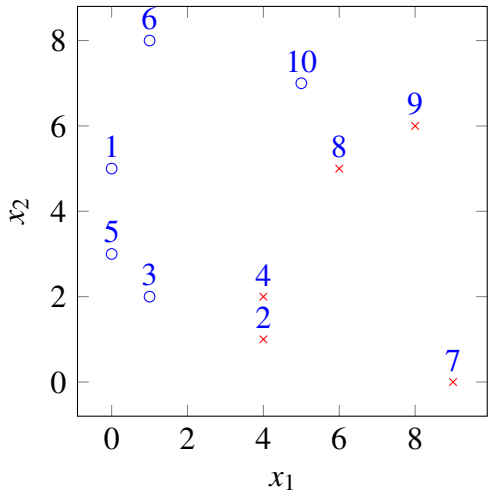
Statistical Machine Learning
Solutions for Exam 2019-03-15

1. i. **False**, A non-linear classifier can still produce a linear classifier.
   ii. **False**
   iii. **True**, Convolutional neural networks are well suited for problems where the data has a local structure, such as images or time series.
   iv. **True**
   v. **False**
   vi. **False**, If the data is collected in an non random fashion this could lead to test dataset only contains data of a certain type.
   vii. **True**
   viii. **False**
   ix. **True**
   x. **False**

2. (a) See figure



(b) For the first bootstrapped dataset we have

Data point index	1	2	4	4	6	7	8	9	9	10
Dimension x_2	5	1	2	2	8	0	5	6	6	7
Class	1	0	0	0	1	0	0	0	0	1

Ordering this table after the x_2 dimension

Dimension x_2	0	1	2	2	5	5	6	6	7	8
Class	0	0	0	0	0	1	0	0	1	1

We consider the possible splittings 3.5, 5.5 and 6.5 since all other possible splittings always could be improved to one of these. We could also argue that the split in 5.5 is inferior without doing the calculations.

Split	$\hat{\pi}_{1,1}$	$\hat{\pi}_{1,2}$	$\hat{\pi}_{2,1}$	$\hat{\pi}_{2,2}$	Cost
3.5	4/4	0/4	3/6	3/6	$0 + 3/6 \cdot (1 - 3/6) \cdot 6 + 3/6 \cdot (1 - 3/6) \cdot 6 = 3$
5.5	5/6	1/6	2/4	2/4	22/6
6.5	7/8	1/8	2/2	2/2	14/8

Here we calculated the cost as

$$\sum_{i}^{P} n_i \sum_{j}^{K} \hat{\pi}_{i,j}(1 - \hat{\pi}_{i,j})$$

where K is the number of classes, 2, P is the number of partitions, 2, and n_i is the number of samples in partition i. We see that the last split is yields the lowest cost so we choose this split.

We do a similar procedure for the next two bootstrapped datasets,

Data point index	1	2	3	4	5	5	6	7	7	9
Dimension x_2	5	1	2	2	3	3	8	0	0	6
Class	1	0	1	0	1	1	1	0	0	0

Ordering this table after the x_2 dimension

Dimension x_2	0	0	1	2	2	3	3	5	6	8
Class	0	0	0	1	0	1	1	1	0	1

We consider the possible splittings 1.5, 2.5, 5.5 and 7. Also in this case we can argue that the split in 5.5 is inferior.

Split	$\hat{\pi}_{1,1}$	$\hat{\pi}_{1,2}$	$\hat{\pi}_{2,1}$	$\hat{\pi}_{2,2}$	Cost
1.5	3/3	0/3	5/7	2/7	20/7
2.5	4/5	1/5	1/5	4/5	16/5
5.5	4/8	4/8	1/2	1/2	5
7	5/9	4/9	1/1	1/1	40/9

Here the split at 1.5 yields the lowest cost. Finally for the last dataset we get

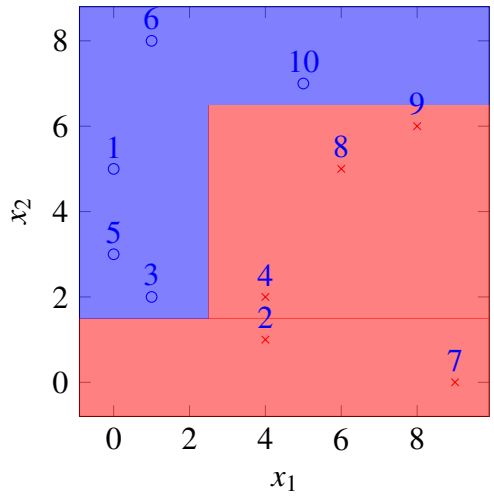
Data point index	1	1	4	5	5	5	6	9	9	9
Dimension x_1	0	0	4	0	0	0	1	8	8	8
Class	1	1	0	1	1	1	1	0	0	0

Ordering this table after the x_2 dimension

Dimension x_1	0	0	0	0	0	1	4	8	8	8
Class	1	1	1	1	1	1	0	0	0	0

we see that the data is split perfectly in 2.5.

