1. Introduction and k-nearest neighbors

1.1. Answer

Supervised, unsupervised, and reinforcement learning

1.2. Answer

A feature describes a certain aspect of an object or phenomenon that we would like to classify. It is usually expressed as a numerical value.

1.3. Answer

a) $\mathbf{x}$ is a vector of input features, $w_1, w_2, \ldots, w_n$ are weights or parameters of the classifier that are adjusted in the training phase, and Ω is a set of discrete class labels.

b) In classification we want to learn to predict a discrete class label. In regression we want to learn to predict a continuous variable, for example a temperature, probability, stock price etc.

c) The parameters $w_1, w_2, \ldots, w_n$ are adjusted by an optimization procedure so that the system obtains the optimal skill to classify training data.

1.4. Answer

a) A learning method generalizes well if it performs well on previously unseen data, that is, it can generalize what it has learned on training data to new data.

b) The generalization error can be estimated by testing the classifier on a test data set that was not used in the training.

c) To avoid overfitting.

d) The algorithm has overtrained if it performs much better on the training data than on a test data set which it has not seen before.

1.5. Answer

The accuracy is the number of correctly classified examples divided by the total number of samples, i.e., $\frac{80+90}{80+90+20+30} = 0.77$.

1.6. Answer

Divide the data set into equal parts P1, P2 and P3 with 300 samples each. Then train and evaluate 3 times as follows:

- Train using P1 and P2 and evaluate using P3.
- Train using P1 and P3 and evaluate using P2.
- Train using P2 and P3 and evaluate using P1.

The total performance is the average classification accuracy of the 3 runs.

1.7. Answer

Advantages are that k-nearest neighbors requires no training and it is easy to implement. Disadvantages are that one must store all training examples, and it takes a long time to classify if the training data set is large, as the distance to all training examples must be calculated.

1.8. Answer

In the case of $k = 1$, X will belong to a black dot, since this is the closest sample. When $k = 3$, X will be classified as a square because two of the three closest samples are squares. In the case of $k = 2$, the votes are even between the two classes and some confusion occurs. One solution is to classify the X as a dot since this is the closest sample.

2. Linear classifiers

2.1. Answer

A (hyper-)plane can be defined with the equation

$$\mathbf{w}^T\mathbf{x} + b = 0,$$

where $\mathbf{w}$ is a normal vector to the plane (i.e., orthogonal to the plane), $\mathbf{x}_p$ is a point in the plane (any point will do) and b is a scalar adjusting the offset of the plane from the origin. If $b = 0$, the plane goes through the origin.
How should we interpret this equation? Consider a point $\mathbf{x}_p$ lying somewhere in the plane. Also consider a vector $\mathbf{x}'$ from $\mathbf{x}_p$ to an arbitrary point $\mathbf{x}$. If $\mathbf{x}$ lies in the plane, $\mathbf{x}'$ should not have any projection on the normal of the plane (right?). This gives us

$$\mathbf{w}^T(\mathbf{x} - \mathbf{x}_p) = 0.$$

This is also an equation for the plane; we just need to know the normal vector and an arbitrary point in the plane for the plane to be well defined. We can however rewrite the equation as

$$\mathbf{w}^T\mathbf{x} - \mathbf{w}^T\mathbf{x}_p = 0.$$

If we then name $-\mathbf{w}^T\mathbf{x}_p$ as b, we have returned to the original equation. We can consequently calculate b as

$$b = -\mathbf{w}^T\mathbf{x}_p.$$

If we anew consider another arbitrary point $\mathbf{x}$ and would like to know the distance to the plane, we can proceed like this:
Let $\mathbf{x}'$ be a vector from $\mathbf{x}_p$ to $\mathbf{x}$, i.e., $\mathbf{x}' = \mathbf{x} - \mathbf{x}_p$. (In this context, a point is the same thing as a vector from the origin to the coordinate of the point). Also let $\mathbf{x}_o$, be a vector orthogonal to the plane, going from the plane to $\mathbf{x}$. It is the length of this vector we seek! This vector is parallel to $\mathbf{w}$ and therefore projects its whole length on $\mathbf{w}$. As we can see in the picture, it is also obvious this projection on $\mathbf{w}$ has the same length as the projection of $\mathbf{x}'$ on $\mathbf{w}$. Consequently, we only need to calculate this projection

$$\mathbf{x}_o = \frac{\mathbf{x}'^T\mathbf{w}}{\mathbf{w}^T\mathbf{w}}\mathbf{w},$$

where the denominator is for the normalization of $\mathbf{w}$.
The distance to the plane, i.e., the length of this vector is

$$|\mathbf{x}_o| = \frac{\mathbf{x}'^T\mathbf{w}}{|\mathbf{w}|^2}|\mathbf{w}| = \frac{\mathbf{x}'^T\mathbf{w}}{|\mathbf{w}|} = \frac{(\mathbf{x} - \mathbf{x}_p)^T\mathbf{w}}{|\mathbf{w}|}$$

Note that the sign of the projection $\mathbf{x}'^T\mathbf{w}$ tells us what side of the plane the point lies on. If the projection is greater than zero we are on the side in the direction of the normal vector $\mathbf{w}$. If it is less than zero, we are on the other side. If the projection is zero, we are exactly in the plane.

2.2. Answer

a) See lecture notes!

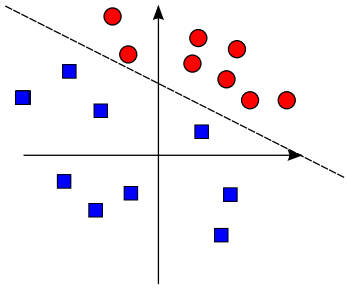
b) The "bias weight" makes it possible to move the discriminating hyperplane away from the origin.

c) The input value to the bias weight is typically 1, but can be any value except zero.

d) The bias weight offsets the decision boundary from the origin, so we need an example where the optimal boundary does not go through the origin, for example

2.3. Answer

In *batch learning*, all the training samples are used for updating the classifier. In *online learning*, the classifier is updated using one training sample at the time.



2.4. Answer

a) $\mathbf{w}^T = \begin{bmatrix} w_1 w_2 \dots w_n \end{bmatrix}$, $\quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$

$$\mathbf{y} = \mathbf{w}^T\mathbf{x} = w_1x_1 + w_2x_2 + \cdots + w_nx_n = \sum w_ix_i$$

$$\frac{\partial y}{\partial w_i} = x_i \,, \qquad \frac{\partial y}{\partial \mathbf{w}} = \mathbf{x}$$

b) $y = \sum_i \sum_j x_i W_{ij} x_j$, $\quad \frac{\partial y}{\partial x_k} = \sum_i x_i W_{ik} + \sum_j x_j W_{kj}$

我们假设x都是列向量，那么就应该是W.T*X而不是X.T*W,后者出来是一个行向量。

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \sum_i x_i W_{i1} + \sum_j x_j W_{1j} \\ \vdots \\ \sum_i x_i W_{in} + \sum_j x_j W_{nj} \end{bmatrix} = \mathbf{W}\mathbf{x} + \mathbf{W}^T\mathbf{x}.$$ Can be reduced to $2\mathbf{W}\mathbf{x}$ if $\mathbf{W}$ is symmetric.

c) $y = \left(\sqrt{w_1^2 + \cdots + w_n^2}\right)^4 = \left(w_1^2 + \cdots + w_n^2\right)^2$

$$\frac{\partial y}{\partial w_i} = 2(w_1^2 + \cdots + w_n^2) \cdot 2w_i = 4(\mathbf{w}^T\mathbf{w}) \cdot w_i$$

$$\frac{\partial y}{\partial \mathbf{w}} = 4(\mathbf{w}^T\mathbf{w})\mathbf{w}$$

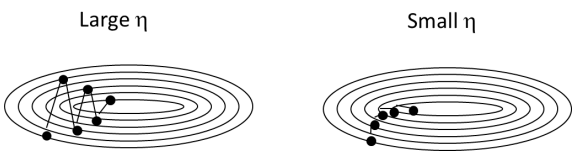
2.5. Answer

a) $\frac{\partial f}{\partial \mathbf{x}} = 6\mathbf{x} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ which gives $\mathbf{x} = \begin{bmatrix} -\frac{1}{6} \\ -\frac{1}{6} \end{bmatrix}$.

b) $\mathbf{x}_t = \mathbf{x}_{t-1} - \eta \frac{\partial f}{\partial \mathbf{x}}$ where $\frac{\partial f}{\partial \mathbf{x}} = 6\mathbf{x}_{t-1} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. We then get $\mathbf{x}_1 = \begin{bmatrix} 0.65 \\ 0.65 \end{bmatrix}$ and $\mathbf{x}_2 = \begin{bmatrix} 0.4050 \\ 0.4050 \end{bmatrix}$

2.6. Answer

The effect of a too long step length may be an oscillating behavior that may have the effect that we never converge to a local minimum. The effect of a too small step length is that we move very slowly towards the local minimum, i.e., the training takes a long time.



2.7. Answer

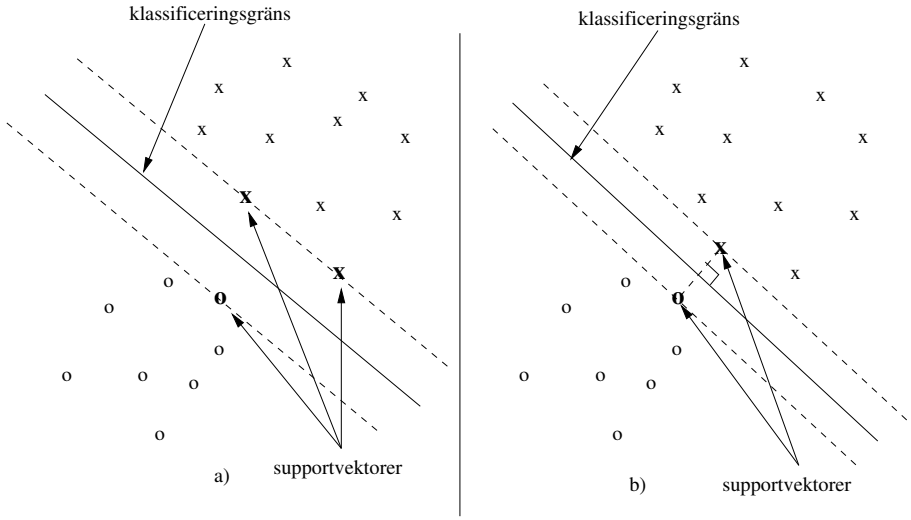
The cost function $\sum_{k=1}^{N} I(z_i \neq y_i)$ is piecewise flat and not differentiable at all points, so that we do not get any information from the gradient, which is required in gradient descent.

2.8. Answer

The maximum margin principle says that the decision boundary between two classes should be placed so that the distance between the boundary and the training data should be as large as possible, i.e., the error-margin should be maximized.

