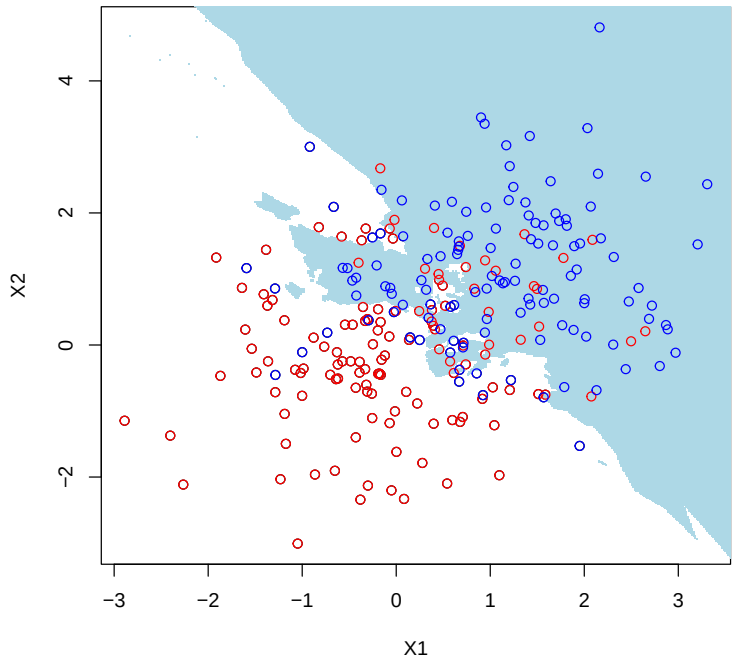

Figure 3: kNN classifier for DA data.

The confusion matrix can be obtained via

```
#confusion matrix
pred_knn <- (knn(X, X, k = floor(sqrt(length(y))), y)==1)*1
conf_knn <- table(list("predicted" = pred_knn, "observed" = y))
conf_knn
```

```
##          observed
## predicted  0  1
##         0 95 25
##         1 31 99
```

```
sum(diag(conf_knn))/sum(conf_knn)
```

```
## [1] 0.776
```

We can also apply this to the bitcoin example to obtain the result shown in Figure 4:

```
library(class)
class_knn <- knn(X_btc, G_btc, k = floor(sqrt(length(y_btc))), y_btc)
plot(X_btc, xlab = "X1", ylab = "X2")
points(G_btc[class_knn == 1,], pch = 46, col = "lightblue", lwd = 1)
points(X_btc[y_btc == 0,], col = "red")
points(X_btc[y_btc == 1,], col = "blue")
```

Similarly as above, the confusion matrix can be obtained via

```
#confusion matrix
pred_knn_btc <- (knn(X_btc, X_btc, k = floor(sqrt(length(y_btc))), y_btc)==1)*1
conf_knn_btc <- table(list("predicted" = pred_knn_btc, "observed" = y_btc))
conf_knn_btc
```

```
##          observed
## predicted   0   1
##         0 113  82
##         1  71  97
```

```
sum(diag(conf_knn_btc))/sum(conf_knn_btc)
```

```
## [1] 0.5785124
```

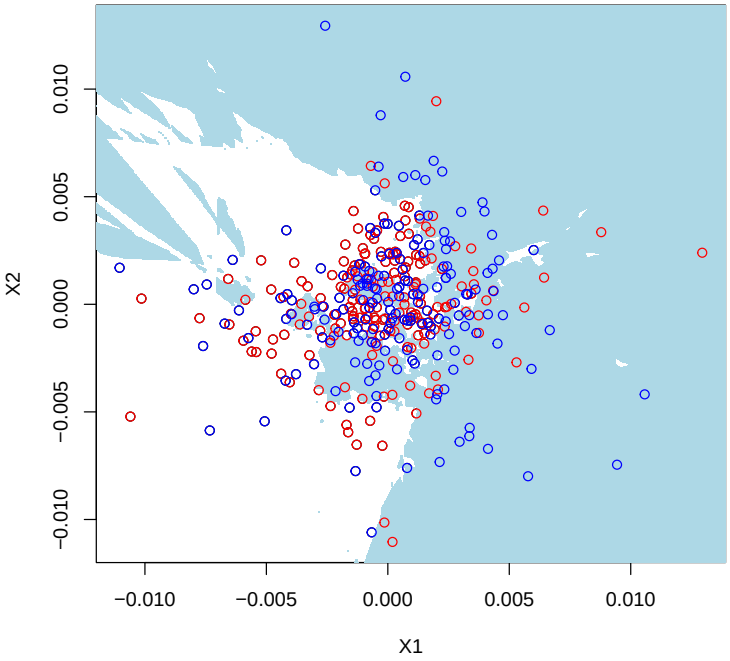

Figure 4: kNN classifier for bitcoin data.

The classification regions obtained in both cases are certainly "erratic". This can be traded off with choosing the number of neighbors larger. The numbers k used is a tuning parameter of the algorithm, which one may determine through validation procedures, such as cross-validation (as we discussed last week).

2 Support vector machines

Throughout this section, we assume that there are only two classes, i.e., $K = 2$. Furthermore, instead of calling the first class 1 and the second 2, we will code the first class as -1 and the second one as 1, because this is more convenient for formulating the theory. That is, Y can take on either of the two values -1 or 1.

2.1 Separating hyperplanes

To understand the intuition behind support vector machines, the name of which certainly is a bit intimidating, we first take a look at a simpler problem: Suppose that the training data $(Y_i, \boldsymbol{X}_i)$ is such that they can be **linearly separated**. That means we assume that there exists a *nonzero* (which we implicitly impose

throughout) vector $(\beta_0, \boldsymbol{\beta}')' \in \mathbb{R}^{p+1}$, such that the following holds: for every i we have

$$\text{sign}(\beta_0 + \boldsymbol{X}_i'\boldsymbol{\beta}) = Y_i,$$

where sign denotes the sign function (which we set 0 at the origin). That is, $Y_i = 1$ if and only if $\beta_0 + \boldsymbol{X}_i'\boldsymbol{\beta} > 0$ and $Y_i = -1$ if and only if $\beta_0 + \boldsymbol{X}_i'\boldsymbol{\beta} < 0$. If this is the case, the **hyperplane**

$$\{\boldsymbol{x} \in \mathbb{R}^p : \beta_0 + \boldsymbol{x}'\boldsymbol{\beta} = 0\}$$

is called *a* **separating hyperplane** of the training sample, because this hyperplane separates the points correctly into the group of those observations with $Y_i = 1$ and the group of those observations with $Y_i = -1$, respectively. A picture helps to clarify this; cf. Figure 5.

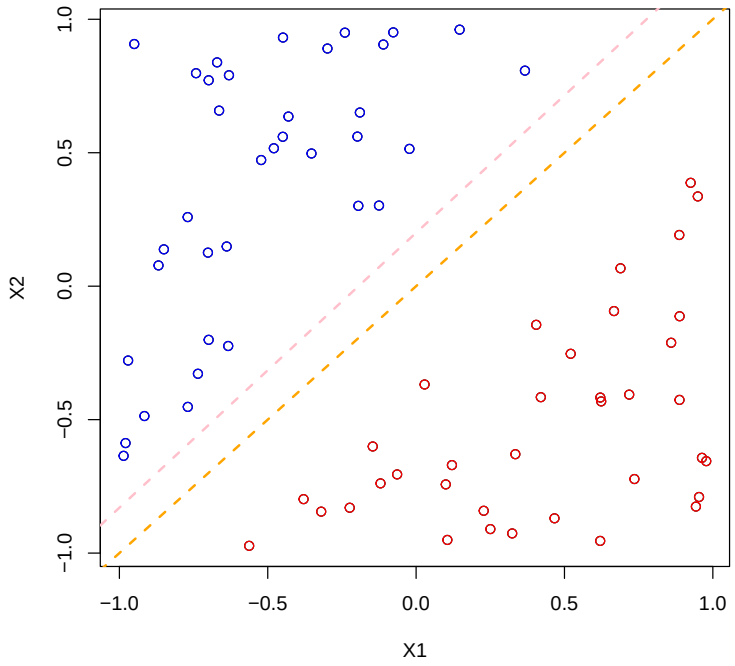

Figure 5: Linearly separated data and 2 separating hyperplanes.

Here, the training sample is linearly separable, and there are of course infinitely many separating hyperplanes. Two are shown in the figure. One possible separating hyperplane is obtained with $\boldsymbol{\beta} = (0, 1, -1)'$ (which is precisely the orange dashed line).

