SGN-41007 Pattern Recognition and Machine Learning
Exam 12.2.2020
Heikki Huttunen

- ▷ Use of calculator is allowed.
- ▷ Use of other materials is not allowed.
- ▷ The exam questions need not be returned after the exam.
- ▷ You may answer in English or Finnish.

1. Are the following statements true or false? No need to justify your answer, just T or F. Correct answer: 1 pts, wrong answer: $-\frac{1}{2}$ pts, no answer 0 pts.

   (a) The Receiver Operating Characteristics curve plots the probability of detection versus the probability of false alarm for all thresholds.

   (b) The number of support vectors of a support vector machine equals the total number of samples.

   (c) A neural network classifier has a linear decision boundary between classes.

   (d) The LDA maximizes the following score:

   $$J(\mathbf{w}) = \frac{\text{Mean variance of each class}}{\text{Squared distance of class means}}$$

   (e) Maxpooling computes the maximum over neighboring pixels of distinct blocks.

   (f) Cross-validation is used for model accuracy evaluation.

2. Consider N independent measurements $x_0, x_1, \ldots, x_{N-1} \in \mathbb{R}_+$ from the PDF

   $$p(x;\theta) = \begin{cases} \theta^2 x \exp(-\theta x), & \text{when } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

   where $\theta > 0$ is the parameter to be estimated.

   (a) Compute the probability $p(\mathbf{x};\theta)$ of observing the samples $\mathbf{x} = (x_0, x_1, \ldots, x_{N-1})$. (2p)

   (b) Compute the logarithm of $p(\mathbf{x};\theta)$ and differentiate the result with respect to θ. (2p)

   (c) Find the maximum of the function, *i.e.*, the value where $\frac{\partial}{\partial\theta} \log p(\mathbf{x};\theta) = 0$. (2p)

3. *Count the number of parameters in a neural network*

   (a) Consider the traditional shallow neural network architecture of Figure 3. Suppose our inputs are 32×32 RGB bitmaps of two categories of traffic signs.
   Let the network structure be the following:
   - On the 1st layer there are 100 nodes (marked in blue)
   - On the 2nd layer there are 100 nodes (marked in blue)
   - On the 3rd (output) layer there are 10 nodes (marked in blue; one for each class)

   Compute the number of parameters (coefficients) in the net.

	Prediction	True label
Sample 1	0.8	1
Sample 2	0.5	1
Sample 3	0.6	0
Sample 4	0.4	0
Sample 5	0.1	0

Table 1: Results on test data for question 5a.

(b) Consider the neural network defined in Figure 1.[1] Inputs are the same as in (a).

i. Compute the number of parameters for each layer, and their total number over all layers.

ii. Compute the number of multiplications required on the first convolutional layer.

4. Compute the LDA weight vector for

$$\mu_0 = \begin{pmatrix} 2 \\ -1 \end{pmatrix} \qquad \mu_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\Sigma_0 = \begin{pmatrix} 1 & 1 \\ 1 & 3 \end{pmatrix} \qquad \Sigma_1 = \begin{pmatrix} 1 & 1 \\ 1 & 4 \end{pmatrix}.$$

5. (a) A random forest classifier is trained on training data set and the `predict_proba` method is applied on the test data of Table 1. Draw the receiver operating characteristic curve. What is the Area Under Curve (AUC) score?

(b) Draw the precision recall curve. What is the Area Under PR Curve (AUPRC) score?

[1] Although we did not study pytorch extensively during the course, if should be readable because the building blocks are the normal ones.

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        ''' The constructor defines the building blocks
            needed for forward pass '''

        super(Net, self).__init__()

        # 3 input image channels, 6 output channels,
        # 5x5 square convolution window. Before the operation
        # we pad each side of the image with 2 rows of zeros.
        self.conv1 = nn.Conv2d(3, 6, 5, padding = 2)
        self.conv2 = nn.Conv2d(6, 16, 3, padding = 1)

        # An affine operation: y = Wx + b
        # Arguments are the dimensions of W.
        self.fc1 = nn.Linear(1024, 100)
        self.fc2 = nn.Linear(100, 10)

    def forward(self, x):
        ''' Executed at every forward pass of the network. '''

        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv2(x)), (2, 2))

        # Flatten the data into a vector.
        # First dim is the sample dimension.
        # '-1' forces to compute the size automatically.
        x = x.reshape(-1, 1024)

        x = F.relu(self.fc1(x))
        x = self.fc2(x)