2.9. Answer

The optimal hyperplane is the line that best separates the two classes, i.e., the line that is as far away from the closest points as possible. The support vectors are the samples that are closest to this plane.



2.10. Answer

$\|\mathbf{w}\|$ is minimized and the equation holds with equality for the *support vectors*.

2.11. Answer

SVMs have good generalization properties and are usually better than e.g. backpropagation networks on small datasets. On large datasets, however, it gets computationally heavy since the kernel matrix grows quadratically with the number of samples.

3. Neural networks and nonlinear classifiers

3.1. Answer

a) The "XOR problem" has one class defined as the points $(-1, 1)$ and $(1, -1)$ and the other class at the points $(1, 1)$ and $(-1, -1)$. These classes cannot be separated with a linear classifier.

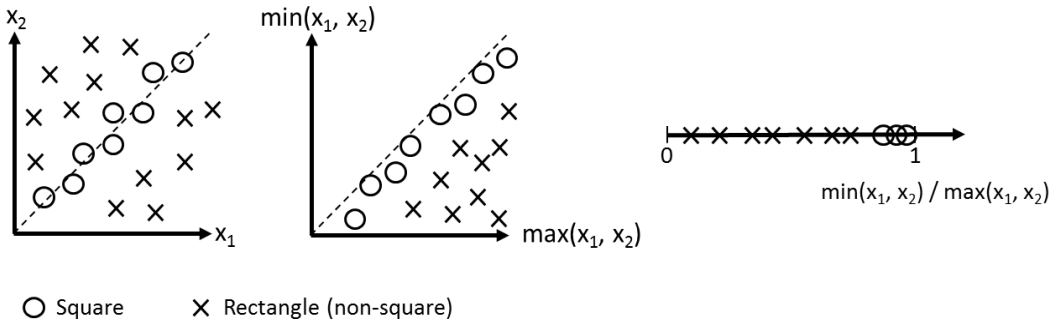
b) It is not linearly separable.

3.2. Answer

The probability that classes are linearly separable increases when the features are nonlinearly mapped to a higher dimensional feature space.

3.3. Answer

The figure shows three different ways this can be done.



- One can keep x_1 and x_2 as they are as features. The squares will then be located along the diagonal line because $x_1 \approx x_2$, while the non-square rectangles will be located away from the diagonal in the feature space. A non-linear classifier is required for this case, e.g., a neural network, SVM, ...
- One can create two new features $y_1 = \max(x_1, x_2)$ and $y_2 = \min(x_1, x_2)$. The feature space will then look like the middle plot, and a linear classifier will be enough.
- Another option is to calculate a new 1D-feature as the ratio $y = \frac{\min(x_1, x_2)}{\max(x_1, x_2)}$, shown in the right figure. Again, a simple linear classifier will do (i.e., a single threshold in this case).

3.4. Answer

For example, one may used a neural network with 5 input nodes, one for the temperature of each of the 5 past days. Given historical data over 6-day periods, one can use the first 5 days as training data and the 6th day as the correct answer. The neural network can in this way be trained to predict tomorrows temperature based on the past 5 days.

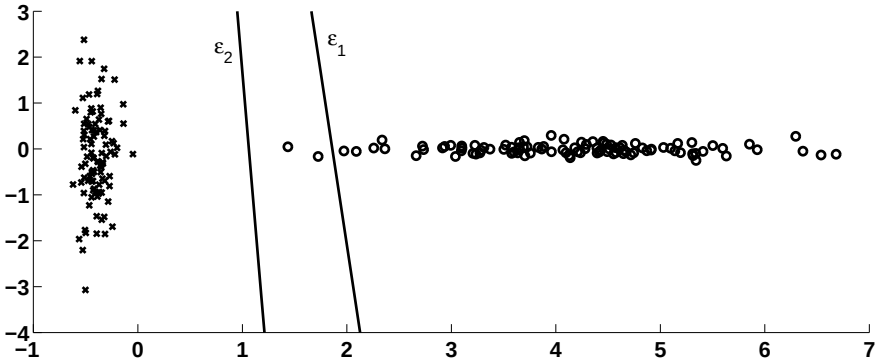
3.5. Answer

a) Not really, tanh(x) never becomes ± 1, just very close. If you try to use $+1$ (or -1) you may force the weights to very large values and risk the robustness of the network. Try using a slightly smaller value. (In practice ± 1 often works though...)

b) The activation function in the output layer can not provide more advanced class boundaries or such. However it can bound the output values which is a good protection against outliers; e.g. very deviating training samples will not affect the solution as much with some kind of sigmoid activation function.

c) The problem here is that we have two linearly separable classes which can be classified using a simple threshold on the horizontal axis. They do however have different distributions along this axis, a factor that can confuse our training if we are not careful. The first error function ϵ_1 is sensitive to the distance of the samples. The effect is that the right most cluster will have a higher impact in the

total error, and the boundary will shift closer to the right class. This is somewhat fixed with ϵ_2 were $\tanh$ quickly saturates to about ± 1, thus limiting the effect of the different distributions. The figure below shows the result after training a single neuron using the two functions. Here we can see that the theory holds true.



3.6. Answer

a)

$$\Delta w(t) = \alpha \Delta w(t-1) - \eta \frac{\partial \epsilon(t)}{\partial w}$$

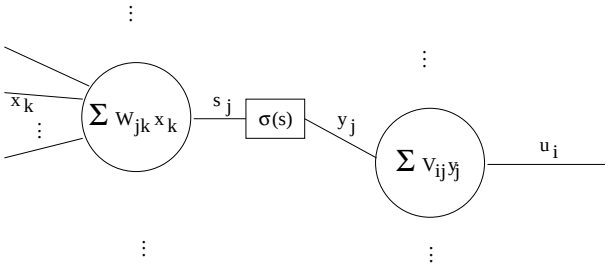
b) You have to answer this during the back-prop assignment.

3.7. Answer

Since both the Euclidian and RBF metrics given above are radially symmetric around μ they have problems with the radial asymmetric form of the data. By including the covariance matrix the contributions of differences in x or y can be tuned in such a way that the metric better fits the data.

3.8. Answer

a) We want to find the gradient direction in which the error decreases the most. This direction tells us how much we should change each of the individual parameters (weights) W_{jk} and V_{ij}. The picture



shows a small part of a general backprop network. The used indexes are shown in the picture. There is also an index μ in superscript, e.g. $\mathbf{x}^\mu$ denotes the μ:th training sample (a column vector). We assume N training samples with dimension D, M neurons in the hidden layer and C neurons ($= C$ classes) in the output layer. In order to train the network we have to run all training samples (in the off-line/batch version) through it and for each sample calculate every u_i, y_j and so on. When we have done that, we will calculate the error gradient and update our weights V and W by taking a small step in the negative gradient direction, just as in the one layer case. (We negate the error gradient because we want to decrease the error, not increase it!) *NB! Please note the indexing of the weights! All input weights connecting to a certain neuron lies along a row in the weight matrix. When we discussed the simple perceptron, the weights were stored in a column vector...so we use the transposed version here! (Adapting to the convention used by Haykin.)*

In order to calculate the error gradient, let us first line up the whole chain of functions involved:

$$\varepsilon = \frac{1}{N}\sum_{i=1}^{C}\sum_{\mu=1}^{N}\varepsilon_i^\mu \tag{1}$$

$$= \frac{1}{N}\sum_{i=1}^{C}\sum_{\mu=1}^{N}\left(d_i^\mu - u_i^\mu\right)^2 \tag{2}$$

$$u_i^\mu = \sum_{j=1}^{M} V_{ij} y_j^\mu \tag{3}$$

$$y_j^\mu = \sigma(s_j^\mu) \tag{4}$$

$$s_j^\mu = \sum_{k=1}^{D} W_{jk} x_k^\mu \tag{5}$$

Now, we want to calculate each of the partial derivatives $\frac{\partial\varepsilon}{\partial V_{ij}}$ and $\frac{\partial\varepsilon}{\partial W_{jk}}$. If we stack the components into a vector, we have found our error gradient.

Don't let all the indices scare you! Look at the graph and concentrate on how the weight connects to other nodes. Also observe how the output error propagates through the network from the output to the inputs.

The chain rule gives us for V (the weights in the output layer):

$$\frac{\partial\varepsilon}{\partial V_{ij}} = \frac{1}{N}\sum_{\mu=1}^{N}\sum_{n=1}^{C}\frac{\partial\varepsilon_n^\mu}{\partial u_n^\mu}\frac{\partial u_n^\mu}{\partial V_{nj}} = \frac{2}{N}\sum_{\mu=1}^{N}\left(d_i^\mu - u_i^\mu\right)(-1)y_j^\mu \tag{6}$$

$$= \frac{2}{N}\sum_{\mu=1}^{N}\left(u_i^\mu - d_i^\mu\right)y_j^\mu = \frac{2}{N}(\mathbf{u}_i - \mathbf{d}_i)\mathbf{y}_j^T \tag{7}$$

Note that the sum over the number of output neurons has changed index name from i to n, so that we don't confuse it with the index in the differentiation variable V_{ij}. In addition, at the second equals sign this sum vanishes because the only term contributing is when $n = i$. If you look at the figure you can see this as it is clear that only the error at the output node $n = i$ is dependent on the weight V_{ij}. The last step is a conversion to vector form. Note that $\mathbf{u}_i$ and $\mathbf{d}_i$ here is *row*-vectors(horizontal) with N components. The output of the expression is therefore simply a scalar. In order to make this efficient, we would like to calculate the update of the whole V at the same time. We can rewrite the expression above in that way, even if it can be a bit cumbersome. Let us rewrite the expression in two steps. First we eliminate the j-index (j denotes a certain neuron in the hidden layer, remember?):

$$\frac{\partial\varepsilon}{\partial\mathbf{v}_i} = \frac{2}{N}(\mathbf{u}_i - \mathbf{d}_i)Y^T \tag{8}$$

Y is a matrix with output signals from each of the neurons in the hidden layer. It has the dimension of $M \times N$, that is M rows and N columns. What remains is to eliminate the i-index (i denotes a certain neuron in the output layer):

$$\nabla_V = \frac{\partial\varepsilon}{\partial\mathbf{V}} = \frac{2}{N}(U - D)Y^T \tag{9}$$

U and D are matrices of the dimension $C \times N$ and hold the output value and the desired (training output) value respectively for all samples and all output neurons (classes). The dimension of V becomes $C \times M$.