If we view the above as the data in a classification problem, most reasonable classifiers would find a separation as the one given in the figure. The question now is *which* linear separation one would consider "most

appropriate".

We shall consider a hyperplane as the **optimal separating hyperplane** (sometimes called the **maximal margin classifier**), if it

(i) separates the points (an obvious requirement indeed), and

(ii) maximizes (among all separating hyperplanes) the minimal distance of the $\boldsymbol{X}_i$ to the hyperplane.

Visually, for $p = 2$, we are determining the "broadest, straight road" which we can squeeze in between the two groups of points. The median of this "road" would then correspond to the hyperplane we seek! To formalize this geometric idea, we need to obtain an expression for the vector that describes the difference between a given point $\boldsymbol{z}$ and the closest point to it (i.e., its **projection onto**) in a given hyperplane (i.e., a candidate for a separating hyperplane)

$$\{\boldsymbol{x} \in \mathbb{R}^p : \beta_0 + \boldsymbol{x}'\boldsymbol{\beta} = 0\}.$$

You probably did such computations in high-school!

First, note that the point $\boldsymbol{x}_0 := -\beta_0\boldsymbol{\beta}/\|\boldsymbol{\beta}\|^2$ is an element of the hyperplane (it satisfies its defining equation). Next, to find the vector we are looking for, we determine the projection of $\boldsymbol{z} - \boldsymbol{x}_0$ onto the space spanned by $\boldsymbol{\beta}$. But (by the general OLS theory) this is $\boldsymbol{\beta}\boldsymbol{\beta}'(\boldsymbol{z} - \boldsymbol{x}_0)/\|\boldsymbol{\beta}\|^2$. The distance of $\boldsymbol{z}$ to the hyperplane thus equals the norm of $\boldsymbol{\beta}\boldsymbol{\beta}'(\boldsymbol{z} - \boldsymbol{x}_0)/\|\boldsymbol{\beta}\|^2$, i.e., equals

$$\frac{|\boldsymbol{\beta}'(\boldsymbol{z} - \boldsymbol{x}_0)|}{\|\boldsymbol{\beta}\|} = \frac{|\beta_0 + \boldsymbol{\beta}'\boldsymbol{z}|}{\|\boldsymbol{\beta}\|}.$$

We can now write down formally what we mean by "optimal separating hyperplane":

1. The hyperplane determined by $(\beta_0, \boldsymbol{\beta})'$ separates the points if $Y_i(\beta_0 + \boldsymbol{\beta}'\boldsymbol{X}_i) > 0$ holds for every $i = 1, \ldots, n$. Hence, this describes a constraint on the possible hyperplanes we are looking for.
2. Among those, we want to maximize the distance to the closest data point. Recall from above that for every feature vector $\boldsymbol{X}_i$ the distance to the hyperplane determined by $(\beta_0, \boldsymbol{\beta}')'$ equals $\frac{|\boldsymbol{\beta}'\boldsymbol{X}_i + \beta_0|}{\|\boldsymbol{\beta}\|}$. We can write the absolute value in the numerator (for a hyperplane satisfying 1.) also as $\frac{Y_i(\beta_0 + \boldsymbol{\beta}'\boldsymbol{X}_i)}{\|\boldsymbol{\beta}\|}$.

Summarizing, and defining the function $M(\beta_0, \boldsymbol{\beta}) = \min_{i=1,\ldots,n} \frac{Y_i(\beta_0 + \boldsymbol{\beta}'\boldsymbol{X}_i)}{\|\boldsymbol{\beta}\|}$ we arrive at the optimization problem (the equality in the constraint can be added without any loss, as those do not contribute to the maximum)

$$\max_{\beta_0, \boldsymbol{\beta}} M(\beta_0, \boldsymbol{\beta}) \quad \text{w.r.t. to the constraint} \quad M(\beta_0, \boldsymbol{\beta}) \geq 0.$$

While, essentially, this would be good enough, note that the maximizer $(\beta_0^*, \boldsymbol{\beta}^*)$, say, is not unique (we can multiply such a vector by $c > 0$ and get the same hyperplane, as you can easily verify). To solve this issue,

we impose an additional constraint. It is convenient to impose the following constraint: $M(\beta_0, \boldsymbol{\beta}) \times \|\boldsymbol{\beta}\|_2 = 1$. The resulting problem can be rewritten as

$$\max_{\beta_0, \boldsymbol{\beta}} M(\beta_0, \boldsymbol{\beta}) \quad \text{w.r.t.} \quad M(\beta_0, \boldsymbol{\beta}) \times \|\boldsymbol{\beta}\|_2 = 1 \text{ and } M(\beta_0, \boldsymbol{\beta}) \geq 0.$$

Two observations:

1. We can drop the condition $M(\beta_0, \boldsymbol{\beta}) \geq 0$ as it is implied by $M(\beta_0, \boldsymbol{\beta}) \times \|\boldsymbol{\beta}\|_2 = 1$.
2. We can replace $M(\beta_0, \boldsymbol{\beta})$ in the objective by $1/\|\boldsymbol{\beta}\|_2$. We can further turn the minimization problem into a maximization problem by replacing the objective $1/\|\boldsymbol{\beta}\|_2$ by $\|\boldsymbol{\beta}\|_2^2/2$. This does not change the optimizer.

We arrive at

$$\min_{\beta_0, \boldsymbol{\beta}} \|\boldsymbol{\beta}\|_2^2/2 \quad \text{w.r.t.} \quad M(\beta_0, \boldsymbol{\beta}) \times \|\boldsymbol{\beta}\|_2 = 1.$$

Note that $M(\beta_0, \boldsymbol{\beta}) \times \|\boldsymbol{\beta}\|_2 = 1$ implies $Y_i(\beta_0 + \boldsymbol{\beta}' \boldsymbol{X}_i) \geq 1$ for every $i = 1, \ldots, n$, and the existence of a single coordinate for which this inequality is realized. We thus see that the constraints are **relaxed**, if we replace $M(\beta_0, \boldsymbol{\beta}) \times \|\boldsymbol{\beta}\|_2 = 1$ by

$$Y_i(\beta_0 + \boldsymbol{\beta}' \boldsymbol{X}_i) \geq 1 \text{ for every } i = 1, \ldots, n.$$

But if the minimizer of this relaxed problem would not have a single index for which the inequality constraint would hold, it would not be a minimizer (we could rescale), so that this relaxation actually coincides with the initial problem. We have now arrived at the optimization formulation of the problem under consideration:

$$\min_{\beta_0, \boldsymbol{\beta}} \|\boldsymbol{\beta}\|_2^2/2 \quad \text{w.r.t.} \quad Y_i(\beta_0 + \boldsymbol{\beta}' \boldsymbol{X}_i) \geq 1 \text{ for every } i = 1, \ldots, n. \tag{1}$$

If the data is not separable, then there is no solution to this optimization problem. We assumed throughout that the data are separable, hence a solution exists.

Separating hyperplane classifier

If the training data are linearly separable, denote the optimizer of (1) by $(\beta_0^*, \boldsymbol{\beta}^*)$. The linear hyperplane classifier is then defined as

$$f_{n,LHC}^*(\boldsymbol{x}) = \text{sign}(\beta_0^* + \boldsymbol{x}' \boldsymbol{\beta}^*).$$

Those indices i for which $Y_i(\beta_0^* + \boldsymbol{\beta}^* \boldsymbol{X}_i) = 1$ are called **support vectors** (note that such vectors must always exist, as we have argued above). These are the indices, which determine the maximally possible distance

between the training data of one group and the other one! That means that if we through away all data points, and only keep the i for which $Y_i(\beta_0^* + \boldsymbol{\beta}^* \boldsymbol{X}_i) = 1$, we would obtain exactly the same separating hyperplane (this is important for algorithmic considerations).

The optimization problem in (1) is a **quadratic optimization problem with inequality constraints**. The particular instance in (1) can be solved with, e.g., the `R` package **e1071** through its function `svm` (which we shall re-encounter further below, as the name already suggests). The following code shows how this can be achieved for the example in Figure 5 and also plots the resulting optimal separating hyperplane in black, which is given in Figure 6.

```
library(e1071)
fit.svm <- svm(factor(yS)~XS, kernel = "linear", scale = FALSE, cost = 10000)
beta <- drop(t(fit.svm$coefs)%*%XS[fit.svm$index,])
beta0 <- fit.svm$rho


plot(XS, xlab ="X1", ylab = "X2")
points(XS[yS == -1,], col = "red")
points(XS[yS == 1,], col = "blue")
abline(a = beta0/beta[2], b = -beta[1]/beta[2], lty = 2, lwd = 2)
abline((beta0 - 1) / beta[2], -beta[1] / beta[2], lty = 2)
abline((beta0 + 1) / beta[2], -beta[1] / beta[2], lty = 2)
```

We need to adjust the cost parameter in the optimization problem, because the problem solved by `svm` is actually more complex, and one needs to force the function to look for separating planes by choosing a penalty parameter particularly strongly, that every solution that would depart from separating the points is severely penalized (this will become clear later, when we discover the full scope of the function `svm`).