(c) The final classifier is the majority decision from these 3 classifiers, we can visualize it as



This classifier for class 1 can be expressed as

$$G(x) = \mathbb{I}\left(\frac{1}{3}(\mathbb{I}(x_2 > 6.5) + \mathbb{I}(x_2 > 1.5) + \mathbb{I}(x_1 < 2.5)) > 0.5\right)$$

3. (a) During the training the model fits the specific examples in the trainingset. This includes the specific noise realizations in the trainingset (overfitting). The training does thus not contain any generalization error.

   (b) Cross-validation gives an estimate of the test error. Using cross-validation is better than using only one single split since you get multiples estimates which you use reduce the noise in your test error estimate.

   (c) i. Since this decision boundary is linear it most probably corresponding to LDA or logistic regression

   ii. This decision boundary looks quadratic hence, QDA is probably the source of this boundary

   iii. This decision boundary is very irregular, the only model that can give rise to this is the k-NN.

   iv. This decision boundary is related to a classification tree. However a tree of depth 2 is deep not enough to do all 4 splits that is required to represent this decision boundary hence only the random forest model can give raise to this decision boundary.

4. (a) The given training data defines the matrix

$$\mathbf{X} = \begin{bmatrix} 1 & -\mathbf{x}_1^\mathsf{T}- \\ 1 & -\mathbf{x}_2^\mathsf{T}- \\ 1 & -\mathbf{x}_3^\mathsf{T}- \end{bmatrix} = \begin{bmatrix} 1 & -1 & 2 \\ 1 & 0 & 0 \\ 1 & 1 & -2 \end{bmatrix}$$

   of input features and the vector

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ -3 \end{bmatrix}$$

   of outputs. We know that the least-squares estimate $\widehat{\boldsymbol{\beta}}$ is the solution to the normal equations $\mathbf{X}^\mathsf{T}\mathbf{X}\widehat{\boldsymbol{\beta}} = \mathbf{X}^\mathsf{T}\mathbf{y}$, i.e., to the linear equation

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & -4 \\ 0 & -4 & 8 \end{bmatrix} \widehat{\boldsymbol{\beta}} = \begin{bmatrix} 0 \\ -5 \\ 10 \end{bmatrix}.$$

   It follows that $\hat{\beta}_1 = 0$ and $\hat{\beta}_2 = 2\hat{\beta}_3 - 2.5$, i.e., the least-squares estimates of $\boldsymbol{\beta}$ are of the form

$$\widehat{\boldsymbol{\beta}} = \lambda \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 2.5 \\ 0 \end{bmatrix}, \qquad \lambda \in \mathbb{R}.$$

   In particular, since $\mathbf{X}^\mathsf{T}\mathbf{X}$ is not invertible there exists no unique solution $\widehat{\boldsymbol{\beta}}$ to the least-squares problem.

   (b) Loosely speaking, in regularized linear regression we try to keep the parameters small unless the data provides enough evidence. Hence by adding regularization we expect $\|\widehat{\boldsymbol{\beta}}\|$ to become smaller, i.e., the estimated parameter values to become closer to zero. Moreover, in contrast to the non-regularized linear regression in (a) ridge regression yields a unique parameter estimate $\widehat{\boldsymbol{\beta}}$.

(c) In a dense layer each input variable is connected to each hidden unit in the following layer, and each connection has a unique parameter associated with it. In a convolutional layer, however, a hidden unit only depends on a small subset of input variables, which corresponds to forcing most of the parameters in a dense layer to be equal to zero ("sparse interactions"). Moreover, in a convolutional layer the same set of parameters (a so-called kernel) is used for different hidden units ("parameter sharing").

(d) Both mini-batch gradient descent and gradient descent are numerical optimization algorithms that update approximate solutions in an iterative manner. In the training of neural networks both algorithms are used to find parameters that minimize a chosen cost function (for more details, see section 7.4 in the lecture notes). Gradient descent considers all data points in every update step whereas mini-batch gradient descent considers only a subset of data points, the so-called mini-batch. We can often assume that the data set is redundant and the gradient obtained by only considering a random subset of data points is similar to the gradient evaluated on the whole data set. Hence the main advantage of mini-batch gradient descent in comparison to gradient descent is the reduced computational time.