For W we proceed in the same way. Look at the figure and follow the errors back through the net to

the weight W_{jk}.

$$\frac{\partial \varepsilon}{\partial W_{jk}} = \frac{1}{N}\sum_{\mu=1}^{N}\sum_{i=1}^{C}\frac{\partial \varepsilon_i^\mu}{\partial u_i^\mu}\frac{\partial u_i^\mu}{\partial y_j^\mu}\frac{\partial y_j^\mu}{\partial s_j^\mu}\frac{\partial s_j^\mu}{\partial W_{jk}^\mu} \tag{10}$$

$$= \frac{1}{N}\sum_{\mu=1}^{N}\sum_{i=1}^{C} 2(d_i^\mu - u_i^\mu)(-1)\left(V_{ij}\sigma'(s_j^\mu)x_k^\mu\right) \tag{11}$$

$$= \frac{2}{N}\sum_{\mu=1}^{N} x_k^\mu \sum_{i=1}^{C}(u_i^\mu - d_i^\mu)V_{ij}\sigma'(s_j^\mu) \tag{12}$$

What remains is just(?) to rewrite this into matrix form. A somewhat cumbersome process leads to

$$\nabla_W = \frac{2}{N}((V^T(U-D)).*\sigma'(S))X^T \tag{13}$$

where S has the same form as Y ($M \times N$) and W has the form $M \times D$. Finally, we have to deal with the bias weights.... In order not to force a number of planes to involuntary go through the origin, we have to add a bias column to W and V. In addition we have to add ones to X and Y (where?). This part you have to find out on your own when you do the lab.

b) The difference between online and offline mode is that in offline mode we have access to all samples before we start to train. In online mode we get access to one sample at a time. This means that we get a training sample, run it through the network, calculate the error gradient and update the weights after each arriving sample.

The error measure is the same as before but without the sum over the training samples. (Or you can use the same measure as before and set N=1):

$$\varepsilon = \sum_{i=1}^{C}(d_i - u_i)^2 \tag{14}$$

Then you just have to differentiate as usual. Note that when we train online it is a good idea to let the learning rate decrease with the number of iterations(epochs). Otherwise the last samples will have disproportional influence over the properties of the planes. You also have to make sure that training samples from different classes arrive in a balanced fashion. If all samples from one class arrive first and then all from another, we might get very bad solutions.

There is no point in iterating an online solution. When all samples have arrived we might just as well use the offline method and avoid all pitfalls.

Conclusion: Avoid online mode if you can. Even if you have to use online mode then you might consider alternatives; one example is to use offline mode training on all the samples that have arrived so far.

3.9. Answer

Begin by sketching the problem and a possible decision boundary, for example as in Figure **(a)**. Since the classification must switch value two times it requires at least two nodes in the hidden layer, as well as a bias. The network will therefore look like the sketch below. From this information we can write the complete function that the neural network will implement:

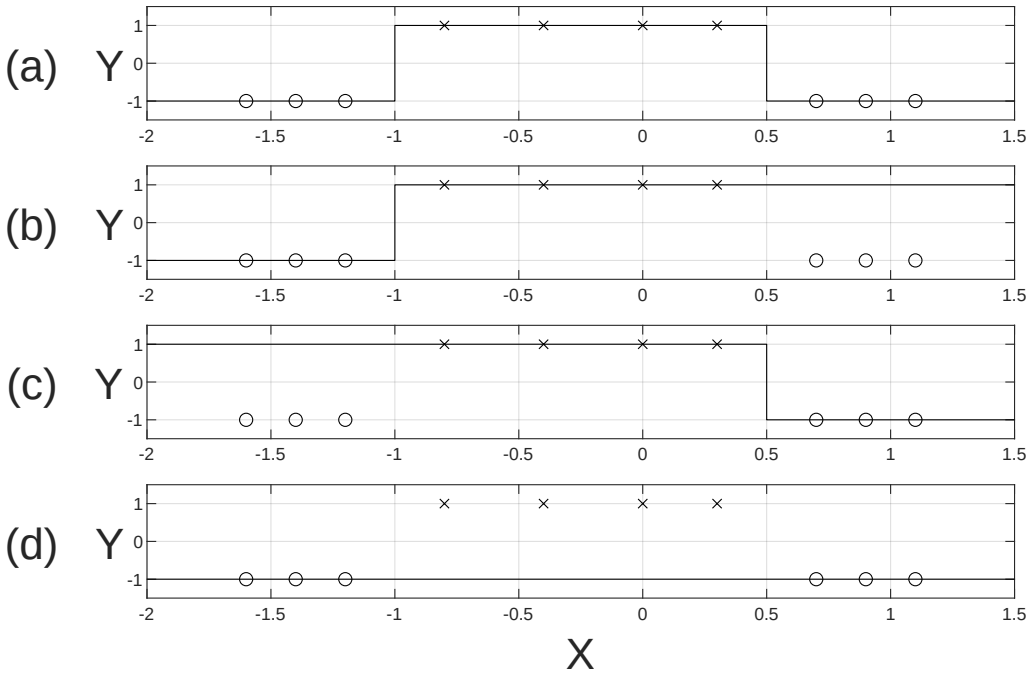
$$Y = sign\left(v_{1,1} + v_{1,2} * sign\left(w_{1,2} * \mathbf{X} + w_{1,1}\right) + v_{1,3} * sign\left(w_{2,2} * \mathbf{X} + w_{2,1}\right)\right)$$

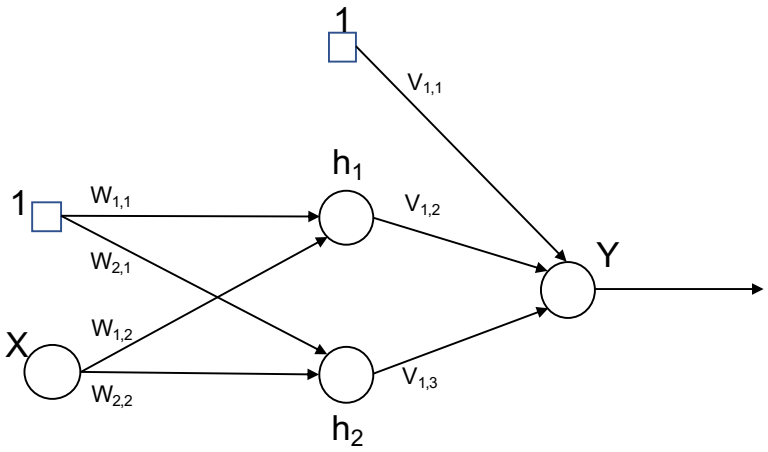
where we use notation corresponding to the weight matrices **W** for the first layer and **V** for the second layer. The two inner $sign$ functions are the two hidden nodes. The only thing that remains is to find the values of the weights.
We can decompose the decision boundary in **(a)** into the sum of the three boundaries in **(b)**, **(c)**, and **(d)**, corresponding to the two hidden nodes and a bias, respectively. Since $sign(\mathbf{X} + \alpha)$ switches from -1 to 1 at $-\alpha$, we can select the bias weights for the first layer: $w_{1,1} = 1$, $w_{2,1} = -0.5$. The weights for **X** can

both be 1, i.e., $w_{1,2} = w_{2,2} = 1$. We can see in **(c)** that the second $sign$ function is inverted, so we set $v_{1,3} = -1$. The first $sign$ function is not inverted, so we set $v_{1,2} = 1$. Finally, the sum of these two sign function will be 2 in the range -1 $\leq \mathbf{X} < 0.5$, and 0 outside this range. The bias $v_{1,1}$ can therefore take any value in the range $(0, -2)$, excluding both 0 and -2. In summary, one solution is the weight matrices

$$\mathbf{W} = \begin{bmatrix} 1 & 1 \\ -0.5 & 1 \end{bmatrix}, \quad \mathbf{V} = \begin{bmatrix} -1 & 1 & -1 \end{bmatrix} \tag{15}$$

which gives **Y** exactly equal to **D**.

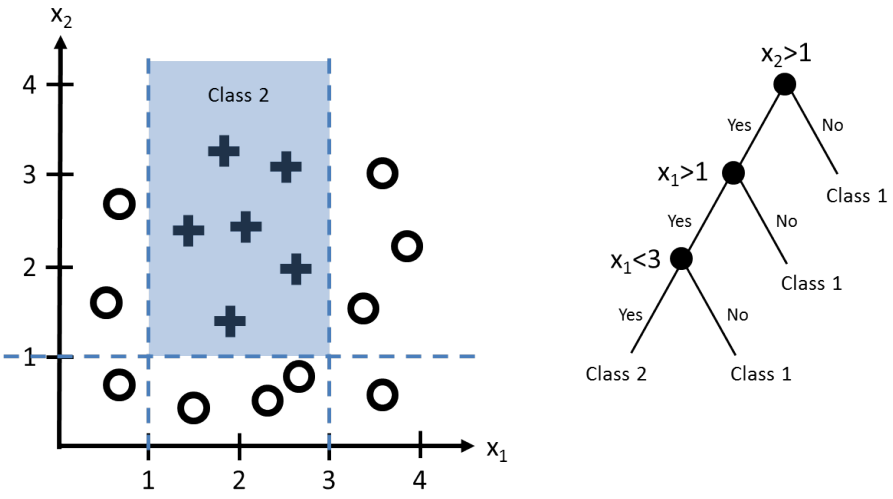




5. Ensemble learning and boosting

Answer 5.1

- One of the many models are selected, and the choice of model is a function of the input variables. In the *decision tree*, a sequence of binary decisions (ordered as a binary tree) yields the chosen model. The models are in the "leafs" of the tree and are often very simple, e.g. "+1".
- The leafs of the regression tree hold real values, not class labels.
- The figure below shows one possible solution. Note that there are other similar solutions.



Answer 5.2

- A decision stump is a decision tree with a single node. It partitions the input space into two regions. The linear decision surface is perpendicular to one of the feature axes.
- The polarity and the threshold.
- $\min_{\tau,p} \epsilon(\tau, p) = \sum_{i=1}^{M} d_i I(y_i \neq h(x_i; \tau, p))$
- If the performance ϵ of the classifier is not better than 0.5, we can always change the polarity (flip the signs) so that we get $(1 - \epsilon)$ in performance.
- We only need to test thresholds in relation to the training data: M thresholds.

Answer 5.3

Earlier in the course we have tried to make *one* classifier/regressor to perform as good as possible. Ensemble learning and boosting, in contrast, combine *multiple* models to achieve this goal.

Answer 5.4

a) *Bagging*, short for bootstrap aggregation, is a committee method where the individual models are trained on separate *bootstrap* datasets that are created from the original dataset using random sampling with replacement.

b) *Boosting* is also a committee method but the base models are trained in sequence, and each base model is trained using a weighted form of the data set in which the weighting coefficient associated with each data sample depends on the performance of the previous classifiers.

Answer 5.5

- A weak classifier is a simple classifier that may perform only slightly better than a random classifier. That is, for two classes, a weak classifier may give a classification accuracy of only slightly above 50%. The opposite of a weak classifier is a strong classifier, that aims to have a very high classification accuracy.
- Small decision trees or even decision stumps are typical weak classifiers.
- The XOR classification problem, see figure 5.
- See Figure 5.