2.2 Support vector machines

In the previous section we have seen how linearly separable features can be separated by an "optimal" separating hyperplane through solving a quadratic program with linearity constraints. The method is purely visually motivated. And the motivation is reasonable for classification if the data is separable. The optimization problem 1 does not permit a solution in case the training sample is not linearly separable.

The basic idea behind support vector machines is to "relax" the condition in the optimization problem imposing that the points *must* be linearly separated. One just gives some room so that this assumption can

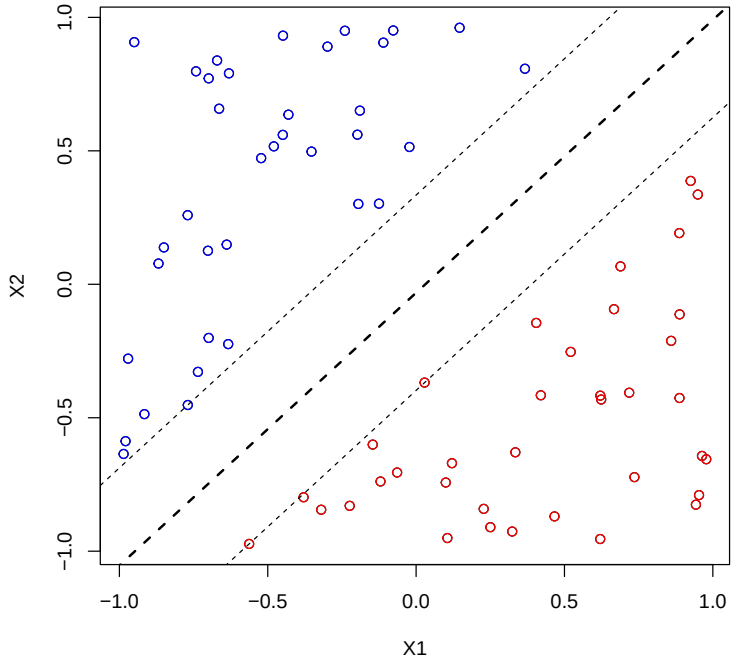

Figure 6: Optimal separating hyperplane.

be slightly violated, and determines a hyperplane that separates the points "as good as possible" within the required constraint. To do so, one simply relaxes the condition in the optimization problem 1, and instead considers for a constant $D > 0$ the optimization problem:

$$\min_{\beta_0,\boldsymbol{\beta}} \|\boldsymbol{\beta}\|_2^2/2 \quad \text{w.r.t.} \quad Y_i(\beta_0 + \boldsymbol{\beta}'\boldsymbol{X}_i) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \text{ for every } i = 1, \ldots, n \text{ and } \sum_{i=1}^{n} \xi_i \leq D. \tag{2}$$

This problem is slightly annoying, because one has to determine D in advance, and it could simply be that for the chosen constant, there is no solution to the optimization problem. An alternative (strongly related) optimization problem is given by the regularized problem

$$\min_{\beta_0,\boldsymbol{\beta},\boldsymbol{\xi}\in\mathbb{R}^n} \|\boldsymbol{\beta}\|_2^2/2 + C\sum_{i=1}^{n} \xi_i \quad \text{w.r.t.} \quad Y_i(\beta_0 + \boldsymbol{\beta}'\boldsymbol{X}_i) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \text{ for every } i = 1, \ldots, n, \tag{3}$$

which gives rise to the support vector machine classifier:

Support vector machine classifier

Fix a number $C \geq 0$. If $(\beta_0^*, \boldsymbol{\beta}^*)$ is a solution to the problem (3), the corresponding support vector machine classifier is

$$\hat{f}_{n,SV,C}(\boldsymbol{x}) = \text{sign}(\beta_0^* + \boldsymbol{x}'\boldsymbol{\beta}^*).$$

Let $\hat{\xi}$ be the vector of "slack variables" that corresponds to the solution. We can then make the following observations:

1. If $\hat{\xi}_i > 1$, then $\boldsymbol{X}_i$ is wrongly classified by $\hat{f}_{n,SV,C}$.
2. If $\hat{\xi}_i \in (0, 1]$, $\boldsymbol{X}_i$ is correctly classified by $\hat{f}_{n,SV,C}$, but it lies in the "interior" of the "road" corresponding to the hyperplane fit.
3. If $\hat{\xi}_i = 0$, $\boldsymbol{X}_i$ is correctly classified by $\hat{f}_{n,SV,C}$, and it does not lie on the "separation road".

Note that the optimization problem (3) is again a quadratic program. The constraint set is convex. Fast algorithms exist for this class of optimization problems.

To see what the optimization problem delivers, let's take a look at the data example we also used for LDA. The classifier obtained is shown in Figure 7. Again, we use the same function `svm` from above, but now we do work with a `cost` parameter that corresponds to our C (the penalty parameter). The following code can be applied:

```
library(e1071)
fit.svm2 <- svm(factor(y)~X, kernel = "linear", scale = FALSE, cost = 2)
beta2 <- drop(t(fit.svm2$coefs)%*%X[fit.svm2$index,])
beta02 <- fit.svm2$rho
plot(X, xlab ="X1", ylab = "X2")
points(X[y == 0,], col = "red")
points(X[y == 1,], col = "blue")
abline(a = beta02/beta2[2], b = -beta2[1]/beta2[2], lty = 2, lwd = 2)
abline((beta02 - 1) / beta2[2], -beta2[1] / beta2[2], lty = 2)
abline((beta02 + 1) / beta2[2], -beta2[1] / beta2[2], lty = 2)
```

From the figure we see that the support vector machine does of course not separate the points (because there does not exist a separating hyperplane). Nevertheless, it does a good job in classifying them, and the classifier is close to the optimal one (as was also the LDA classifier). We can also take a look at the confusion matrix:

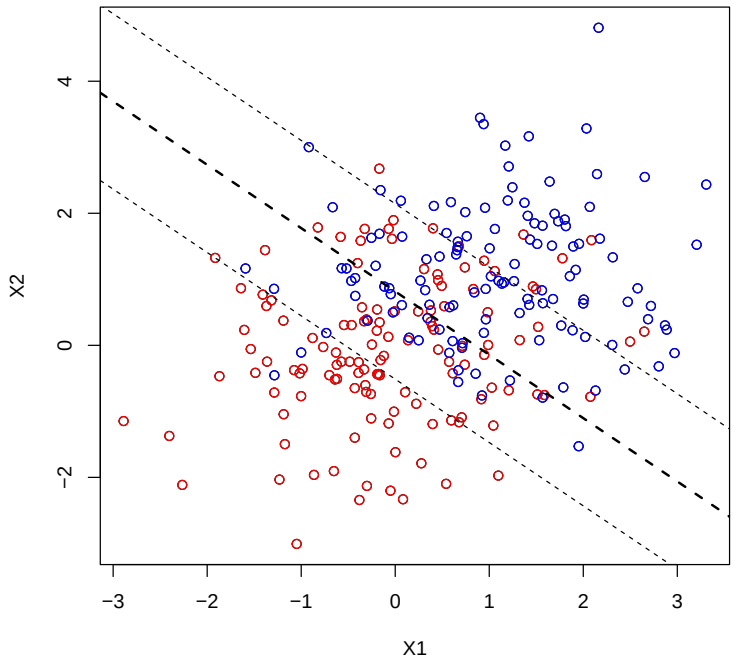

Figure 7: Support vector machine classifier.

```
conf_svm <- table(predict(fit.svm2, X), y)
conf_svm
```

```
##    y
##      0  1
##   0 93 31
##   1 33 93
```

```
sum(diag(conf_svm))/sum(conf_svm)
```

```
## [1] 0.744
```

For our next purpose (departing from linearity!), it is very beneficial to take a look at the dual problem. Let's revise the necessary terminology from mathematical optimization.

