University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 9: October 2

Lecturer: Alexander Schwing Scribe: Diab Abueidda

Recommended reading material: K. Murphy; Machine Learning: A Probabilistic Perspective; Chapter 14

1 Introduction

So far, we have covered binary classifications using three frameworks: linear regression, logistic regression, and support vector machine (SVM). Then, the log- and hinge-loss functions were combined to build a general binary classifier:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \frac{C}{2}\|\mathbf{w}\|_2^2 + \sum_{i\in D} \epsilon log\left(1 + exp\left(\frac{L - y^{(i)}\mathbf{w}^T\phi(\mathbf{x}^{(i)})}{\epsilon}\right)\right). \tag{1}$$

One can obtain the logistic regression classifier from the general binary classifier if ϵ is set to 1 and L is set to 0. However, taking the limit as ϵ approaches 0 and setting L to 1 yield the binary SVM framework.

Although the classifiers mentioned above have two possible answers (binary classifiers), they are still useful for different applications such as edge detection and object detection. One famous example is realizing if a cat appears in an image or not. The purpose of this lecture is to extend these binary classifiers to multiclass classifiers, classifiers considering more than two possible answers (multiple classes). Multiclass classification frameworks allow us to classify; for example, whether the image shown in Figure 1 is a car, a truck, a recreational vehicle, an ambulance truck, or a fire truck. The intuitive approach to solve multiclass classification tasks is to transform the multiclass classification task into multiple binary classification tasks. This can be in the form of **1 vs all** or **1 vs 1**.