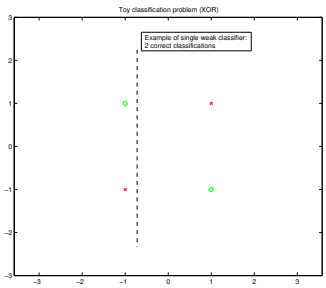


Figure 5: Toy example

Answer 5.6

See the lecture.

a) `Set weights` $d_1 := 1/N; i := 1;$

b) `Train weak classifier` $h_i(x)$ `using weights` d_i

c) `Increase and decrease weight for wrongly and correctly classified training examples respectively ->` d_{i+1}

d) $i := i + 1;$

e) `Repeat from b) until` $i > M$

Answer 5.7

a) In figure 6 the classification problem is sketched along with the initial weights. All vertical/horizontal decision stumps within the square of the data samples give the error $\epsilon = \frac{1}{4}$. In figure 6 we have chosen one of these lines as an solution example. We get $\alpha_1 = \frac{1}{2}\ln\frac{1-\epsilon_1}{\epsilon_1} = \frac{1}{2}\ln\frac{3/4}{1/4} = \frac{\ln 3}{2}$. Now we update (decreasing) the weights of the correctly classified samples with $e^{-\alpha_1} = \frac{1}{\sqrt{3}}$. The weight of the erroneous classified sample (upper right in the solution example) is instead increased with $e^{\alpha_1} = \sqrt{3}$. The sum of the weights are now $3\frac{1}{4\sqrt{3}} + \frac{\sqrt{3}}{4} = \frac{\sqrt{3}}{2}$. After normalizing the weights (dividing with the sum) we get the resulting weights as shown in the right part of figure 6. As we can see, AdaBoost works well here.

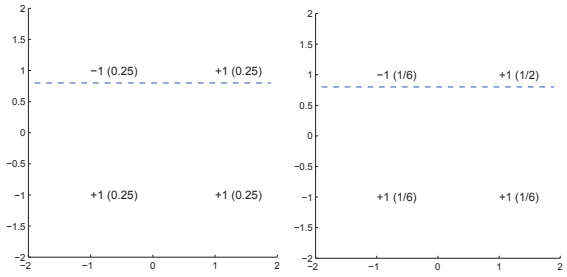


Figure 6:

b) In figure 7 the classification problem is sketched along with the initial weights. All vertical/horizontal decision stumps within the square of the data samples give the error $\epsilon = \frac{1}{2}$. In figure 7 we have chosen one of these lines as an solution example. We get $\alpha_1 = \frac{1}{2}\ln\frac{1-\epsilon_1}{\epsilon_1} = \frac{1}{2}\ln\frac{1/2}{1/2} = \frac{\ln 1}{2} = 0$. If we try to update the weights, we get $e^{\pm\alpha_1} = 1$, i.e. the weights don't change. This means that the weak classifier in iteration two will face the same problem as in iteration 1. As we can see, AdaBoost does not work in this setting. One solution is to allow other types of weak classifiers.

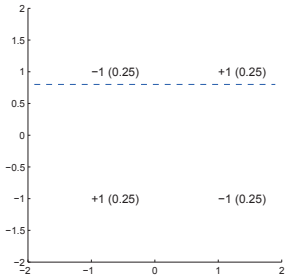


Figure 7:

c) Outliers gain weight exponentially which will eventually result in bad weak classifiers. This may ruin the composite strong classifier. Solutions to this problem include various forms of weight trimming and alternative weight update approaches.

7. Reinforcement learning

Answer 7.1

In supervised learning, the feedback is the correct action or class for every input, situation or state. In reinforcement learning, a scalar feedback (reward or punishment) is obtained for taking certain actions or reaching certain states, but the correct or optimal action is not given to the learning system.

Answer 7.2

The "temporal credit assignment problem" refers to the problem of assessing individual actions in a sequence of actions when the reward/punishment comes delayed. For example, which were the winning moves that led to the victory in a game?

Answer 7.3

The V-function describes the value (expected reward) that will be obtained in the future when being in certain state, when following a certain policy (which action to take in every state). Each policy has a different value function. The Q-function describes the value for each action in each state, given that the optimal policy is followed after the action has been taken.

Answer 7.4

The discount factor controls the trade-off between optimizing for immediate rewards and long term rewards.

Answer 7.5

ϵ-greedy exploration means that a random action (exploration) is made with probability ϵ and a greedy action (exploitation), i.e., an action that maximizes the reward, is made with probability $1 - \epsilon$.

Answer 7.6

The definition of the optimal Q-function is given by:

$$\begin{aligned} Q^*(x,a) &= r(x,a) + \gamma V^*(g(x,\mu^*(x))) \\ &= r(x,a) + \gamma \max_b Q^*(g(x,\mu^*(x)),b) \end{aligned}$$

We can find a solution by traversing the state graph recursively starting from the end in the first exercise, since we know the value of $V^*(4)$.

a) According to the state graph, $V^*(4) = 0$.

$$\begin{aligned} Q^*(3,up) &= 1 + \gamma V^*(4) = 1 = V^*(3) \\ Q^*(2,up) &= 0 + \gamma V^*(3) = \gamma = V^*(2) \\ Q^*(1,up) &= 0 + \gamma V^*(2) = \gamma^2 = V^*(1) \end{aligned}$$

b) In this exercise there is no end. The system will however always end up in state 3 and stay there. For $\gamma \neq 1$ we therefore have

$$\begin{aligned} Q^*(3,same) &= 1 + \gamma Q^*(3,same) \Leftrightarrow \\ Q^*(3,same) &= 1/(1-\gamma) = V^*(3) \\ Q^*(2,up) &= 0 + \gamma V^*(3) = \gamma/(1-\gamma) = V^*(2) \\ Q^*(1,up) &= 0 + \gamma V^*(2) = \gamma^2/(1-\gamma) = V^*(1) \end{aligned}$$

We can consequently conclude that we have to use a discounting factor $\gamma \neq 1$ in order to avoid infinite Q- and V-functions.

c) According to the state graph, $V^*(4) = 0$ and $V^*(5) = 0$. We can therefore calculate (in order)

$$\begin{aligned}
Q^*(3, up) &= 2 + \gamma V^*(4) = 2 = V^*(3) \\
Q^*(2, up) &= 0 + \gamma V^*(3) = 2\gamma \\
Q^*(2, right) &= 1 + \gamma V^*(5) = 1 \\
V^*(2) = \max_a Q^*(2, a) &= \begin{cases} 2\gamma & \text{om } \gamma > \frac{1}{2} \\ 1 & \text{om } \gamma \leq \frac{1}{2} \end{cases} \\
Q^*(1, up) = 0 + \gamma V * (2) &= \begin{cases} 2\gamma^2 & \text{om } \gamma > \frac{1}{2} \\ \gamma & \text{om } \gamma \leq \frac{1}{2} \end{cases} = V^*(1)
\end{aligned}$$

For this exercise we can conclude that we get different optimal policies depending on the value of γ. A low γ value corresponds to upweighted rewards in the short term and the system therefore turns right immediately in spite of this causing a lower unweighted reward sum over time for this policy.

Answer 7.7

The definition of the optimal Q-function is given by:

$$\begin{aligned}
Q^*(x, a) &= r(x, a) + \gamma V^*(g(x, \mu^*(x))) \\
&= r(x, a) + \gamma \max_b Q^*(g(x, \mu^*(x)), b)
\end{aligned}$$

a) For system A, there is only one choice of policy, to continue forward. For system B, we can see that the optimal policy is to get to the upper loop as soon as possible, and then continue around the loop, since this will maximize the rewards over time.

b) Since there is only one action in every state, the Q- and V-functions will be equal. By application of the definition of the optimal Q-function, we get the following:

$$\begin{aligned}
V^*(1) &= Q^*(1, forward) = 3 + \gamma V^*(2) \\
V^*(2) &= Q^*(2, forward) = 0 + \gamma V^*(3) \\
V^*(3) &= Q^*(3, forward) = 0 + \gamma V^*(4) \\
V^*(4) &= Q^*(4, forward) = 0 + \gamma V^*(5) \\
V^*(5) &= Q^*(5, forward) = 0 + \gamma V^*(6) \\
V^*(6) &= Q^*(6, forward) = 0 + \gamma V^*(1)
\end{aligned}$$

Which gives

$$\begin{aligned}
V^*(1) &= Q^*(1, forward) = \frac{3}{1 - \gamma^6} \\
V^*(2) &= Q^*(2, forward) = \frac{3\gamma^5}{1 - \gamma^6} \\
V^*(3) &= Q^*(3, forward) = \frac{3\gamma^4}{1 - \gamma^6} \\
V^*(4) &= Q^*(4, forward) = \frac{3\gamma^3}{1 - \gamma^6} \\
V^*(5) &= Q^*(5, forward) = \frac{3\gamma^2}{1 - \gamma^6} \\
V^*(6) &= Q^*(6, forward) = \frac{3\gamma^1}{1 - \gamma^6}
\end{aligned}$$

c) From exercise A we know that the optimal policy when standing in some of the states in the upper loop, i.e. 1, 2, 3 or 6, is to go around the loop. We therefore start calculating the V- and Q-functions

in this loop.

$$V^*(1) = Q^*(1, forward) = 3 + \gamma V^*(2)$$
$$V^*(2) = Q^*(2, forward) = 0 + \gamma V^*(3)$$
$$V^*(3) = Q^*(3, shortcut) = 0 + \gamma V^*(6)$$
$$V^*(6) = Q^*(6, forward) = 0 + \gamma V^*(1)$$

Which in the same way as in B gives

$$V^*(1) = Q^*(1, forward) = \frac{3}{1-\gamma^4}$$
$$V^*(2) = Q^*(2, forward) = \frac{3\gamma^3}{1-\gamma^4}$$
$$V^*(3) = Q^*(3, shortcut) = \frac{3\gamma^2}{1-\gamma^4}$$
$$V^*(6) = Q^*(6, forward) = \frac{3\gamma^1}{1-\gamma^4}$$

Now when we know the value of the V-function for state 6, we can calculate the V- and Q-functions for the state 5,4 and 3 recursively.

$$V^*(5) = Q^*(5, forward) = 0 + \gamma V^*(6) = \frac{3\gamma^2}{1-\gamma^4}$$
$$V^*(4) = Q^*(4, forward) = 0 + \gamma V^*(5) = \frac{3\gamma^3}{1-\gamma^4}$$
$$Q^*(3, forward) = 0 + \gamma V^*(4) = \frac{3\gamma^4}{1-\gamma^4}$$