2.3 Interlude on optimization: Lagrange/Wolfe duality and the KKT conditions

Consider a minimization problem

$$\min_{\boldsymbol{a}\in\mathbb{R}^l} f(\boldsymbol{a}) \quad \text{w.r.t. the constraints} \quad h_i(\boldsymbol{a}) \leq 0 \text{ for } i = 1, \ldots, m. \tag{4}$$

The functions f and h_i are defined on $\mathbb{R}^l$ and are all assumed to be convex and everywhere (coordinate-wise) continuously differentiable with gradient ∇f and ∇h_i.

Recall that the **Lagrangian function** of (4) is defined as

$$L(\boldsymbol{a}, \boldsymbol{\lambda}) = f(\boldsymbol{a}) + \sum_{i=1}^{m} \lambda_i h_i(\boldsymbol{a}).$$

Furthermore, the **Lagrangian dual function** is defined as

$$g(\boldsymbol{\lambda}) = \min_{\boldsymbol{a}\in\mathbb{R}^l} \left(f(\boldsymbol{a}) + \sum_{i=1}^{m} \lambda_i h_i(\boldsymbol{a}) \right).$$

The **Lagrange dual** to the optimization problem (4) (which is then called the **primal problem**) is the optimization problem

$$\max_{\boldsymbol{\lambda}\in\mathbb{R}^m} g(\boldsymbol{\lambda}) \quad \text{w.r.t. the constraint } \lambda_i \geq 0 \text{ for } i = 1, \ldots, m. \tag{5}$$

If $\boldsymbol{a}^*$ denotes a solution to the primal problem (4) and $\boldsymbol{\lambda}^*$ denotes a solution to the dual problem (5) and $\boldsymbol{a}$ and $\boldsymbol{\lambda}$ denote feasible points in the primal and dual problems, respectively, we obtain that

$$g(\boldsymbol{\lambda}) \leq g(\boldsymbol{\lambda}^*) \leq L(\boldsymbol{a}^*, \boldsymbol{\lambda}^*) \leq f(\boldsymbol{a}^*) \leq f(\boldsymbol{a}).$$

In particular, **weak duality** always holds, i.e.,

$$g(\boldsymbol{\lambda}^*) \leq f(\boldsymbol{a}^*).$$

That is a solution of the dual problem always provides a lower bound on the minimum in the primal problem. If equality is achieved for $\boldsymbol{a}^*$ and $\boldsymbol{\lambda}^*$, the optimization problem is **strongly dual** and $\boldsymbol{a}^*$ is a solution to the primal program and $\boldsymbol{\lambda}^*$ is a solution to the dual problem. Strong duality follows (for example) from **Slater's condition** which assumes the existence of a $\boldsymbol{a} \in \mathbb{R}^l$ such that $h_i(\boldsymbol{a}) < 0$ for all $i = 1, \ldots, m$.

Karush-Kuhn-Tucker conditions

The **Karush-Kuhn-Tucker conditions** for the (primal) minimization problem above and vectors $\boldsymbol{a}$ and $\boldsymbol{\lambda}$ are:

- **Stationarity**: $\mathbf{0} = \nabla f(\boldsymbol{a}) + \sum_{i=1}^{m} \lambda_i \nabla h_i(\boldsymbol{a})$.
- **Complementarity**: $\lambda_i h_i(\boldsymbol{a}) = 0$ for $i = 1, \ldots, m$.
- **Primal feasibility**: $h_i(\boldsymbol{a}) \leq 0$ for $i = 1, \ldots, m$.
- **Dual feasibility**: $\lambda_i \geq 0$ for $i = 1, \ldots, m$.

We then have the following result:

Theorem 1. *If the optimization problem is strongly dual (e.g., because Slater's condition holds), then vectors $\boldsymbol{a}^*$ and $\boldsymbol{\lambda}^*$ satisfy the KKT conditions if and only if $\boldsymbol{a}^*$ and $\boldsymbol{\lambda}^*$ are primal and dual solutions, respectively.*

Proof. If $\boldsymbol{a}^*$ and $\boldsymbol{\lambda}^*$ are solutions, then the feasibility conditions are satisfied by construction, and it remains to establish stationarity and complementarity. Note that, because of strong duality, it holds that

$$f(\boldsymbol{a}^*) = g(\boldsymbol{\lambda}^*) = \min_{\boldsymbol{a} \in \mathbb{R}^l} L(\boldsymbol{a}, \boldsymbol{\lambda}^*) = \min_{\boldsymbol{a} \in \mathbb{R}^l} \left(f(\boldsymbol{a}) + \sum_{i=1}^{m} \lambda_i^* h_i(\boldsymbol{a}) \right) \leq \left(f(\boldsymbol{a}^*) + \sum_{i=1}^{m} \lambda_i^* h_i(\boldsymbol{a}^*) \right) \leq f(\boldsymbol{a}^*).$$

Therefore, the inequality is an equality. From that we can conclude that

1. $\boldsymbol{a}^*$ minimizes $L(\boldsymbol{a}, \boldsymbol{\lambda}^*)$ over $\mathbb{R}^l$. The first order conditions for a minimum imply stationarity.
2. $\sum_{i=1}^{m} \lambda_i^* h_i(\boldsymbol{a}^*) = 0$ which is only possible if the complementarity condition holds.

It remains to show that if $\boldsymbol{a}^*$ and $\boldsymbol{\lambda}^*$ satisfy the KKT conditions, then they are solutions to the primal and dual problem, respectively. The stationarity condition implies[1] that $g(\boldsymbol{\lambda}^*) = f(\boldsymbol{a}^*) + \sum_{i=1}^{m} \lambda_i^* h_i(\boldsymbol{a}^*)$ which (by complementarity) coincides with $f(\boldsymbol{a}^*)$. Thus, the duality gap is zero, and it follows that $\boldsymbol{a}^*$ and $\boldsymbol{\lambda}^*$ are primal and dual solutions, respectively. □

There is another dual problem which is important for our developments, which is called **Wolfe dual**

$$\max_{\boldsymbol{a} \in \mathbb{R}^l, \boldsymbol{\lambda} \in \mathbb{R}^m} L(\boldsymbol{a}, \boldsymbol{\lambda}) \quad \text{w.r.t. the constraint } \mathbf{0} = \nabla f(\boldsymbol{a}) + \sum_{i=1}^{m} \lambda_i \nabla h_i(\boldsymbol{a}), \lambda_i \geq 0.$$

One can show the following result (cf. Wolfe [1961]).

Theorem 2. *If $\boldsymbol{a}^*$ solves the primal problem and Slater's condition is satisfied, then there exists a solution to the Wolfe dual problem of the form $(\boldsymbol{\lambda}^*, \boldsymbol{a}^*)$ such that $f(\boldsymbol{a}^*) = L(\boldsymbol{a}^*, \boldsymbol{\lambda}^*)$.*

[1] Recall that if a function $g : \mathbb{R}^l \to \mathbb{R}$ is convex and differentiable and its gradient vanishes at a point, then this point is a global minimizer of the function, because $g(y) \geq g(x) + \nabla g(x)'(y - x)$ holds for every y and every x.

Hence, essentially, we can instead work with the Wolfe dual under our assumptions, which is more convenient for our purpose.

2.4 Dual formulation of the optimization problem for support vector machines

Reconsider the quadratic optimization problem defining the support vector machine classifier, i.e.,

$$\min_{\beta_0,\boldsymbol{\beta},\boldsymbol{\xi}\in\mathbb{R}^n} \|\boldsymbol{\beta}\|_2^2/2 + C\sum_{i=1}^n \xi_i \quad \text{w.r.t.} \quad Y_i(\beta_0 + \boldsymbol{\beta}'\boldsymbol{X}_i) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \text{ for every } i = 1,\ldots,n. \tag{6}$$

The Lagrange function to this problem (note the sign change), is

$$L(\beta_0, \boldsymbol{\beta}, (\boldsymbol{\lambda}, \boldsymbol{\gamma})) = \|\boldsymbol{\beta}\|_2^2/2 - C\sum_{i=1}^n \xi_i + \sum_{i=1}^n \lambda_i(Y_i(\beta_0 + \boldsymbol{\beta}'\boldsymbol{X}_i) - (1 - \xi_i)) - \sum_{i=1}^n \gamma_i\xi_i.$$