        return x

net = Net()
```

Figure 1: Network definition for Question 3b.

Related Wikipedia pages

Another complication in applying LDA and Fisher's discriminant to real data occurs when the number of measurements of each sample (i.e., the dimensionality of each data vector) exceeds the number of samples in each class.[4] In this case, the covariance estimates do not have full rank, and so cannot be inverted. There are a number of ways to deal with this. One is to use a pseudo inverse instead of the usual matrix inverse in the above formulae. However, better numeric stability may be achieved by first projecting the problem onto the subspace spanned by Σ_b.[22] Another strategy to deal with small sample size is to use a shrinkage estimator of the covariance matrix, which can be expressed mathematically as

$$\Sigma = (1-\lambda)\Sigma + \lambda I$$

where I is the identity matrix, and λ is the *shrinkage intensity* or *regularisation parameter*. This leads to the framework of regularized discriminant analysis[23] or shrinkage discriminant analysis.[24]

The terms *Fisher's linear discriminant* and *LDA* are often used interchangeably, although Fisher's original article[1] actually describes a slightly different discriminant, which does not make some of the assumptions of LDA such as normally distributed classes or equal class covariances.

Suppose two classes of observations have means $\vec{\mu}_0, \vec{\mu}_1$ and covariances Σ_0, Σ_1. Then the linear combination of features $\vec{w} \cdot \vec{x}$ will have means $\vec{w} \cdot \vec{\mu}_i$ and variances $\vec{w}^T \Sigma_i \vec{w}$ for $i = 0, 1$. Fisher defined the separation between these two distributions to be the ratio of the variance between the classes to the variance within the classes:

$$S = \frac{\sigma^2_{\text{between}}}{\sigma^2_{\text{within}}} = \frac{(\vec{w}\cdot\vec{\mu}_1 - \vec{w}\cdot\vec{\mu}_0)^2}{\vec{w}^T\Sigma_1\vec{w} + \vec{w}^T\Sigma_0\vec{w}} = \frac{(\vec{w}\cdot(\vec{\mu}_1 - \vec{\mu}_0))^2}{\vec{w}^T(\Sigma_0+\Sigma_1)\vec{w}}$$

This measure is, in some sense, a measure of the signal-to-noise ratio for the class labelling. It can be shown that the maximum separation occurs when

$$\vec{w} \propto (\Sigma_0 + \Sigma_1)^{-1}(\vec{\mu}_1 - \vec{\mu}_0)$$

When the assumptions of LDA are satisfied, the above equation is equivalent to LDA.

Inversion of 2 × 2 matrices [edit]

The *cofactor equation* listed above yields the following result for 2 × 2 matrices. Inversion of these matrices can be done as follows:[6]

$$\mathbf{A}^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{\det \mathbf{A}} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}.$$

This is possible because $1/(ad - bc)$ is the reciprocal of the determinant of the matrix in question, and the same strategy could be used for other matrix sizes.

Tikhonov regularization, named for Andrey Tikhonov, is a method of regularization of ill-posed problems. Also known as **ridge regression**,[6] it is particularly useful to mitigate the problem of multicollinearity in linear regression, which commonly occurs in models with large numbers of parameters.[7] In general, the method provides improved efficiency in parameter estimation problems in exchange for a tolerable amount of bias (see bias–variance tradeoff).[8]

In the simplest case, the problem of a near-singular moment matrix $(\mathbf{X}^\mathsf{T}\mathbf{X})$ is alleviated by adding positive elements to the diagonals. The approach can be conceptualized by posing a constraint $\sum \beta_i^2 = c$ to the least squares problem, such that

$$\min_\beta (\mathbf{y} - \mathbf{X}\beta)^\mathsf{T}(\mathbf{y} - \mathbf{X}\beta) + \lambda(\beta^\mathsf{T}\beta - c)$$

where λ is the Lagrange multiplier of the constraint. The minimizer of the problem is the simple ridge estimator

$$\hat{\beta}_R = (\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y}$$

where $\mathbf{I}$ is the identity matrix and the ridge parameter λ serves as the positive constant shifting the diagonals,[3] thereby decreasing the condition number of the moment matrix. A more general approach to Tikhonov regularization is discussed below.

The ROC curve simply plots $T(t)$ against $F(t)$ while varying t from 0 to 1. Thus, if we view T as a function of F, the AUC can be rewritten as follows.

$$\begin{aligned}