Answer 7.8

Since we are dealing with stochastic reward functions we are forced to include a expectation value in our expression for optimal Q-function. (In the lab we solved this by choosing a low value for the learning rate η.)
The definition of the optimal Q-function is given by:

$$\begin{aligned} Q^*(x,a) &= E\{r(x,a)\} + \gamma V^*(g(x, \mu^*(x))) \\ &= E\{r(x,a)\} + \gamma \max_b Q^*(g(x, \mu^*(x)), b) \end{aligned}$$

We can find a solution by traversing the state graph recursively starting from the end in the first exercise, since we know the value of $V^*(4)$.

a) According to the state graph, $V^*(4) = 0$.

$$Q^*(3, up) = 3 + \gamma V^*(4) = 3 = V^*(3)$$
$$Q^*(2, up) = (3 \cdot 0.70 + 1 \cdot 0.30) + \gamma V^*(3) = 2.4 + 3\gamma = V^*(2)$$
$$Q^*(1, up) = 0 + \gamma V^*(2) = 2.4\gamma + 3\gamma^2 = V^*(1)$$

b) According to the state graph, $V^*(3) = 0$ and $V^*(5) = 0$. We can therefore calculate (in order)

$$Q^*(4, up) = 3 + \gamma V^*(5) = 3 = V^*(4)$$
$$Q^*(2, up) = (3 \cdot 0.90 - 25 \cdot 0.10) + \gamma V^*(3) = 0.2$$
$$Q^*(2, left) = 0 + \gamma V^*(4) = 3\gamma$$
$$V^*(2) = \max_a Q^*(2, a) = \begin{cases} 0.2 & \text{om } \gamma \leq \frac{2}{30} \\ 3\gamma & \text{om } \gamma > \frac{2}{30} \end{cases}$$
$$Q^*(1, up) = 0 + \gamma V*(2) = \begin{cases} 0.2\gamma & \text{om } \gamma \leq \frac{2}{30} \\ 3\gamma^2 & \text{om } \gamma > \frac{2}{30} \end{cases} = V^*(1)$$

For this exercise we can conclude that we get different optimal policies depending on the value of γ. A low γ value corresponds to upweighted rewards in the short term and the system therefore goes straight up in spite of this causing a lower unweighted reward sum over time for this policy than if it had turned left in the crossing.

Answer 7.9

The formula for the optimal Q-function is given by:

$$\begin{aligned} Q^*(x,a) &= r(x,a) + \gamma V^*(g(x,\mu^*(x))) \\ &= r(x,a) + \gamma \max_b Q^*(g(x,\mu^*(x)),b) \end{aligned}$$

a) Counting backwards we get:

$$\begin{aligned} Q^*(3,right) &= 150 + 0 \cdot \gamma = 150 \\ Q^*(2,down) &= \gamma Q^*(3,right) - 50 = 150\gamma - 50 \\ Q^*(1,up) &= \gamma Q^*(2,down) - 50 = 150\gamma^2 - 50\gamma - 50 \\ Q^*(1,right) &= p \cdot \gamma Q^*(3,right) + 0 - 150 \cdot (1-p) \end{aligned}$$

b) Using the result from A we get:

$$\begin{aligned} V^*(5) &= 0 \\ V^*(4) &= 0 \\ V^*(3) &= Q^*(3,right) = 150 + 0 = 150 \\ V^*(2) &= Q^*(2,down) = Q^*(3,right) - 50 = 150 - 50 = 100 \\ p \leq \frac{2}{3} &\Rightarrow V^*(1) = Q^*(1,up) = Q^*(2,down) - 50 = 150 - 50 - 50 = 50 \\ p \geq \frac{2}{3} &\Rightarrow V^*(1) = Q^*(1,right) = p \cdot Q^*(3,right) + 0 - 150 \cdot (1-p) = (2p-1) \cdot 150 \end{aligned}$$

c) If $0 < \alpha < 1$ the system will eventually converge to the average reward for the action "right" from "1", giving us an optimal policy. But if $\alpha = 1$ only the reward of last action taken will be remembered, and the policy might never converge to a stable state as long as we are exploring. For example when $2/3 < p < 1$ it is optimal to move right from node 1, since this is the path where we can expect the maximum reward over time, but sometimes there will be a huge negative reward. If we were to use $\alpha = 1$ in this example the policy will shift every time the we try to move right from "1" and end up where we did not expect, and thus the final policy will only be dependent on when we stop training and not on which path is optimal.

8. Unsupervised learning and dimensionality reduction

Answer 8.1

Hard clustering means that each training sample is assigned to only one cluster, whereas soft/fuzzy clustering means that a training sample can belong to several clusters to certain degrees.

Answer 8.2

a) In k-means clustering, only the centers of the clusters are modeled, whereas in Mixture of Gaussian clustering, also the cluster shape is modeled using the covariance matrix, which can model circular and elliptical cluster shapes.

b) Expectation maximization, commonly known as the EM-algorithm.

Answer 8.3

k-NN: k is the number of the stored data vectors that vote for the classification decision.
k-means: k is the number of clusters.

Answer 8.4

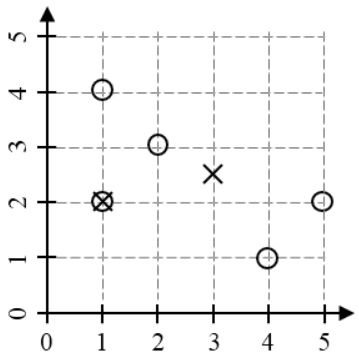
The prototype vectors are updated to the means of the closest data points, i.e.,

$$\mathbf{p}_1 = \frac{1}{4}\left[\begin{pmatrix}1\\4\end{pmatrix} + \begin{pmatrix}2\\3\end{pmatrix} + \begin{pmatrix}4\\1\end{pmatrix} + \begin{pmatrix}5\\2\end{pmatrix}\right] = \begin{pmatrix}3\\2.5\end{pmatrix}$$

and

$$\mathbf{p}_2 = \begin{pmatrix}1\\2\end{pmatrix}$$

as shown in the figure.



The algorithm must perform 2 more iteration before no more points change clusters.

Answer 8.5

a) N-dim. normal (Gaussian) distribution:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{N/2}\sqrt{\det \mathbf{C}}} \exp\left[-\frac{1}{2}(\mathbf{x}-\mathbf{m})^T\mathbf{C}^{-1}(\mathbf{x}-\mathbf{m})\right]$$

b) Remember that $\mathbf{A^{-1}} = \begin{bmatrix}a & b\\ c & d\end{bmatrix}^{-1} = \frac{1}{\det(\mathbf{A})}\begin{bmatrix}d & -b\\ -c & a\end{bmatrix} = \frac{1}{ad-bc}\begin{bmatrix}d & -b\\ -c & a\end{bmatrix}$. Inserting the values in a) we get $p(x_1) = \frac{1}{2\pi\sqrt{8}} \approx 0.06, p(x_2) \approx 0.04, p(x_3) \approx 0.03$ respectively. x_2 is in the direction of the largest variance, i.e., the Gaussian has a wider support in that direction. x_3 on the other hand is in the direction of the smallest variance, i.e. the probability density decays faster in this direction.

Answer 8.6

Please see the Matlab demo code published on the course web site.

Answer 8.7

a) $m_X = E(X)$, $m_Y = E(Y)$ and $\mathrm{Cov}(x,y) = C(X,Y) = E((X - m_X)(Y - m_Y)) = E(XY) - E(X)E(Y)$

b) $m_x \approx \frac{1}{N}\sum_{i=1}^{N} x_i$ and $C(x,y) \approx \frac{1}{N}\sum_{i=1}^{N}(x_i - m_x)(y_i - m_y)$

c) $m_X = E(X)$, and $\mathrm{Var}(x) = \sigma^2 = E((X - m_X)^2) = E(X^2) - (E(X))^2$

d) $\mathrm{Var}(x) = \mathrm{Cov}(x,x)$

e) $m_x \approx \frac{1}{N}\sum_{i=1}^{N} x_i$ and $\mathrm{Var}(x) = C(x,x) \approx \frac{1}{N}\sum_{i=1}^{N}(x_i - m_x)^2$

f) $\mathrm{Corr}(x,y) = \rho(x,y) = \frac{C(x,y)}{\sqrt{\sigma_x^2\sigma_y^2}}$ where $-1 \leq \rho \leq 1$

Answer 8.8

a) $\mathbf{m_X} = E(\mathbf{X})$, and Cov(**x**)= $\mathbf{C}(\mathbf{X}) = E((\mathbf{X} - \mathbf{m_X})(\mathbf{X} - \mathbf{m_X})^T)$

b) $\mathbf{m_x} \approx \frac{1}{N}\sum_{i=1}^{N} \mathbf{x_i}$ and $\mathbf{C}(\mathbf{x}) \approx \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x_i} - \mathbf{m_x})(\mathbf{x_i} - \mathbf{m_x})^T$

c) $\mathbf{m_X} = E(\mathbf{X})$, $\mathbf{m_Y} = E(\mathbf{Y})$, and Cov(**x**, **y**)= $\mathbf{C}(\mathbf{X},\mathbf{Y}) = E((\mathbf{X} - \mathbf{m_X})(\mathbf{Y} - \mathbf{m_Y})^T)$

Answer 8.9

a) $\mathbf{C} = \begin{bmatrix} \mathrm{Var}(x_1) = \mathrm{Cov}(x_1,x_1) & \mathrm{Cov}(x_1,x_2) & \mathrm{Cov}(x_1,x_3) \\ \mathrm{Cov}(x_2,x_1) & \mathrm{Var}(x_2) = \mathrm{Cov}(x_2,x_2) & \mathrm{Cov}(x_2,x_3) \\ \mathrm{Cov}(x_3,x_1) & \mathrm{Cov}(x_3,x_2) & \mathrm{Var}(x_3) = \mathrm{Cov}(x_3,x_3) \end{bmatrix}$.
Since $\mathrm{Cov}(x_i,x_j) = \mathrm{Cov}(x_j,x_i)$, $\mathbf{C}$ is symmetric.

b) $\mathbf{C}_{\text{corr}} = \begin{bmatrix} 1 & \mathrm{Corr}(x_1,x_2) & \mathrm{Corr}(x_1,x_3) \\ \mathrm{Corr}(x_2,x_1) & 1 & \mathrm{Corr}(x_2,x_3) \\ \mathrm{Corr}(x_3,x_1) & \mathrm{Corr}(x_3,x_2) & 1 \end{bmatrix}$
Since $\mathrm{Corr}(x_i,x_j) = \mathrm{Corr}(x_j,x_i)$, $\mathbf{C}_{\text{corr}}$ also symmetric.

c) The covariance matrix will be scaled with a factor $5^2 = 25$. The correlation matrix is invariant to signal scaling.

d) A diagonal covariance matrix imply uncorrelated components in $\mathbf{z}(t)$.