One can show that the Wolfe dual problem matches the solutions to the primal problem (cf. the computation in Section 4.4.1 of Richter [2019]), and that this dual problem can furthermore be re-formulated as

$$\min_{\boldsymbol{\lambda}\in\mathbb{R}^n} \boldsymbol{\lambda}'\boldsymbol{Q}\boldsymbol{\lambda} - \sum_{i=1}^n \lambda_i \quad \text{w.r.t. the constraint} \quad \sum_{i=1}^n Y_i\alpha_i = 0, 0 \leq \lambda_i \leq C \text{ for every } i = 1,\ldots,n, \tag{7}$$

where the ij-th coordinate of the matrix $\boldsymbol{Q}$ is defined via

$$Q_{ij} = Y_iY_j\boldsymbol{X}_i'\boldsymbol{X}_j.$$

Note that this formulation is completely free of $\boldsymbol{\gamma}$ and that it is formulated as a minimization problem (which can easily be achieved through multiplying by -1). If we solve the dual problem, this delivers $\boldsymbol{\lambda}^*$. By the stationarity part of the KKT conditions, we obtain that the gradient of the Lagrange function w.r.t. $\boldsymbol{\beta}$ and evaluated at $\boldsymbol{\lambda}^*$ must be zero at $\boldsymbol{\beta}^*$. If we compute this gradient, this results in the equation

$$\boldsymbol{\beta}^* = \sum_{i=1}^n \lambda_i^* Y_i \boldsymbol{X}_i.$$

Similarly we can show that $\beta_0^* = Y_{i'} - \boldsymbol{X}_{i'}'\boldsymbol{\beta}^*$, where i' is chosen such that $\lambda_{i'} \in (0, C)$. Hence, we can determine the solution to the initial problem through solving the dual problem.

We make the following observations:

1. The dual problem delivers a solution to the primal problem.
2. The values of λ_i^* (i.e., the optimal values in the dual formulation) have the following interpretation:
   - $\lambda_{i'} \in (0, C)$: $\boldsymbol{X}_i$ is correctly classified and is a support vector (i.e., lies at the boundary of the "road" corresponding to the hyperplane defined by the classifier).
   - If $\hat{\lambda}_i = 0$, the vector $\boldsymbol{X}_i$ is correctly classified, but it does lie on the "road".

- If $\hat{\lambda}_i = C$, then $\boldsymbol{X}_i$ is incorrectly classified. In this case, we also call this vector a support vector, because it determines the hyperplane.

2.5 Nonlinearities

The dual formulation is particularly important to incorporate nonlinearities. At first sight, this just means that instead of the predictors $\boldsymbol{X}_i$, we work with a nonlinear function $h : \mathbb{R}^p \to \mathbb{R}^q$ of the predictors: $\tilde{\boldsymbol{X}}_i = h(\boldsymbol{X}_i)$. Typically, q is much larger than p, because we are never really sure what the "right" nonlinear function is that we should work with. One example would be the function

$$h(\boldsymbol{x}) = (\boldsymbol{x}, x_1^2, \ldots, x_p^2, x_1 x_2, \ldots, x_{p-1} x_p)'.$$

Here, in addition to the predictors themselves, we would include polynomials of order 2. Higher order polynomials are possible too, which in turn increases the dimension q.

In any case, we can apply all methods we discussed so far to $\tilde{\boldsymbol{X}}_i$ instead of $\boldsymbol{X}_i$. Classification rules that previously lead to linear classifiers, such as support vector machines, LDA and logistic regression based classifiers, are no longer linear in $\boldsymbol{X}_i$ (but only linear in $\tilde{\boldsymbol{X}}_i$).

For example, the support vector machine classifier would then take the form $\text{sign}(\beta_0^* + \boldsymbol{\beta}^* h(\boldsymbol{X}_i))$, which can be highly nonlinear.

You might ask yourself: why bother. A simple example will bring some motivation. Look at the data in Figure 8. Clearly, there is a (nonlinear!) pattern. Would we try to use a linear classifier: rather not! But what can we do?

```
XN <- matrix(runif(500, -1, 1), nrow = 250, ncol = 2)
XN <- XN[abs(XN[,1]^2 + XN[,2]^2 - 1/2) > 0.2,]
y <- ((XN[,1]^2 + XN[,2]^2) < 1/2)*1
plot(XN, xlab = "X1", ylab = "X2")
points(XN[y == 0,], col = "red")
points(XN[y == 1,], col = "blue")
s <- seq(0, 2*pi, length = 10000)
lines(sin(s)/sqrt(2), cos(s)/sqrt(2), lty = 2, lwd = 2)
```

Actually, we can perfectly separate the points, but we cannot separate them linearly. To understand which transformation of the h we should use, it is beneficial to think about where these points could come from. That is, we want to think about which points $h(\boldsymbol{X}_i)$ could have given rise to these points. Thinking

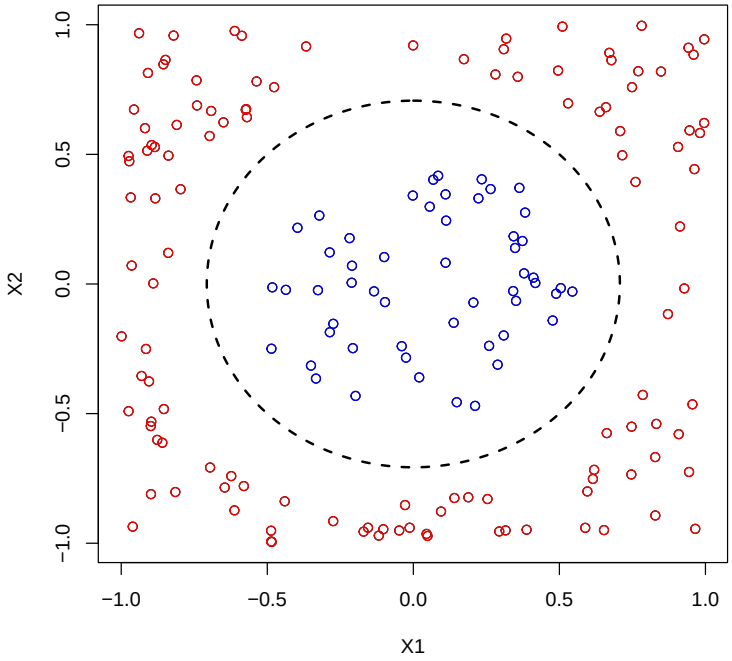

Figure 8: Nonlinearly separated data.

about that could lead to a picture such as the one we give next. Here, $q = 3$ and we use the function $h(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2)$. Actually, this is a pure thought experiment, but proceeding this way, we embed the points into the 3-dimensional space in such a way that they become linearly separable there. The result is shown in Figure 9.

```
library("scatterplot3d")
XN3 <- cbind(XN, XN[,1]^2 + XN[,2]^2)
col.N <- y
col.N[y == 1] <- "blue"
col.N[y == 0] <- "red"
sp <- scatterplot3d(XN3, xlab = "X1", ylab = "X2", zlab = "X12 + X22",
color = col.N, angle = 10, grid=TRUE, box=FALSE)
sp$plane3d(1/2, 0, 0)
```

Obviously, this is just a toy example, because in this case $p = 2$ and we could just plot the data, take our pencil and draw the regions. However, if $p > 2$ this is no longer possible, and we need to find some automatic way of dealing with that. In any case, we can (in this example) linearly separate the points if we work with

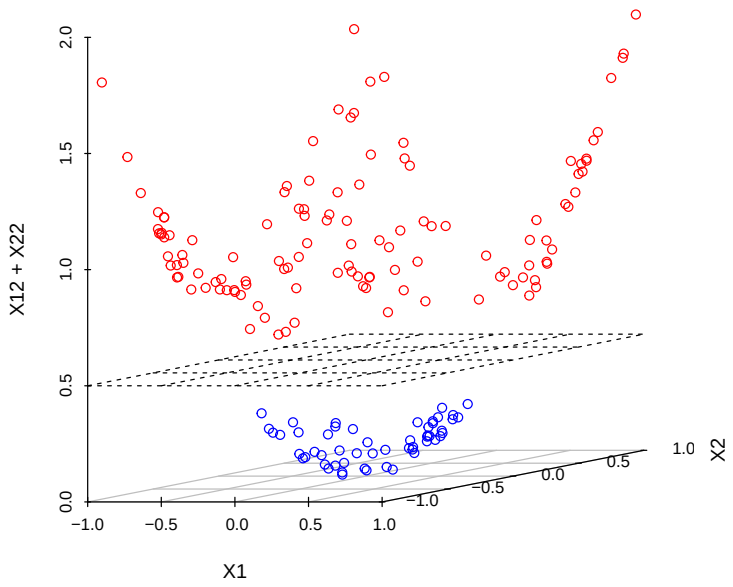

Figure 9: Support vector machine classifier applied to generated features.

the function $h(\boldsymbol{x}) = (x_1, x_2, x_1^2 + x_2^2)$. We can also separate them, if we have more information, such as the function $h(\boldsymbol{x}) = (x_1, x_2, x_1^2, x_2^2, x_2 x_2)$. This adds polynomials of order 2 to the feature vector.