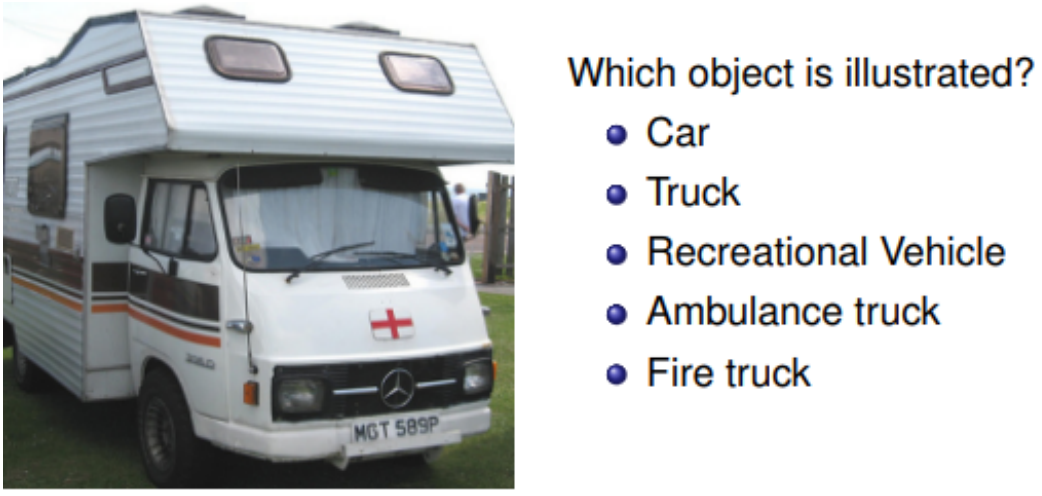


Figure 1: Example of multiclass classification tasks

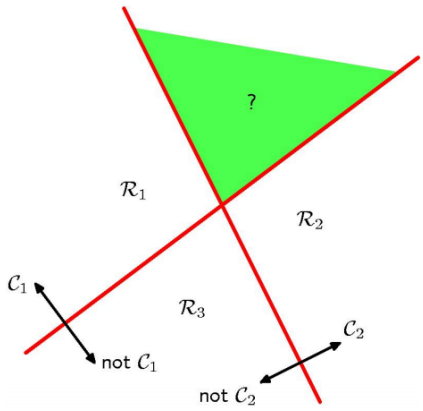


Figure 2: Illustration of **1 vs all** classifier

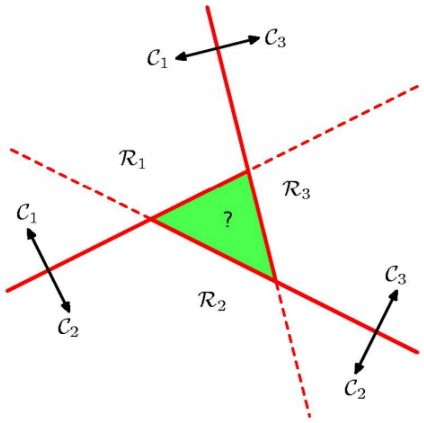


Figure 3: Illustration of **1 vs 1** classifier

1.1 1 vs all classifier

1 vs all (or **1 vs rest**) is based on training a K binary classifier where K is the number of classes. Then a classifier f_c separates the points of that class from points not in the class by assigning positive values for points in class c and negative values for all other points [1]. This may lead to regions of input space which are ambiguously labeled [4]. For example, let's consider a case with two classifiers as shown in Figure 2. One can infer that points in R_1 and R_2 belong to C_1 and C_2, respectively. Points in R_3 are classified as neither C_1 nor C_2 which can be a non-desirable output. An issue raises with this framework; points in the green region are predicted to be in C_1 and C_2, simultaneously.

1.2 1 vs 1 classifier

Assuming that we are interested in solving a multiclass classification problem that has K classes, one needs $\frac{K(K-1)}{2}$ binary classifiers, one for each possible pair of classes. Each classifier is trained using the samples corresponding to the pair of classes from the original training set [1]. Then, we classify points according to the majority of votes. Similar to the **1 vs all** classifier, the **1 vs 1** classifier results in ambiguities. From the illustrative example portrayed in Figure 3, one can infer that a tie is obtained for all points located in the green region. In other words, two-way preferences need not be transitive.

It is worth mentioning that all the difficulties and ambiguities encountered when using the **1 vs all** and **1 vs 1** classifiers are due to modeling uncertainties using non-probabilistic models. Non-probabilistic models lead to output scores that are non-comparable across classes [4]. In order to resolve this issue, two approaches, namely multiclass logistic regression and multiclass SVM, are used. Both approaches are discussed below.

2 Multiclass logistic regression

This section extends the binary logistic regression classifier to a multiclass classifier. Considering a probabilistic perspective in performing multiclass classification tasks can help us resolve the issues arisen when the **1 vs all** and **1 vs 1** classifiers. When we have a classification task with K classes, we use a multinomial distribution over $y \in \{0, 1, ..., K-1\}$, and we use K weight vectors $\mathbf{w}_{(y)}$, one vector per class.

Recall: Multinomial distribution is a generalization of binomial distribution, and it has the following characteristics: the experiment has N different samples where each sample has a finite (discrete) number of possible outputs (classes), the output of each sample is independent on the output of other samples (one sample does not affect other samples), and the probability of a specific output to occur is constant [2].

After stating the characteristics of multinomial distribution, we are interested in parametrizing the multinomial distribution. We use the following probability definition to parametrize it:

$$p(y = k|x^{(i)}) = \frac{exp(\mathbf{w}_{(k)}^T \phi(x^{(i)}))}{\sum_{j \in \{0,1,...K-1\}} exp(\mathbf{w}_{(j)}^T \phi(x^{(i)}))}. \tag{2}$$

.

To find the weight vectors, we maximize the likelihood (equivalent to minimizing the negative log likelihood) of a given dataset $D = \{(x^{(i)}, y^{(i)})\}$ assuming training examples are independent and identically distributed (**iid**).

$$\underset{\mathbf{w}}{\text{arg maximize}} \prod_{(x^{(i)},y^{(i)}) \in D} p(y = y^{(i)}|x^{(i)}) = \underset{\mathbf{w}}{\text{arg minimize}} \sum_{(x^{(i)},y^{(i)}) \in D} -log\big(p(y = y^{(i)}|x^{(i)})\big). \tag{3}$$

.