Answer 8.10

$$m_{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i \quad \text{and} \quad C(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - m_{\mathbf{x}})(\mathbf{x}_i - m_{\mathbf{x}})^T$$

According to the above, we then get $m_{\mathbf{x}} = \begin{bmatrix}1\\2\end{bmatrix}$ and $C(\mathbf{x}) = \begin{bmatrix}2 & -1\\ -1 & \frac{2}{3}\end{bmatrix}$. In practice you need a lot more than three samples in order to get a good estimate of the mean value and the covariance.

Answer 8.11

Let the feature vector be defined by

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

The correlation matrix tells us that the correlation between x_1 and x_3 is -1, which means that there is an inverse linear relationship $x_1 = -k\,x_3 + m$ with some $k > 0$ and some arbitrary difference in mean value m. The correlation between x_1 and x_2 is 0, as is the correlation between x_3 and x_2, i.e., there is no linear

relationship between x_2 and the other features. This information can be used for dimensionality reduction, for example to make a new reduced feature vector

$$\tilde{\mathbf{x}} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix},$$

because x_1 and x_3 carry the same information (up to a scaling and mean value, which usually is not interesting).

Answer 8.12

a) In order to find the eigenvectors $\mathbf{e}$ to the matrix $\mathbf{C}$ we have to solve $\mathbf{Ce} = \lambda\mathbf{e}$. First, find the eigenvalues using the characteristic (secular) equation: $\det(\mathbf{C} - \lambda\mathbf{I}) = 0$

We then get the eigenvalues $\lambda_1 = 5$, $\lambda_2 = 1$. Insert these values and solve for the corresponding eigenvectors. We get $\mathbf{e}_1 = \frac{1}{2}\begin{bmatrix} \sqrt{3} \\ 1 \end{bmatrix}$, $\mathbf{e}_2 = \frac{1}{2}\begin{bmatrix} 1 \\ -\sqrt{3} \end{bmatrix}$

b) $\mathbf{e}_1$ and $\mathbf{e}_2$ are orthogonal, i.e., $\mathbf{e}_1^T\mathbf{e}_2 = 0$. For symmetric matrices, like the covariance matrix, the eigenvectors are always orthogonal as proven by the so-called Spectral Theorem in linear algebra.

c) λ_1 is the variance of the projected data onto $\mathbf{e}_1$, i.e., $\lambda_1 = \mathrm{Var}(\mathbf{x}^T\mathbf{e}_1)$, where $\mathbf{x}$ is the 2-dimensional data vectors whose covariance matrix is $\mathbf{C}$.

Answer 8.13

a)

$$V = \mathrm{Var}[\hat{\mathbf{w}}^T\mathbf{x}] = E[(\hat{\mathbf{w}}^T\mathbf{x})^T(\hat{\mathbf{w}}^T\mathbf{x})] = E[\hat{\mathbf{w}}^T\mathbf{x}\mathbf{x}^T\hat{\mathbf{w}}] = \hat{\mathbf{w}}^T\mathbf{C}_{xx}\hat{\mathbf{w}} = \frac{\mathbf{w}^T\mathbf{C}_{xx}\mathbf{w}}{\mathbf{w}^T\mathbf{w}}$$

$$\frac{\partial V}{\partial \mathbf{w}} = \frac{2\mathbf{C}_{xx}\mathbf{w}\cdot\mathbf{w}^T\mathbf{w} - 2\mathbf{w}^T\mathbf{C}_{xx}\mathbf{w}\cdot\mathbf{w}}{(\mathbf{w}^T\mathbf{w})^2} = 0$$

$$\mathbf{C}_{xx}\mathbf{w}\cdot\mathbf{w}^T\mathbf{w} = \mathbf{w}^T\mathbf{C}_{xx}\mathbf{w}\cdot\mathbf{w}$$

$$\mathbf{C}_{xx}\mathbf{w} = \frac{\mathbf{w}^T\mathbf{C}_{xx}\mathbf{w}}{\mathbf{w}^T\mathbf{w}}\cdot\mathbf{w} = V\mathbf{w}$$

which is our eigenvalue problem.

b)

$$\begin{aligned} E[|\mathbf{x} - (\hat{\mathbf{w}}^T\mathbf{x})\hat{\mathbf{w}}|^2] &= E[(\mathbf{x}^T - \hat{\mathbf{w}}^T(\mathbf{x}^T\hat{\mathbf{w}}))(\mathbf{x} - (\hat{\mathbf{w}}^T\mathbf{x})\hat{\mathbf{w}})] \\ &= E[\mathbf{x}^T\mathbf{x}] - 2E[(\hat{\mathbf{w}}^T\mathbf{x})(\mathbf{x}^T\hat{\mathbf{w}})] + E[(\mathbf{x}^T\hat{\mathbf{w}})(\hat{\mathbf{w}}^T\mathbf{x})\hat{\mathbf{w}}^T\hat{\mathbf{w}}] \\ &= E[\mathbf{x}^T\mathbf{x}] - E[(\hat{\mathbf{w}}^T\mathbf{x})^T(\hat{\mathbf{w}}^T\mathbf{x})] = E[\mathbf{x}^T\mathbf{x}] - V \end{aligned}$$

having its minimum where V has its maximum.

Answer 8.14

This means that $\frac{2}{3}z_1 + \frac{1}{3}z_2 + \frac{2}{3}z_3$ is the linear combination of the data components that has the highest variance among all possible projection directions.

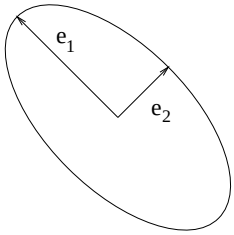
Answer 8.15

The principal directions are the same as the eigenvectors of the covariance matrix. These are

$$\hat{\mathbf{e}}_1 = \frac{1}{\sqrt{2}}\begin{bmatrix} -1 \\ 1 \end{bmatrix} \text{ with the eigenvalue } \lambda_1 = 4$$

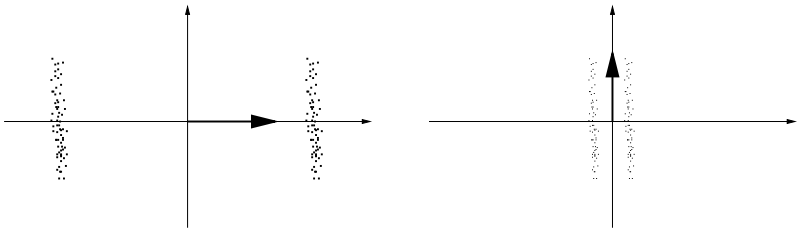
$$\hat{\mathbf{e}}_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ with the eigenvalue } \lambda_2 = 2$$

and the eigenvalues specifies the variance in respective direction. A sketch of the distribution could therefore look something like this:



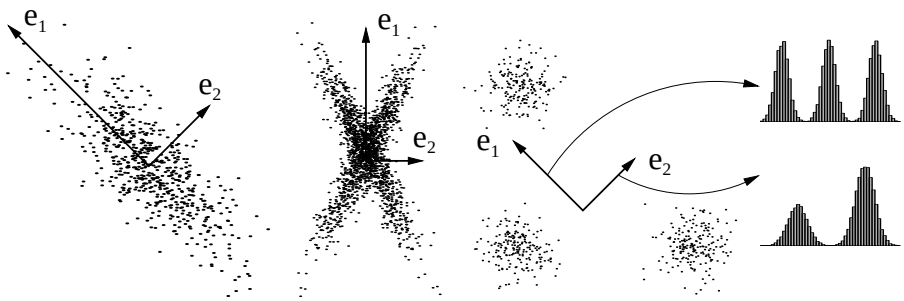
Answer 8.16

The principal directions are shown in the figure. Since the variance of the projection is maximized, PCA is dependent on the scaling of the variables. This means the method is an appropriate preprocessing step in the first case but not in the second.



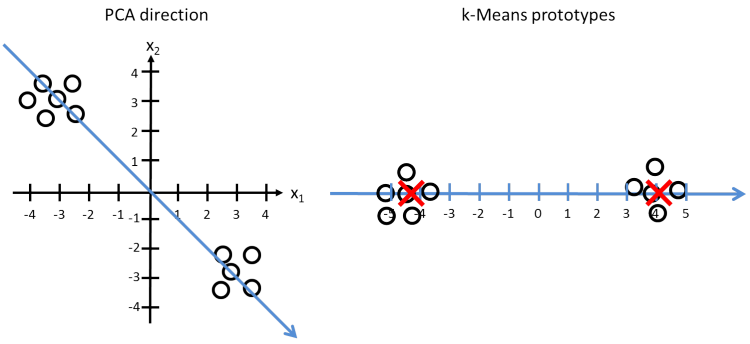
Answer 8.17

The principal directions are those with maximal and minimal variance respectively (in the 2D-case).



Answer 8.18

The left image below show the major PCA direction for dimensionality reduction. The right image shows the prototype vector positions at the center of the clusters (if we use $k = 2$ clusters). The original clusters are centered around (-3,3) and (3,-3). After projection on the PCA direction, the clusters are centered around $\sqrt{3^2 + 3^2} \approx 4.2$ and -4.2, which should be the output from the k-means algorithm.



Answer 8.19

The principal directions are orthogonal and if we normalize them they constitute an ON-base for the space. This means we can write $\mathbf{x}$ as a linear combination of the principal directions $\hat{\mathbf{e}}_1$, $\hat{\mathbf{e}}_2$ and $\hat{\mathbf{e}}_3$:

$$\mathbf{x} = \beta_1\hat{\mathbf{e}}_1 + \beta_2\hat{\mathbf{e}}_2 + \beta_3\hat{\mathbf{e}}_3$$

We can reduce the number of dimensions from 3 to 1 by only transmitting the coordinate β_1 for the first principal direction $\hat{\mathbf{e}}_1 = \frac{1}{\sqrt{3}}(-1,1,1)^T$. Due to the orthonormality we can calculate this coordinate as

$$\beta_1 = \mathbf{x}^T\hat{\mathbf{e}}_1$$

This dimension reduction is optimal in the sense of transmitting as much of the signal variance as possible. For the vector in the exercise we therefore transmit:

$$\beta_1 = \frac{1}{\sqrt{3}}\begin{pmatrix}1 & 2 & 0\end{pmatrix}\begin{pmatrix}-1\\1\\1\end{pmatrix} = \frac{1}{\sqrt{3}}$$

On the receiver side it can be reconstructed into $\frac{1}{\sqrt{3}}\hat{\mathbf{e}}_1 = \frac{1}{3}(-1,1,1)^T$.