How would we incorporate this into support vector machines. The answer is simple, given h the classifier applied to the generated features $h(\boldsymbol{X}_i)$ instead of $\boldsymbol{X}_i$ is obtained by solving the optimization problem:

$$\min_{\beta_0, \boldsymbol{\beta}, \boldsymbol{\xi} \in \mathbb{R}^n} \|\boldsymbol{\beta}\|_2^2/2 + C \sum_{i=1}^{n} \xi_i \quad \text{w.r.t.} \quad Y_i(\beta_0 + \boldsymbol{\beta}' h(\boldsymbol{X}_i)) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \text{ for every } i = 1, \ldots, n. \tag{8}$$

The only thing that changes, is that we replace $\boldsymbol{X}_i$ by $h(\boldsymbol{X}_i)$. Furthermore, the dimension of the vector $\boldsymbol{\beta}$ is then not p but q, the length of $h(\boldsymbol{X}_i)$.

Support vector machine classifier applied to generated features

Given a function $h : \mathbb{R}^p \to \mathbb{R}^q$, fix a number $C \geq 0$. If $(\beta_0^*, \boldsymbol{\beta}^*)$ is a solution to the problem (8), the corresponding support vector machine classifier is

$$\hat{f}_{n,SV,C}(\boldsymbol{x}) = \text{sign}(\beta_0^* + h(\boldsymbol{x})' \boldsymbol{\beta}^*).$$

Let's now check if (with the right) transformation, i.e., polynomials, we can achieve the correct separation.

Again, let's use the function `svm`, because there this is already built-in with the option `kernel = polynomial`. The result is shown in Figure 10. To illustrate the classification regions, we use the out-of-the-box plotting function for `svm` objects the package **e1071** is equipped with.

```
XN_dat <- data.frame("y"= factor(y), "x1" = XN[,1], "x2" = XN[,2])
fit.svm3 <- svm(factor(y)~., data = XN_dat, kernel = "polynomial", degree = 2, cost = 1)
plot(fit.svm3, XN_dat)
```

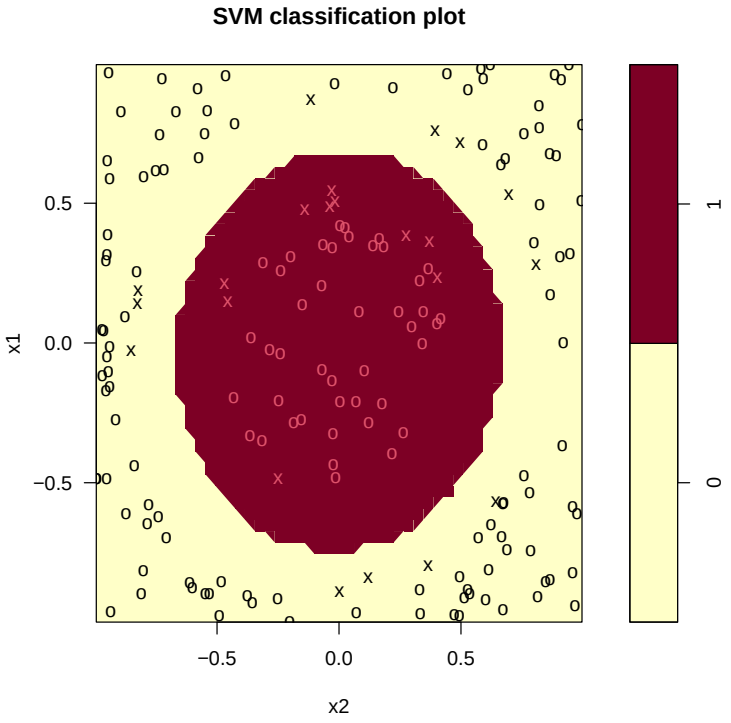

Figure 10: Support vector machine classifier.

We have seen that to classify this type of data correctly, we need to work with transformations. In `R` there is the option `kernel = ....` Where does that come from, and what do the other types of kernel actually mean? Let's take a look into that!

There is some caveat here: we used that we know exactly that the polynomial order we need to separate the data is 2. If we work with `degree = 3` and do not choose `C` carefully (or adapt the number of iterations), we end up with what is shown in Figure 11 below. The reason we do this is to emphasize that the tuning parameters one chooses: `C` or the degree of the polynomial etc. **do** matter.

```
XN_dat <- data.frame("y"= factor(y), "x1" = XN[,1], "x2" = XN[,2])
fit.svm3 <- svm(factor(y)~., data = XN_dat, kernel = "polynomial", degree = 3, cost = 1)
plot(fit.svm3, XN_dat)
```

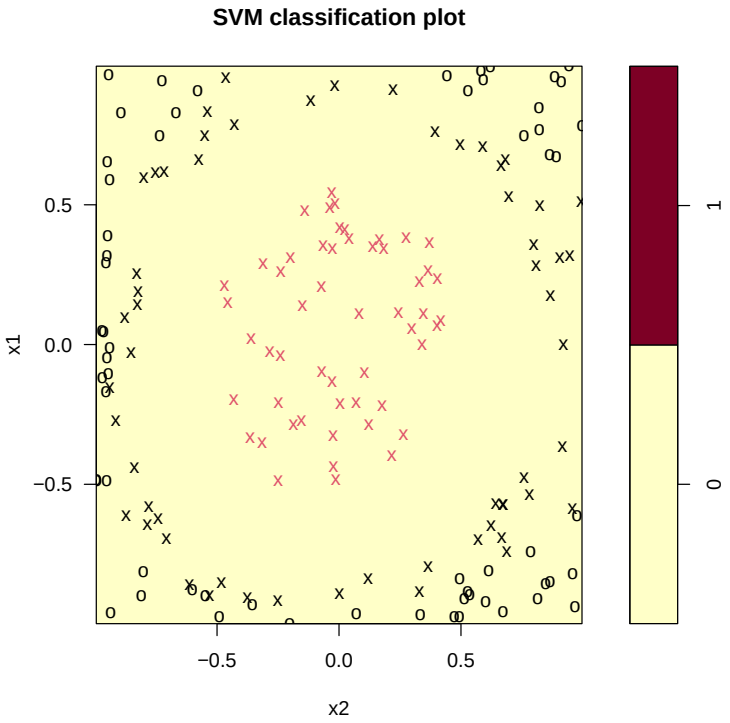

Figure 11: Support vector machine classifier did not really converge to something useful ... .

As an alternative to the function `svm` one can also use the package **kernlab**. More specifically, its function `ksvm`.

2.6 Kernel trick

Now, we know how to apply support vector machines to nonlinear transformations of the feature vector. In practice, the obvious question is: how do we choose this function. To see how this is typically formulated, we need to come back to the dual program that corresponds to the SVM classifier. Let's recall that this program was defined as:

$$\min_{\boldsymbol{\lambda}\in\mathbb{R}^n} \boldsymbol{\lambda}'\boldsymbol{Q}\boldsymbol{\lambda} - \sum_{i=1}^{n}\lambda_i \quad \text{w.r.t. the constraint} \quad \sum_{i=1}^{n} Y_i\alpha_i = 0, 0 \leq \lambda_i \leq C \text{ for every } i = 1,\ldots,n, \tag{9}$$

where the ij-th coordinate of the matrix $\boldsymbol{Q}$ was defined via

$$Q_{ij} = Y_i Y_j \boldsymbol{X}_i' \boldsymbol{X}_j,$$

and which gave rise to the classifier via

$$\boldsymbol{\beta}^* = \sum_{i=1}^n \lambda_i^* Y_i \boldsymbol{X}_i,$$

and

$$\beta_0^* = Y_{i'} - \boldsymbol{X}_{i'}' \boldsymbol{\beta}^*,$$

where i' was chosen such that $\lambda_{i'} \in (0, C)$.