\text{AUC} &= \int_0^1 T(F_0)\,\mathrm{d}F_0 \\
&= \int_0^1 P[\hat{p}(\mathbf{x}) > F^{-1}(F_0) \mid y(\mathbf{x}) = 1]\,\mathrm{d}F_0 \\
&= \int_1^0 P[\hat{p}(\mathbf{x}) > F^{-1}(F(t)) \mid y(\mathbf{x}) = 1] \cdot \frac{\partial F(t)}{\partial t}\,\mathrm{d}t \\
&= \int_0^1 P[\hat{p}(\mathbf{x}) > t \mid y(\mathbf{x}) = 1] \cdot P[\hat{p}(\mathbf{x}') = t \mid y(\mathbf{x}') = 0]\,\mathrm{d}t \\
&= \int_0^1 P[\hat{p}(\mathbf{x}) > \hat{p}(\mathbf{x}') \,\&\, \hat{p}(\mathbf{x}') = t \mid y(\mathbf{x}) = 1 \,\&\, y(\mathbf{x}') = 0]\,\mathrm{d}t \\
&= P[\hat{p}(\mathbf{x}) > \hat{p}(\mathbf{x}') \mid y(\mathbf{x}) = 1 \,\&\, y(\mathbf{x}') = 0],
\end{aligned}$$

where we used the fact that the probability density function

$$P[\hat{p}(\mathbf{x}') = t \mid y(\mathbf{x}') = 0] =: f(t)$$

is the derivative with respect to t of the cumulative distribution function

$$P[\hat{p}(\mathbf{x}') \le t \mid y(\mathbf{x}') = 0] = 1 - F(t).$$

So, given a randomly chosen observation $\mathbf{x}$ belonging to *class 1*, and a randomly chosen observation $\mathbf{x}'$ belonging to class *0*, the AUC is the probability that the evaluated classification algorithm will assign a higher score to $\mathbf{x}$ than to $\mathbf{x}'$, i.e., the conditional probability of $\hat{p}(\mathbf{x}) > \hat{p}(\mathbf{x}')$.

ROC space [edit]

The contingency table can derive several evaluation "metrics" (see infobox). To draw a ROC curve, only the true positive rate (TPR) and false positive rate (FPR) are needed (as functions of some classifier parameter). The TPR defines how many correct positive results occur among all positive samples available during the test. FPR, on the other hand, defines how many incorrect positive results occur among all negative samples available during the test.

A ROC space is defined by FPR and TPR as *x* and *y* axes respectively, which depicts relative trade-offs between true positive (benefits) and false positive (costs). Since TPR is equivalent to sensitivity and FPR is equal to 1 − specificity, the ROC graph is sometimes called the sensitivity vs (1 − specificity) plot. Each prediction result or instance of a confusion matrix represents one point in the ROC space.

space

For degree-*d* polynomials, the polynomial kernel is defined as[2]

$$K(x, y) = (x^\mathsf{T} y + c)^d$$

where *x* and *y* are vectors in the *input space*, i.e. vectors of features computed from training or test samples and $c \ge 0$ is a free parameter trading off the influence of higher-order versus lower-order terms in the polynomial. When $c = 0$, the kernel is called homogeneous.[3] (A further generalized polykernel divides $x^\mathsf{T}y$ by a user-specified scalar parameter a.[4])

As a kernel, K corresponds to an inner product in a feature space based on some mapping φ:

$$K(x, y) = \langle \varphi(x), \varphi(y) \rangle$$

The nature of φ can be seen from an example. Let $d = 2$, so we get the special case of the quadratic kernel. After using the multinomial theorem (twice—the outermost application is the binomial theorem) and regrouping,

$$K(x, y) = \left(\sum_{i=1}^n x_i y_i + c\right)^2 = \sum_{i=1}^n (x_i^2)(y_i^2) + \sum_{i=2}^n \sum_{j=1}^{i-1} (\sqrt{2} x_i x_j)(\sqrt{2} y_i y_j) + \sum_{i=1}^n (\sqrt{2c} x_i)(\sqrt{2c} y_i) + c^2$$

From this it follows that the feature map is given by:

$$\varphi(x) = \langle x_n^2, \ldots, x_1^2, \sqrt{2} x_n x_{n-1}, \ldots, \sqrt{2} x_n x_1, \sqrt{2} x_{n-1} x_{n-2}, \ldots, \sqrt{2} x_{n-1} x_1, \ldots, \sqrt{2} x_2 x_1, \sqrt{2c} x_n, \ldots, \sqrt{2c} x_1, c \rangle$$

		True condition				
	Total population	Condition positive	Condition negative	Prevalence $= \frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) $= \frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive,** Power	**False positive,** Type I error	Positive predictive value (PPV), Precision $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
	Predicted condition negative	**False negative,** Type II error	**True negative**	False omission rate (FOR) $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR−}}$	F_1 score $= \frac{1}{\frac{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}{2}}$
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) $= \frac{\text{FNR}}{\text{TNR}}$		

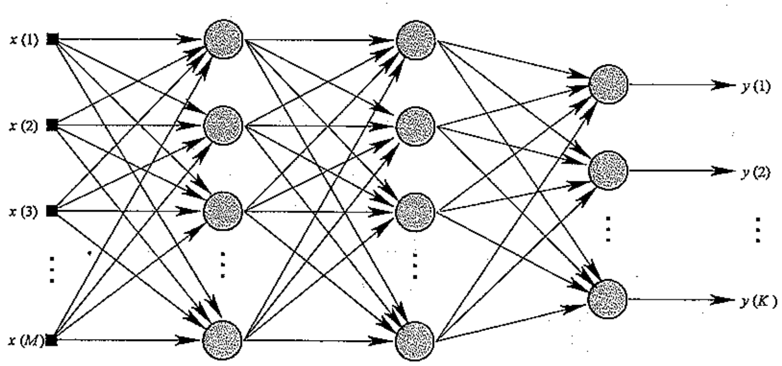

Figure 2: Vanilla neural network.

Figure 3: Network definition for Question 3a.