Answer 8.20

Maximal and minimal variance are found in the principal directions. We start with calculating the covariance matrix

$$C = \begin{pmatrix}\mathrm{Var}(x) & \mathrm{Cov}(x,y)\\ \mathrm{Cov}(x,y) & \mathrm{Var}(y)\end{pmatrix} = \begin{pmatrix}s_x^2 & c_{x,y}\\ c_{y,x} & s_y^2\end{pmatrix}$$

where

$$\begin{aligned}
s_x^2 &= \frac{1}{80-1}\sum_{i=1}^{80}(x_i-\bar{x})^2 = \frac{1}{79}\sum_{i=1}^{80}\frac{1}{4} = \frac{20}{79}\\
s_y^2 &= \ldots = \frac{20}{79}\\
c_{x,y} &= \frac{1}{80-1}\sum_{i=1}^{80}(x_i-\bar{x})(y_i-\bar{y}) = \frac{1}{79}\left(20\frac{1}{4} - 60\frac{1}{4}\right) = -\frac{10}{79}\\
c_{y,x} &= c_{x,y}
\end{aligned}$$

so

$$C = \frac{10}{79}\begin{pmatrix}2 & -1\\ -1 & 2\end{pmatrix}$$

The principal directions (the eigenvectors) of the covariance matrix are

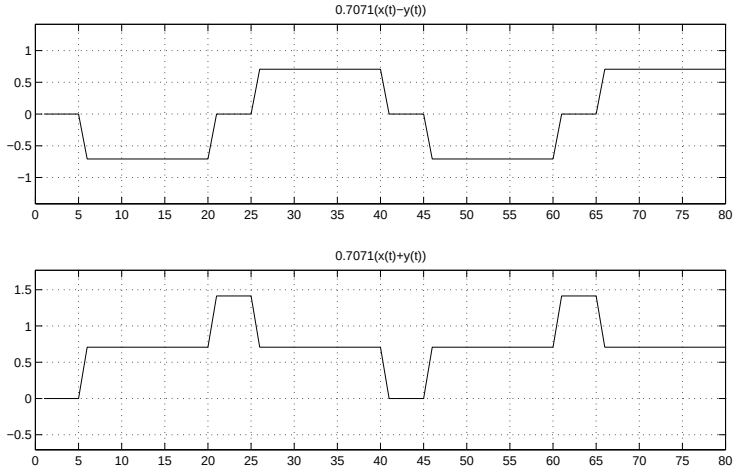
$$\hat{\mathbf{w}}_1 = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix} \text{ and } \hat{\mathbf{w}}_2 = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}$$

and the principal components become

$$\hat{s}_1(t) = \hat{\mathbf{w}}_1^T\begin{pmatrix}x(t)\\y(t)\end{pmatrix} = \frac{1}{\sqrt{2}}(x(t)-y(t))$$

and

$$\hat{s}_2(t) = \hat{\mathbf{w}}_2^T\begin{pmatrix}x(t)\\y(t)\end{pmatrix} = \frac{1}{\sqrt{2}}(x(t)+y(t))$$



Answer 8.21

The ordinary eigenvalue problem is

$$\mathbf{X}\mathbf{X}^T\hat{\mathbf{w}} = \lambda\hat{\mathbf{w}}$$

If we multiply with $\mathbf{X}^T$ from the left, we get

$$\mathbf{X}^T\mathbf{X}\mathbf{X}^T\hat{\mathbf{w}} = \lambda\mathbf{X}^T\hat{\mathbf{w}}$$

which can be written as

$$\mathbf{X^T}\mathbf{X}\hat{\mathbf{v}} = \lambda\hat{\mathbf{v}}$$

if we let $\hat{\mathbf{v}} = \mathbf{X}^T\hat{\mathbf{w}}$. Now we can solve the eigenvalue problem to obtain $\hat{\mathbf{v}}$. By using the relation above in the left step in the upmost equation, we get

$$\mathbf{X}\hat{\mathbf{v}} = \lambda\hat{\mathbf{w}}$$

Answer 8.22

a) See the lecture notes for details!

b) That both classes have the same covariance matrix, i.e., that the shapes of the distributions are identical.

Answer 8.23

Both vectors point in the X-direction, thus a classification can be made by thresholding the projection of the data onto that vector. While PCA maximizes the variance globally LDA maximizes the ratio of the between class and within class variances. This also means that LDA need some sort of preclassified training data.

Answer 8.24

The direction that optimality separates the two classes is given by

$$\mathbf{W} = \mathbf{C}_{tot}^{-1}\left(\mathbf{m}_1 - \mathbf{m}_2\right)$$

where $\mathbf{C}_{tot}$ is the sum of the individual covariance matrices for the two respective classes, and $\mathbf{m}_1$ and $\mathbf{m}_2$ are the centers of the two classes. We begin with the "crosses" class:

$$\mathbf{X}_1 = \begin{pmatrix} -1 & 1 & 2 & 3 & 5 \\ 2 & 2 & 4 & 3 & 4 \end{pmatrix}$$

$$\mathbf{C}_1 = \frac{1}{N-1}\left(\mathbf{X}_1 - \mathbf{m}_1\right)\left(\mathbf{X}_1 - \mathbf{m}_1\right)^T = \left[\mathbf{m}_1 = \begin{pmatrix} 2 \\ 3 \end{pmatrix}\right] = \frac{1}{4}\begin{pmatrix} 20 & 7 \\ 7 & 4 \end{pmatrix}$$

Now for the "circles" class:

$$\mathbf{X}_2 = \begin{pmatrix} 0 & 2 & 3 & 4 & 6 \\ -1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\mathbf{C}_2 = \frac{1}{N-1}\left(\mathbf{X}_2 - \mathbf{m}_2\right)\left(\mathbf{X}_2 - \mathbf{m}_2\right)^T = \left[\mathbf{m}_1 = \begin{pmatrix} 3 \\ 0 \end{pmatrix}\right] = \frac{1}{4}\begin{pmatrix} 20 & 4 \\ 4 & 2 \end{pmatrix}$$

We now get:

$$\mathbf{C}_{tot} = \mathbf{C}_1 + \mathbf{C}_2 = \frac{1}{4}\begin{pmatrix} 40 & 11 \\ 11 & 6 \end{pmatrix}$$

$$\mathbf{C}_{tot}^{-1} = \frac{4 * 4}{40 * 6 - 11 * 11} * \frac{1}{4}\begin{pmatrix} 6 & -11 \\ -11 & 40 \end{pmatrix} \approx \begin{pmatrix} 0.20 & -0.37 \\ -0.37 & 1.34 \end{pmatrix}$$

$$\mathbf{W} = \mathbf{C}_{tot}^{-1}\left(\mathbf{m}_1 - \mathbf{m}_2\right) \approx \begin{pmatrix} -1.31 \\ 4.40 \end{pmatrix}$$

Normalize $\mathbf{W}$:

$$\hat{\mathbf{W}} = \frac{\mathbf{W}}{\|\mathbf{W}\|_2} \approx \frac{1}{4.59}\begin{pmatrix} -1.31 \\ 4.40 \end{pmatrix} \approx \begin{pmatrix} -0.29 \\ 0.96 \end{pmatrix}$$

Project the data on $\hat{\mathbf{W}}$ which gives the following:

$$\mathbf{Y}_1 = \hat{\mathbf{W}}^T\mathbf{X}_1 \approx \begin{pmatrix} 2.20 & 1.63 & 3.26 & 2.02 & 2.41 \end{pmatrix}$$

$$\mathbf{Y}_2 = \hat{\mathbf{W}}^T\mathbf{X}_2 \approx \begin{pmatrix} -0.96 & -0.57 & -0.86 & -0.18 & -1.71 \end{pmatrix}$$

The projected data is shown in the figure below.

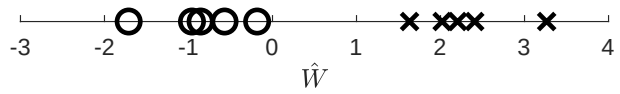


Figure 8: Data projected on $\hat{\mathbf{W}}$

Answer 8.25

a) PCA finds the directions of biggest variance in the data. If the features are on different scales (such as "Average number of children" compared to "Ability to read") the variance calculation will be skewed towards the larger features. Normalization removes this problem.

b) The covariance and correlation matrices are equal because we have normalized the data to have unit variance. The covariance matrix scales with the variance of the data, while the correlation matrix does not.

c) The most correlated features are "Average lifespan of men" and "Average lifespan of women" ($c = 0.98$). The most uncorrelated features are "Nativity/Mortality" and "Proportion of town residents" ($c = -0.0041$).

d) 73.91% of the information is kept by the first two principle components.

e) We need 5 principle components to keep more than 90% of the information.

f) Georgia.

g) The most important feature is "Nativity/Mortality". The least important is "Increase in population".

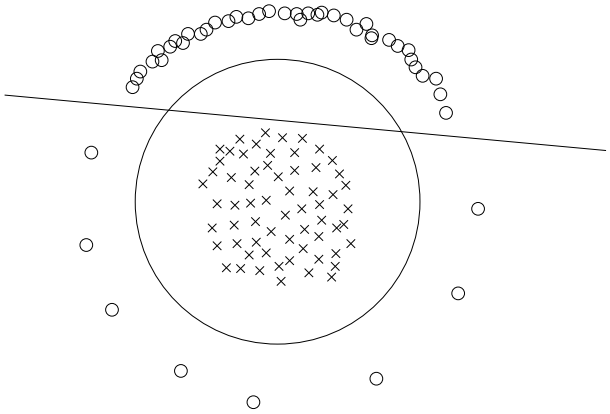
9. Kernel methods

Answer 9.1

Since the linear classifier produce a line the placement is pretty straightforward and some misclassifications will occur. With access to the squared feature values $\varphi_1 = x_1^2$ and $\varphi_2 = x_2^2$, and a bias $\varphi_3 = 1$, a linear classifier with a circular boundary in the original space is easily generated,

$$f(\varphi_1, \varphi_2, \varphi_3) = \text{sign}(w_1\varphi_1 + w_2\varphi_2 + w_3\varphi_3) = \text{sign}(w_1 x_1^2 + w_2 x_2^2 + w_3),$$

which provides perfect classification.