What happens if we replace $\boldsymbol{X}_i$ by $h(\boldsymbol{X}_i)$? That's easy, we just need to replace $\boldsymbol{X}_i$ by $h(\boldsymbol{X}_i)$ throughout. Hence, the dual program for generated predictors becomes

$$\min_{\boldsymbol{\lambda} \in \mathbb{R}^n} \boldsymbol{\lambda}' \boldsymbol{Q}_h \boldsymbol{\lambda} - \sum_{i=1}^n \lambda_i \quad \text{w.r.t. the constraint} \quad \sum_{i=1}^n Y_i \alpha_i = 0, 0 \leq \lambda_i \leq C \text{ for every } i = 1, \ldots, n, \tag{10}$$

where the ij-th coordinate of the matrix $\boldsymbol{Q}_h$ is defined via

$$\boldsymbol{Q}_{h,ij} = Y_i Y_j h(\boldsymbol{X}_i)' h(\boldsymbol{X}_j),$$

and which gives rise to the classifier

$$\boldsymbol{x} \mapsto \text{sign}(\beta_0^* + \boldsymbol{\beta}^* h(\boldsymbol{x}))$$

where

$$\boldsymbol{\beta}^* = \sum_{i=1}^n \lambda_i^* Y_i h(\boldsymbol{X}_i),$$

and

$$\beta_0^* = Y_{i'} - h(\boldsymbol{X}_{i'})' \boldsymbol{\beta}^*,$$

where i' is chosen such that $\lambda_{i'} \in (0, C)$. Note that we can write the classifier as

$$\boldsymbol{x} \mapsto \text{sign} \left(Y_{i'} - \sum_{i=1}^n \lambda_i^* Y_i h'(\boldsymbol{X}_{i'}) h(\boldsymbol{X}_i) + \sum_{i=1}^n \lambda_i^* Y_i h'(\boldsymbol{X}_i) h(\boldsymbol{x}) \right)$$

The nice thing about this observation is that even though we typically increase the number of features if we work with functions h, the optimization problem is still a problem in n variables! That is, the complexity of the optimization problem does not increase if we add variables!

What is more, the features $\tilde{X}_i$ enter the optimization problem only through $h(\boldsymbol{X}_i)' h(\boldsymbol{X}_j)$ and enter the decision rule only through $h'(\boldsymbol{X}_i) h(\boldsymbol{x})$. Hence, we do not really need to compute $h(\boldsymbol{X}_i)$ but only need to

know the before-mentioned terms. In particular, it follows that we only need to fix all values of the function

$$\mathsf{K}(\boldsymbol{x}, \boldsymbol{z}) := h(\boldsymbol{x})' h(\boldsymbol{z}) \quad \text{for all } \boldsymbol{x} \text{ and } \boldsymbol{z} \text{ in } \mathbb{R}^p.$$

We do not really need to explicitly write down h, but we only need to decide on which function K we work with. Visually, the function h describes the inner product between two transformed feature vectors.

A function $\mathsf{K} : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ is called a kernel. Typically, one imposes the following conditions on a kernel

1. **Symmetry**: $\mathsf{K}(\boldsymbol{x}, \boldsymbol{z}) = \mathsf{K}(\boldsymbol{x}, \boldsymbol{z})$ for all $\boldsymbol{x}$ and $\boldsymbol{z}$ in $\mathbb{R}^p$.
2. **Continuity**: the function $(\boldsymbol{x}, \boldsymbol{z}) \mapsto \mathsf{K}(\boldsymbol{x}, \boldsymbol{z})$ is continuous.
3. **Positive definiteness**: For all n and all $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^p$ the matrix with ij-th element $\mathsf{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is positive definite.

Given such a kernel function, we can construct the corresponding matrix

$$\boldsymbol{Q}^{\mathsf{K}} \text{ the ij-th coordinate of which is defined as } Y_i Y_j \mathsf{K}(\boldsymbol{X}_i, \boldsymbol{X}_j).$$

The nonlinear SVM classifier with kernel K (in its dual version) is then obtained as follows:

Nonlinear SVM classifier with kernel K

Let $\boldsymbol{\lambda}^*$ be the solution to the minimization problem

$$\min_{\boldsymbol{\lambda} \in \mathbb{R}^n} \boldsymbol{\lambda}' \boldsymbol{Q}^{\mathsf{K}} \boldsymbol{\lambda} - \sum_{i=1}^n \lambda_i \quad \text{w.r.t. the constraint} \quad \sum_{i=1}^n Y_i \alpha_i = 0, 0 \leq \lambda_i \leq C \text{ for every } i = 1, \ldots, n. \tag{11}$$

Define the classifier via

$$\boldsymbol{x} \mapsto \operatorname{sign}\left(\beta_0^* + \sum_{i=1}^n \lambda_i^* Y_i \mathsf{K}(\boldsymbol{X}_i, \boldsymbol{x}) \right) \quad \text{with } \beta_0^* = Y_{i'} - \sum_{j=1}^n \lambda_i^* Y_j \mathsf{K}(\boldsymbol{X}_j, \boldsymbol{X}_{i'}),$$

where i' is chosen such that $\lambda_i^* \in (0, C)$.

Frequently used kernels are:

1. **Linear kernel**

$$\mathsf{K}^{lin}(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x}' \boldsymbol{z}.$$

2. **Polynomial kernel** of degree l:

$$\mathsf{K}^{poly,l}(\boldsymbol{x}, \boldsymbol{z}) = (1 + \boldsymbol{x}\boldsymbol{z}') \times \ldots (1 + \boldsymbol{x}\boldsymbol{z}'),$$

and the multiplication is l done l times.

3. **Gaussian kernel** with "slope" $\gamma > 0$:

$$\mathsf{K}^{gauss,\gamma}(\boldsymbol{x}, \boldsymbol{z}) = \exp(-\gamma \|\boldsymbol{x} - \boldsymbol{z}\|^2).$$

4. **Sigmoid kernel** with parameters $\kappa_1 > 0$ and $\kappa_2 \in \mathbb{R}$:

$$\mathsf{K}^{sigm,\kappa}(\boldsymbol{x}, \boldsymbol{z}) = \tanh(\kappa_1 \boldsymbol{x}'\boldsymbol{z} + \kappa_2).$$

This is the final support vector machine classifier! In practice, one now has to choose the kernel, which again can be done through a validation procedure. Note that the above mentioned kernels are only 4 different ones. Hence, we can simply apply the 4 different classifiers to a validation set, and choose the method that performs best there. We will see how this works in the example below.

3 Data example

We now take a look at the performance of different support vector machines when applied to a real-world problem (although a fairly well defined one). The problem is the one of banknote authentication. The dataset Dua and Graff [2017] contains information obtained by scanning fraudulent and genuine banknotes. The underlying technique they use to extract the features is called wavelet transform, which essentially expands the picture information (a 2-dimensional function) into what is called a wavelet basis (which is a basis for a function space). The coefficients in this expansion then allow one to reconstruct (fairly well) the function itself. One can now take a look at those (sequences of) coefficients and extract properties thereof, such as their (i) variance, (ii) skewness, (iii) curtosis, or (iv) entropy, which are the features available for each banknote in the database. Furthermore, we have a class label that equals 1 if the banknote is genuine, and equals 0 else (that's my 50:50 guess, as the dataset does not provide information on that ... but for us this

is irrelevant). A publication related to the dataset can is Lohweg et al. [2013].

Let's download the dataset from this link[2]. Rename the dataset as `notes.txt` for convenience.

```
set.seed(123)
notes <- read.table("notes.txt", header = FALSE, sep = ",")
names(notes) <- c("var", "skew", "curt", "entr", "y")
x <- notes[,1:4]
y <- as.factor(notes[,5])
```

Lets randomly select a training sample, and check whether it is balanced w.r.t. the dependent variable.

```
n <- dim(notes)[1]
tsa <- sample(1:n, size = floor(0.8*n), replace = FALSE)
xtr <- x[tsa,]
ytr <- y[tsa]
mean(ytr == 1)
```

```
## [1] 0.4412033
```

We now have more than 2 features, so we can no longer plot it in a single figure as easily. Nevertheless, we can plot bivariate scatterplots for all combinations of features. There are 6 of them.

```
par(mfrow = c(3, 2))
for(i in 1:3){
  for(j in (i+1):4){
    plot(xtr[,c(i,j)])
    points(xtr[ytr == 0, i], xtr[ytr == 0, j], col = "red")
    points(xtr[ytr == 1, i], xtr[ytr == 1, j], col = "blue")
  }
}
```

```
par(mfrow = c(1, 1))
```

The relationship between variables 2 and 4 seems quadratic. It therefore could be beneficial, to add second order terms of the two variables as features to the dataset. We'll skip this possibility now (partly because polynomial functions are accounted for when using polynomial kernels in `svm`; however adding such regressors

[2]By the way, the data repository this site contains a host of interesting datasets with which you can improve your modeling skills.