5. (a) We have

$$\mathbf{X}_{-i}^\top \mathbf{X}_{-i} = \sum_{j=1, j\neq i}^{n} \mathbf{x}_j \mathbf{x}_j^\top = \left(\sum_{j=1}^{n} \mathbf{x}_j \mathbf{x}_j^\top \right) - \mathbf{x}_i \mathbf{x}_i^\top = \mathbf{X}^\top \mathbf{X} - \mathbf{x}_i \mathbf{x}_i^\top$$

and similarly

$$\mathbf{X}_{-i}^\top \mathbf{y}_{-i} = \sum_{j=1, j\neq i}^{n} \mathbf{x}_j y_j = \left(\sum_{j=1}^{n} \mathbf{x}_j y_j^\top \right) - \mathbf{x}_i y_i = \mathbf{X}^\top \mathbf{y} - \mathbf{x}_i y_i.$$

For matrices $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $i \in \{1, \ldots, n\}$ we can express element $[\mathbf{A}]_{ii}$ as $[\mathbf{A}]_{ii} = \mathbf{e}_i^\top \mathbf{A} \mathbf{e}_i$, where $\mathbf{e}_i \in \mathbb{R}^n$ is the ith unit vector (its ith element is 1 and all other entries are 0). Thus we get

$$\begin{aligned}
[\mathbf{H}(\gamma)]_{ii} &= \mathbf{e}_i^\top \mathbf{H}(\gamma) \mathbf{e}_i = \mathbf{e}_i^\top \left(\mathbf{X} (\mathbf{X}^\top \mathbf{X} + \gamma \mathbf{I})^{-1} \mathbf{X}^\top \right) \mathbf{e}_i \\
&= \left(\mathbf{e}_i^\top \mathbf{X} \right) (\mathbf{X}^\top \mathbf{X} + \gamma \mathbf{I})^{-1} \left(\mathbf{X}^\top \mathbf{e}_i \right) = \mathbf{x}_i^\top (\mathbf{X}^\top \mathbf{X} + \gamma \mathbf{I})^{-1} \mathbf{x}_i,
\end{aligned}$$

where we used that $\mathbf{X}^\top \mathbf{e}_i = \mathbf{x}_i$.

(b) From the ridge regression expression we know that the parameters $\widehat{\boldsymbol{\beta}}_{-i}$ learned from all data points except i are given by

$$\widehat{\boldsymbol{\beta}}_{-i} = \left(\mathbf{X}_{-i}^\top \mathbf{X}_{-i} + \gamma \mathbf{I} \right)^{-1} \mathbf{X}_{-i}^\top \mathbf{y}_{-i}.$$

Using the first two equations from (a) we get

$$\widehat{\boldsymbol{\beta}}_{-i} = \left(\mathbf{X}^\top \mathbf{X} - \mathbf{x}_i \mathbf{x}_i^\top + \gamma \mathbf{I} \right)^{-1} (\mathbf{X}^\top \mathbf{y} - \mathbf{x}_i y_i). \tag{1}$$

Applying the matrix inversion lemma with $\mathbf{A} = \mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}$ and $\mathbf{v} = \mathbf{x}_i$ and using the third equation from (a) yields

$$\begin{aligned}\left(\mathbf{X}^\top\mathbf{X} - \mathbf{x}_i\mathbf{x}_i^\top + \gamma\mathbf{I}\right)^{-1} &= \left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1} + \frac{\left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{x}_i\mathbf{x}_i^\top\left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}}{1 - \mathbf{x}_i^\top\left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{x}_i} \\ &= \left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1} + \frac{\left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{x}_i\mathbf{x}_i^\top\left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}}{1 - [\mathbf{H}(\gamma)]_{ii}}.\end{aligned}$$

If we plug this result in eq. (1) and use the third equation from (a) we get

$$\begin{aligned}\widehat{\boldsymbol{\beta}}_{-i} &= \left(\left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1} + \frac{\left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{x}_i\mathbf{x}_i^\top\left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}}{1 - [\mathbf{H}(\gamma)]_{ii}}\right)(\mathbf{X}^\top\mathbf{y} - \mathbf{x}_i y_i) \\ &= \left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{X}^\top\mathbf{y} + \frac{1}{1 - [\mathbf{H}(\gamma)]_{ii}}\left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{x}_i\mathbf{x}_i^\top\left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{X}^\top\mathbf{y} \\ &\quad - \left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{x}_i\left(1 + \frac{\mathbf{x}_i^\top\left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{x}_i}{1 - [\mathbf{H}(\gamma)]_{ii}}\right)y_i \\ &= \left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{X}^\top\mathbf{y} + \frac{1}{1 - [\mathbf{H}(\gamma)]_{ii}}\left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{x}_i\mathbf{x}_i^\top\left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{X}^\top\mathbf{y} \\ &\quad - \frac{1}{1 - [\mathbf{H}(\gamma)]_{ii}}\left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{x}_i y_i \\ &= \left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{X}^\top\mathbf{y} \\ &\quad + \frac{1}{1 - [\mathbf{H}(\gamma)]_{ii}}\left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{x}_i\left(\mathbf{x}_i^\top\left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{X}^\top\mathbf{y} - y_i\right).\end{aligned}$$