Answer 9.2

a) The scalar product is defined as $\mathbf{x}_1 \cdot \mathbf{x}_2 = \mathbf{x}_1^T \mathbf{x}_2$.

b) The length of the vector $\|\mathbf{x}\| = \sqrt{\|\mathbf{x}\|^2} = \sqrt{\mathbf{x}^T\mathbf{x}} = \sqrt{\mathbf{x} \cdot \mathbf{x}}$.

c) The distance $\|\mathbf{x}_1 - \mathbf{x}_2\| = \sqrt{(\mathbf{x}_1 - \mathbf{x}_2)^T(\mathbf{x}_1 - \mathbf{x}_2)} = \sqrt{\mathbf{x}_1^T\mathbf{x}_1 + \mathbf{x}_2^T\mathbf{x}_2 - 2\mathbf{x}_1^T\mathbf{x}_2} = \sqrt{\mathbf{x}_1 \cdot \mathbf{x}_1 + \mathbf{x}_2 \cdot \mathbf{x}_2 - 2\mathbf{x}_1 \cdot \mathbf{x}_2}$

d) The vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ span a triangle. The Law of Cosines from trigonometry states that if the lengths of the triangle sides are a, b and c, the following relation holds: $c^2 = a^2 + b^2 - 2ab\cos\theta$. If $\mathbf{x}_1$ represents a and $\mathbf{x}_2$ represents b, c is $\mathbf{x}_1 - \mathbf{x}_2$. The Law of Cosines then gives $\|\mathbf{x}_1 - \mathbf{x}_2\|^2 = \|\mathbf{x}_1\|^2 + \|\mathbf{x}_2\|^2 - 2\|\mathbf{x}_1\|\|\mathbf{x}_2\|\cos\theta$. Rearranging and using the results above gives the angle θ between $\mathbf{x}_1$ and $\mathbf{x}_2$ in terms of scalar products $\cos\theta = \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{\sqrt{\mathbf{x}_1 \cdot \mathbf{x}_1}\sqrt{\mathbf{x}_2 \cdot \mathbf{x}_2}}$.

Answer 9.3

The kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$ defines a scalar product between two vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ that have been mapped to a (usually high-dimensional) feature space via a function $\varphi()$.

$$\varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j) = \text{ some function.}$$

The clue here is that we never deal with $\varphi(\mathbf{x}_i)$ and $\varphi(\mathbf{x}_j)$ explicitly, i.e., we never calculate them or store them in the computer, we only deal the scalar products defined by $k(\mathbf{x}_i, \mathbf{x}_j)$. As the kernel function defines the scalar product, which in turn defines the distance between vectors according to 9.2, the kernel function can also be seen as a similarity function.

Answer 9.4

A kernel function defines the inner product $\kappa(\mathbf{x}_1, \mathbf{x}_2) = \Phi(\mathbf{x}_1)^T\Phi(\mathbf{x}_2)$ in the new feature space. Thus, $\kappa(\mathbf{x}_1, \mathbf{x}_2)$ specifies the feature space by defining how distances and angles are measured, instead of explicitly stating the mapping function $\Phi(\mathbf{x})$.

The distance between $\mathbf{x}_1$ and $\mathbf{x}_2$ in the new feature space is

$$\begin{aligned}\|\Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2)\| &= \sqrt{(\Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2))^T (\Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2))} \\ &= \sqrt{\Phi(\mathbf{x}_1)^T\Phi(\mathbf{x}_1) - 2\Phi(\mathbf{x}_1)^T\Phi(\mathbf{x}_2) + \Phi(\mathbf{x}_2)^T\Phi(\mathbf{x}_2)} \\ &= \sqrt{\kappa(\mathbf{x}_1, \mathbf{x}_1) - 2\,\kappa(\mathbf{x}_1, \mathbf{x}_2) + \kappa(\mathbf{x}_2, \mathbf{x}_2)} \\ &= \sqrt{1 - 2\,e^{-\frac{1}{4}} + 1} = 0.6651\end{aligned}$$

Answer 9.5

$$\varphi(\mathbf{x})^T\varphi(\mathbf{y}) = x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2 = (x_1y_1 + x_2y_2)^2 = \left(\mathbf{x}^T\mathbf{y}\right)^2$$

Answer 9.6

a) The kernel matrix $\mathbf{K}$ contains all scalar products between all training data feature vectors (generally mapped through a nonlinear function $\varphi()$). For example, if we have three training data samples $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$, the kernel matrix is

$$\mathbf{K} = \begin{pmatrix} \varphi(\mathbf{x}_1)^T\varphi(\mathbf{x}_1) & \varphi(\mathbf{x}_1)^T\varphi(\mathbf{x}_2) & \varphi(\mathbf{x}_1)^T\varphi(\mathbf{x}_3) \\ \varphi(\mathbf{x}_2)^T\varphi(\mathbf{x}_1) & \varphi(\mathbf{x}_2)^T\varphi(\mathbf{x}_2) & \varphi(\mathbf{x}_2)^T\varphi(\mathbf{x}_3) \\ \varphi(\mathbf{x}_3)^T\varphi(\mathbf{x}_1) & \varphi(\mathbf{x}_3)^T\varphi(\mathbf{x}_2) & \varphi(\mathbf{x}_3)^T\varphi(\mathbf{x}_3) \end{pmatrix} = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & k(\mathbf{x}_1, \mathbf{x}_3) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & k(\mathbf{x}_2, \mathbf{x}_3) \\ k(\mathbf{x}_3, \mathbf{x}_1) & k(\mathbf{x}_3, \mathbf{x}_2) & k(\mathbf{x}_3, \mathbf{x}_3) \end{pmatrix}.$$

One can note that $\mathbf{K}$ is symmetric as $\mathbf{x}_i^T\mathbf{x}_j = \mathbf{x}_j^T\mathbf{x}_i$.

b) 10×10.

Answer 9.7

We have the following kernel matrix

$$\mathbf{K} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 2 & 2 \\ 0 & 2 & 4 \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1^T\mathbf{x}_1 & \mathbf{x}_1^T\mathbf{x}_2 & \mathbf{x}_1^T\mathbf{x}_3 \\ \mathbf{x}_2^T\mathbf{x}_1 & \mathbf{x}_2^T\mathbf{x}_2 & \mathbf{x}_2^T\mathbf{x}_3 \\ \mathbf{x}_3^T\mathbf{x}_1 & \mathbf{x}_3^T\mathbf{x}_2 & \mathbf{x}_3^T\mathbf{x}_3 \end{pmatrix}.$$

We have three training data vectors that we denote $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$. Using the derivations in 9.2 above, we can extract the following information from $\mathbf{K}$:

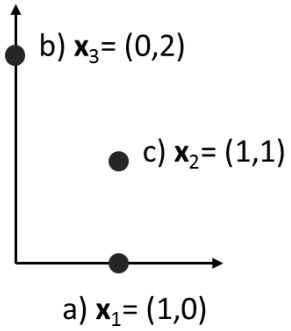
- $\|\mathbf{x}_1\| = \sqrt{\mathbf{x}_1 \cdot \mathbf{x}_1} = 1$, i.e., this point is 1 unit length from the origin of the space. Similarly, $\|\mathbf{x}_2\| = \sqrt{2}$ and $\|\mathbf{x}_3\| = 2$.
- The distances $\|\mathbf{x}_1 - \mathbf{x}_2\| = 1$, $\|\mathbf{x}_1 - \mathbf{x}_3\| = \sqrt{5}$ and $\|\mathbf{x}_2 - \mathbf{x}_3\| = \sqrt{2}$.
- The angle between $\mathbf{x}_1$ and $\mathbf{x}_2$ is 45°. The angle between $\mathbf{x}_1$ and $\mathbf{x}_3$ is 90° (orthogonal as the scalar product is 0).

Now we can reconstruct the data as follows (see figure c below):

a) Let's begin by placing $\mathbf{x}_1$ at any point with distance 1 from the origin.

b) Next, place $\mathbf{x}_3$ on an axis orthogonal (angle 90°) to $\mathbf{x}_1$, a distance 2 from the origin.

c) Finally, $\mathbf{x}_2$ must be at one of 2 possible positions that are at an angle 45° and a distance 1 relative $\mathbf{x}_1$. As the distance between $\mathbf{x}_2$ and $\mathbf{x}_3$ must also be $\sqrt{2}$, there is only one possible point left. Note that there is an infinite number of solutions depending on where we start with $\mathbf{x}_1$.



Answer 9.8

a)

$$\mathbf{K} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

b) The vectors in the feature space become

$$\varphi(\mathbf{x}_1) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \varphi(\mathbf{x}_2) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \varphi(\mathbf{x}_3) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

The average vector in the feature space then becomes

$$\overline{\varphi}(\mathbf{x}) = \frac{1}{3} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

The centered feature vectors the become $\varphi'(\mathbf{x}_i) = \varphi(\mathbf{x}_i) - \overline{\varphi}(\mathbf{x})$:

$$\varphi'(\mathbf{x}_1) = \frac{1}{3} \begin{bmatrix} 2 \\ 0 \\ 0 \\ -1 \end{bmatrix}, \quad \varphi'(\mathbf{x}_2) = \frac{1}{3} \begin{bmatrix} -1 \\ 0 \\ 0 \\ -1 \end{bmatrix}, \quad \varphi'(\mathbf{x}_3) = \frac{1}{3} \begin{bmatrix} -1 \\ 0 \\ 0 \\ 2 \end{bmatrix}$$

The centered kernel matrix then becomes

$$\text{K'} = \frac{1}{9} \begin{bmatrix} 5 & -1 & -4 \\ -1 & 2 & -1 \\ -4 & -1 & 5 \end{bmatrix}$$

This is the same matrix as we get if we subtract the row and column averages from K and add the total average.

Answer 9.9

The problem must be possible to formulate in terms of scalar products between the samples so that we can swap in a nonlinear kernel function here, the so-called kernel trick. A first step to do this is typically to show that the optimal solution can be written as a linear combination of the training data vectors $\mathbf{w}^* = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i$ for some values of the α's.

Answer 9.10

The original cost function is:

$$\min \|\mathbf{w}\|^2 = \mathbf{w}^T \mathbf{w}$$

subject to the constraint $y_i \left(\mathbf{w}^T \mathbf{x}_i + w_0 \right) \geq 1$ for all i.

The kernelized function can be derived for problems for which it can be shown that the optimal solution $\mathbf{w}^*$ lies in the span of the input training data, i.e., $\mathbf{w}^* = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i$, where α_i are some number, $\mathbf{x}_i$ is a traing data example and N is the number of training examples. Inserting this relationship in the original cost function yields $\mathbf{w}^T\mathbf{w} = \sum_{i=1}^{N}\sum_{i=j}^{N} \alpha_i\alpha_j \mathbf{x}_i^T\mathbf{x}_j$ (which can also be written in a vector form $\boldsymbol{\alpha}^T\mathbf{K}\boldsymbol{\alpha}$ using a kernel matrix $\mathbf{K}$). The entire kernelized cost function is

$$\min \sum_{i=1}^{N}\sum_{i=j}^{N} \alpha_i\alpha_j \mathbf{x}_i^T\mathbf{x}_j$$

$$\text{subject to the constraint } y_i\left(\alpha_i\mathbf{x}_i^T\mathbf{x}_i + \alpha_0\right) \geq 1 \text{ for all } i.$$

In non-linear kernel methods, the inner products $\mathbf{x}_i^T\mathbf{x}_j$ is replaced with a non-linear kernel function $\kappa(\mathbf{x}_i, \mathbf{x}_j)$.

Answer 9.11

The classification function in SVM has the value ± 1 for the support vectors. Hence, $\mathbf{x}$ is a support vector since $f(\mathbf{x}) = 1$