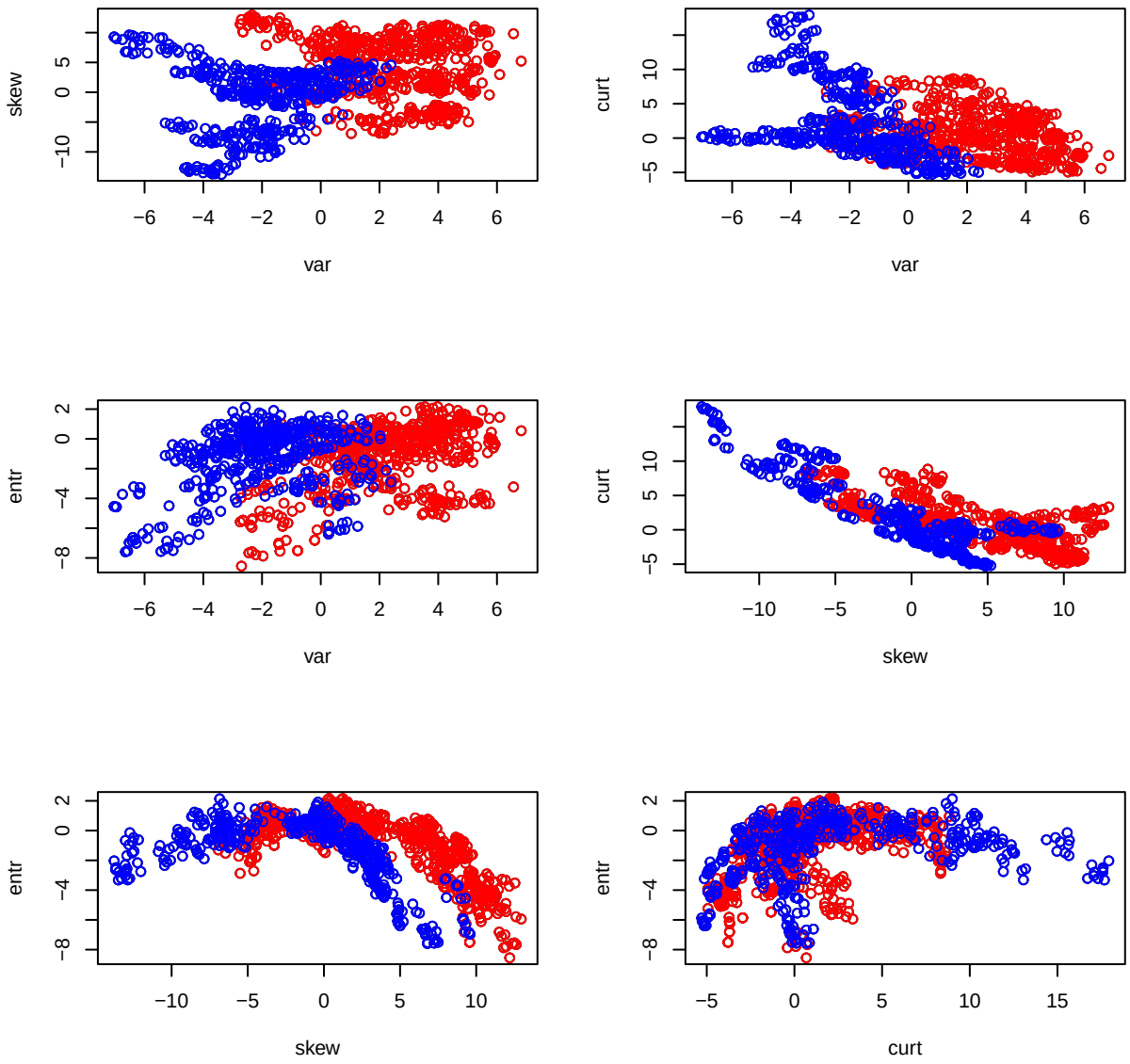

Figure 12: Bivariate feature plots for the banknote authentication dataset. The axes in the individual plots show which variables are plotted. Blue indicates that the outcome equals 1

could be worth trying).

Next, let's fit support vector machines defined through different kernels.

```
data.note <- data.frame("y" = ytr)
data.note <- cbind(data.note, xtr)
sv.lin <- svm(y ~ ., data = data.note, kernel = "linear")
sv.pol <- svm(y ~ ., data = data.note, kernel = "polynomial")
sv.rad <- svm(y ~ ., data = data.note, kernel = "radial")
sv.sig <- svm(y ~ ., data = data.note, kernel = "sigmoid")

sv.tre.lin <- predict(sv.lin, data = data.note)
sv.tre.pol <- predict(sv.pol, data = data.note)
sv.tre.rad <- predict(sv.rad, data = data.note)
sv.tre.sig <- predict(sv.sig, data = data.note)

mean(sv.tre.lin == data.note$y)
```

```
## [1] 0.9845032
```

```
mean(sv.tre.pol == data.note$y)
```

```
## [1] 0.9881495
```

```
mean(sv.tre.rad == data.note$y)
```

```
## [1] 1
```

```
mean(sv.tre.sig == data.note$y)
```

```
## [1] 0.7721057
```

Let's compute the testing error, by applying the support vector machines to the testing sample.

```
yte <- y[-tsa]
xte <- x[-tsa,]
data.note.te <- data.frame("y" = yte)
data.note.te <- cbind(data.note.te, xte)

sv.p.lin <- predict(sv.lin, newdata = xte)
```

```
sv.p.pol <- predict(sv.pol, newdata = xte)
sv.p.rad <- predict(sv.rad, newdata = xte)
sv.p.sig <- predict(sv.sig, newdata = xte)

mean(sv.p.lin == yte)
```

```
## [1] 0.9854545
```

```
mean(sv.p.pol == yte)
```

```
## [1] 0.9818182
```

```
mean(sv.p.rad == yte)
```

```
## [1] 1
```

```
mean(sv.p.sig == yte)
```

```
## [1] 0.7781818
```

It is somewhat surprising to see that we obtain a perfect separation using the radial basis support vector machine. It works perfectly fine, not only on the training data (which would make one believe that there is some serious overfitting going on), but also in terms of its testing error, which concerns an out of sample prediction.

That we can actually predict with uncertainty also has to do with the problem under consideration. At the end of the day, we make physical measurements with very small error. In such cases, it is reasonable to assume that there is some "distance" between genuine banknotes and fraudulent ones. The radial basis function seems to be able to detect what matters.

In the paper cited above, the authors also consider a LD classifier. Let's reconsider that one also here, to see how it compares to the svm classifiers we have obtained through `svm`.

```
library(MASS)
lda.note <- lda(y ~ ., data = data.note)
qda.note <- qda(y ~ ., data = data.note)
pred_lda.note <- predict(lda.note, newdata = xte)$class
pred_qda.note <- predict(qda.note, newdata = xte)$class
```

```
mean(pred_lda.note == yte)
```

```
## [1] 0.9818182
```

```
mean(pred_qda.note == yte)
```

```
## [1] 0.9854545
```

The methods also do very well in this classification task. The support vector machine classifier cannot be beaten, however.

4 Exercises

Both exercises count 2.5 points, amounting to a total of 5 maximally achievable points.

1. For the case where $p = 1$ (i.e., there is only a single feature) think about how to implement the kNN algorithm efficiently (hint: you may want to first sort the features in the training sample). Formulate your algorithm, and then write a function in `R` (or a similar language) that implements the algorithm you come up with (it is *not allowed* to use built-in `R` functions that return the kNN classifier or related quantities).
2. Based on your implementation of the first problem, write an `R` function that determines k in the kNN classifier via cross-validation (i.e., in a data-driven way). Apply your function to a data set of your choice (which can be simulated data), and compare the resulting k with the "standard choice" $\lfloor\sqrt{n}\rfloor$. Compute the confusion matrix for the $\hat{k}$ determined via cross validation and for the "standard choice" and compare.

References

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer: New York, 2009.

Volker Lohweg, Jan Leif Hoffmann, Helene Dörksen, Roland Hildebrand, Eugen Gillich, Jürg Hofmann, and Johannes Schaede. Banknote authentication with mobile devices. In *Media Watermarking, Security, and Forensics 2013*, volume 8665, page 866507. International Society for Optics and Photonics, 2013.

Stefan Richter. *Statistisches und maschinelles Lernen*. Springer: Berlin, 2019.

Philip Wolfe. A duality theorem for non-linear programming. *Quarterly of applied mathematics*, 19(3): 239–244, 1961.