The parameters $\widehat{\boldsymbol{\beta}}$ learned from all data points satisfy

$$\widehat{\boldsymbol{\beta}} = \left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{X}^\top\mathbf{y},$$

and hence the model learned from all data points predicts the value

$$\hat{y}_i = \widehat{\boldsymbol{\beta}}^\top\mathbf{x}_i = \mathbf{x}_i^\top\widehat{\boldsymbol{\beta}} = \mathbf{x}_i^\top\left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{X}^\top\mathbf{y}$$

for the ith data point. Thus we obtain

$$\widehat{\boldsymbol{\beta}}_{-i} = \widehat{\boldsymbol{\beta}} + \frac{1}{1 - [\mathbf{H}(\gamma)]_{ii}}\left(\mathbf{X}^\top\mathbf{X} + \gamma\mathbf{I}\right)^{-1}\mathbf{x}_i\left(\hat{y}_i - y_i\right).$$

(c) Using the result from (b) we get

$$
\begin{aligned}
E_{\text{val}} &= \frac{1}{n}\sum_{i=1}^{n}\left(\widehat{\boldsymbol{\beta}}_{-i}^{\mathsf{T}}\mathbf{x}_i - y_i\right)^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(\left(\widehat{\boldsymbol{\beta}} + \frac{1}{1-[\mathbf{H}(\gamma)]_{ii}}\left(\mathbf{X}^{\mathsf{T}}\mathbf{X}+\gamma\mathbf{I}\right)^{-1}\mathbf{x}_i\left(\hat{y}_i - y_i\right)\right)^{\mathsf{T}}\mathbf{x}_i - y_i\right)^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(\left(\widehat{\boldsymbol{\beta}}^{\mathsf{T}} + \frac{1}{1-[\mathbf{H}(\gamma)]_{ii}}\mathbf{x}_i^{\mathsf{T}}\left(\mathbf{X}^{\mathsf{T}}\mathbf{X}+\gamma\mathbf{I}\right)^{-1}\left(\hat{y}_i - y_i\right)\right)\mathbf{x}_i - y_i\right)^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(\widehat{\boldsymbol{\beta}}^{\mathsf{T}}\mathbf{x}_i + \frac{1}{1-[\mathbf{H}(\gamma)]_{ii}}\mathbf{x}_i^{\mathsf{T}}\left(\mathbf{X}^{\mathsf{T}}\mathbf{X}+\gamma\mathbf{I}\right)^{-1}\mathbf{x}_i\left(\hat{y}_i - y_i\right) - y_i\right)^2.
\end{aligned}
$$

Since $\hat{y}_i = \widehat{\boldsymbol{\beta}}^{\mathsf{T}}\mathbf{x}_i$ and from (a) we know that $[\mathbf{H}(\gamma)]_{ii} = \mathbf{x}_i^{\mathsf{T}}\left(\mathbf{X}^{\mathsf{T}}\mathbf{X}+\gamma\mathbf{I}\right)^{-1}\mathbf{x}_i$, we obtain

$$
\begin{aligned}
E_{\text{val}} &= \frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}_i + \frac{1}{1-[\mathbf{H}(\gamma)]_{ii}}[\mathbf{H}(\gamma)]_{ii}\left(\hat{y}_i - y_i\right) - y_i\right)^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\hat{y}_i - y_i}{1-[\mathbf{H}(\gamma)]_{ii}}\right)^2.
\end{aligned}
$$

(d) Ideally we would like to choose regularization parameter γ in ridge regression such that the new data error of the resulting trained model is minimized. Unfortunately, usually we can not compute the new data error. The cross validation error E_{val} provides an approximation of the new data error, and hence can be used to select a "good" γ. However, performing c-fold cross validation can be computationally expensive and time consuming, since usually it requires training the model from scratch and evaluating its performance c times. In particular, performing leave-one-out cross validation with $c = n$ might not be feasible. In ridge regression, however, eq. (1) allows us to train the model only once on the full dataset and to compute the leave-one-out cross validation error from its prediction errors without having to retrain the model n times on all subsets of our dataset with $n - 1$ samples.