Using the log of the likelihood allows us to sum the probabilities. Then gradient descent or other optimization algorithms can be utilized to find the $\mathbf{w}$. Now, let's consider the example to illustrated in Figure 4a and understand what the $\mathbf{w}$ indicates. In this example, we have three-dimensional feature vector with bias $\phi_3(x^{(i)}) = 1$. Having a bias term is crucial as it gives us more freedom to divide the feature space. Without a bias term, all the lines have to pass through the origin. In this example, we also have three possible classes, red, blue, and green. Hence, we have three weight vectors. These weight vectors should be three-dimensional as we have three-dimensional feature vector. Note that the lines appearing in 4a **do not** classify the data in a binary fashion (e.g., green or not green). They tell us the scores for each class for a given data point. The further the data point from the line, the larger the absolute value of the score where the sign of the score depends on where the data points located with respect to the line. More specifically, each of these lines separates the feature space into two regions where the line tends to assign positive scores to data points corresponding to its weight vector. Points located on the other side of the feature space are assigned negative scores. For instance, the blue line assigns high positive scores to the blue points as they are far from the line while the green points are assigned small positive scores, and the red points are assigned small negative scores. Now, let's consider the green line. The green points are assigned high positive scores, blue points are assigned small negative scores, and red points are assigned relatively large negative scores.

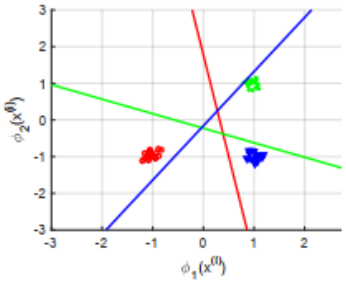
Recall: high positive scores correspond to high probabilities (probabilities close to 1) while negative scores yield probabilities close to zero.

After finding the weight vectors, one can create the decision boundaries. Dealing with decision boundaries is more intuitive than coping with scores. The decision boundaries for the multiclass classifier mentioned earlier are formed and shown in Figure 4b. Also, 4c depicts the probability distribution of the red over the feature space. The probability of red is close to one in the red region, and it drastically drops to values close to zero at the decision boundaries.

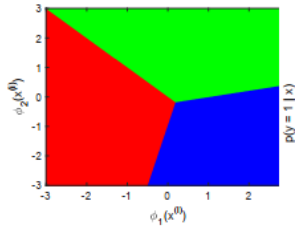
Now, let's consider the case when we have only two classes. Hence, we have two weight vectors and two decision boundaries. The decision boundaries are related to the line obtained from binary logistic classifier such that the average of the two decision boundaries obtained from the multiclass classifier is the same as the decision boundary attained from the binary classifier assuming global optimality is achieved for both classifiers. Figure 5 portrays an example of using a multiclass classifier to perform a binary classification task. We can see that the average of the green and

red lines is the same as the line obtained from a binary classifier. In this particular example, the parameters are tuned such that the red and green lines coincide on each other.

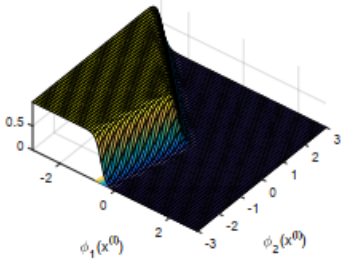
So far, we have discussed extending binary logistic regression classifiers to multiclass logistic regression classifiers. In the next section, we will discuss generalizing binary support vector machine classifiers to multiclass support vector machine classifiers.



a) Setup



b) Decision boundary



c) Probability for red

Figure 4: Example of multiclass classification task with $\phi_3(x^{(i)}) = 1$

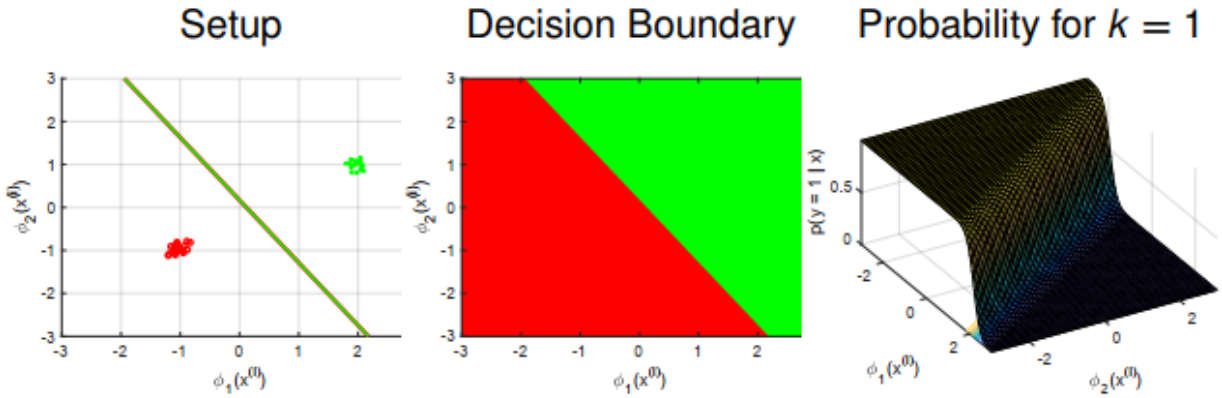


Figure 5: Demonstration of using multiclass classifier to solve a binary classification task.

3 Multiclass support vector machine

In binary SVM classifiers depicted in Figure 6, we tried to maximize the margin $\frac{2}{\|\mathbf{w}\|}$ after introducing a slack variable ζ to deal with data not linearly separable. Having said that, the optimization problem we were interested in solving was

$$\underset{\mathbf{w},\zeta^{(i)}\geq 0}{\text{minimize}} \quad \frac{C}{2}\|\mathbf{w}\|_2^2 + \sum_{i\in D}\zeta^{(i)} \quad s.t. \quad y^{(i)}\mathbf{w}^T\phi(x^{(i)}) \geq 1-\zeta^{(i)} \quad \forall i \in D. \tag{4}$$

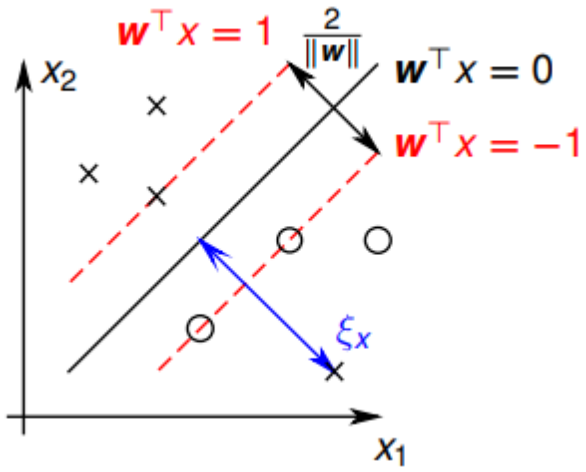


Figure 6: Illustration of binary SVM

Similar to multiclass logistic regression classifier, we use one weight vector per class, and then calculate the scores for the different classes. The basic idea behind multiclass SVM classifier is that the score for the ground-truth class should be larger than those of other classes by some margin.

$$\mathbf{w}_{y^{(i)}}^T\phi(x^{(i)}) \geq \mathbf{w}_{\bar{y}}^T\phi(x^{(i)}) \quad \forall i \in D, \bar{y} \in \{0,1,2,...,K-1\}). \tag{5}$$

To ensure that we have the following optimization problem

$$\underset{\mathbf{w},\zeta^{(i)}\geq 0}{\text{minimize}} \quad \frac{C}{2}\|\mathbf{w}\|_2^2 + \sum_{i\in D}\zeta^{(i)} \quad s.t. \quad \mathbf{w}_{y^{(i)}}^T\phi(x^{(i)}) - \mathbf{w}_{\bar{y}}^T\phi(x^{(i)}) \geq 1-\zeta^{(i)}) \quad \forall i, \bar{y} \tag{6}$$

where w is the concatenation of all $\mathbf{w}_{y^{(i)}}$

$$\mathbf{w} = \begin{bmatrix} \mathbf{w}_0 \\ \vdots \\ \mathbf{w}_{K-1} \end{bmatrix}. \tag{7}$$

Now, we introduce a new variable ψ which is the stacking of all feature vectors ϕ, ψ is defined as

$$\psi(x^{(i)}, y^{(i)}) = \begin{bmatrix} \phi(x^{(i)})\delta(y^{(i)}=0) \\ \vdots \\ \phi(x^{(i)})\delta(y^{(i)}=K-1) \end{bmatrix}. \tag{8}$$

Introducing this variable is useful in terms of making the formulation more concise and easier to follow. **Recall** that is equal to 1 if its argument is true, and it is zero otherwise. Using of $\mathbf{w}$ and ψ allows us to reduce the optimization problem stated in equation 6 to

$$\underset{\mathbf{w},\zeta^{(i)}\geq 0}{\text{minimize}} \quad \frac{C}{2}\|\mathbf{w}\|_2^2 + \sum_{i\in D}\zeta^{(i)} \quad s.t. \quad \mathbf{w}^T\big(\psi(x^{(i)}, y^{(i)}) - \psi(x^{(i)}, \bar{y})\big) \geq 1-\zeta^{(i)}) \quad \forall i, \bar{y}. \tag{9}$$

It is worth mentioning that the superscript $^{(i)}$ indicates the data point (i). Hence, we have one slack variable per data point. Now, let's get rid of the slack variables by reformulation the problem to the following form:

$$\underset{\mathbf{w},\zeta^{(i)}\geq 0}{\text{minimize}} \quad \frac{C}{2}\left\|\mathbf{w}\right\|_2^2 + \sum_{i\in D} max\{0, \underbrace{\underset{\bar{y}}{max}(1-\mathbf{w}^T(\psi(x^{(i)},y^{(i)})-\psi(x^{(i)},\bar{y})))}_{\geq 0}\} \quad \forall i,\bar{y}. \tag{10}$$

The term $\underset{\bar{y}}{max}(1-\mathbf{w}^T(\psi(x^{(i)},y^{(i)})-\psi(x^{(i)},\bar{y})))$ is ≥ 0. Hence, equation 10 can be rewritten as

$$\underset{\mathbf{w},\zeta^{(i)}\geq 0}{\text{minimize}} \quad \frac{C}{2}\left\|\mathbf{w}\right\|_2^2 + \sum_{i\in D} \underset{\bar{y}}{max}(1-\mathbf{w}^T(\psi(x^{(i)},y^{(i)})-\psi(x^{(i)},\bar{y}))) \quad \forall i,\bar{y}. \tag{11}$$

Since the maximization problem appearing in the second term does not depend on $y^{(i)}$, terms in equation 12 can be rearranged

$$\underset{\mathbf{w},\zeta^{(i)}\geq 0}{\text{minimize}} \quad \frac{C}{2}\left\|\mathbf{w}\right\|_2^2 + \sum_{i\in D} \underset{\bar{y}}{max}(1+\mathbf{w}^T\psi(x^{(i)},\bar{y})) - \mathbf{w}^T\psi(x^{(i)},y^{(i)}) \quad \forall i,\bar{y}. \tag{12}$$

where $\underset{\bar{y}}{max}(1+\mathbf{w}^T\psi(x^{(i)},\bar{y}))$ is commonly called the loss-augmented inference. Similar to the case of binary classification, multiclass logistic and SVM classifiers can be combined, and this yield the general multiclass classification

$$\underset{\mathbf{w}}{\text{minimize}} \quad \frac{C}{2}\left\|\mathbf{w}\right\|_2^2 + \sum_{i\in D} \epsilon log \sum_{\bar{y}} exp\left(\frac{L(y^{(i)},\bar{y}) + \mathbf{w}^T\psi(x^{(i)},\bar{y})}{\epsilon}\right) - \mathbf{w}^T\psi(x^{(i)},y^{(i)}). \tag{13}$$

It is noteworthy that we can get back to the binary classifier if we use one weight vector $\mathbf{w}$ and replace ψ with ϕ. This general multiclass classifier is still in the realm of linear inference models. In many cases, this linearity restricts the accuracy of classifiers. To tackle this issue, kernels and deepnets are introduced. Deepnets are out of the scope of this lecture, and they are covered in the next lecture. The rest of the lecture discusses the incorporation of nonlinearities through the use of kernels.

4 Kernel trick

To understand the need for kernel, we look into the dual of logistic regression and dual of SVM covered in earlier lectures.

- Dual of logistic regression:

$$\underset{0\leq\lambda^{(i)}\leq 1}{\max} \quad g(\lambda) := \frac{-1}{2C}\left\|\sum_i \lambda^{(i)}y^{(i)}\phi(x^{(i)})\right\|_2^2 + \sum_i H(\lambda^{(i)}) \tag{14}$$

  - Prediction with dual variables:

$$\mathbf{w}^T\phi(x) = \frac{1}{C}\sum_i \lambda^{(i)}y^{(i)}\phi^T(x^{(i)})\phi(x) \tag{15}$$

- Dual of SVM:

$$\max_{0\leq\alpha^{(i)}\leq 1} \quad g(\alpha) := \frac{-1}{2C}\left\|\sum_i \alpha^{(i)}y^{(i)}\phi(x^{(i)})\right\|_2^2 + \sum_i \alpha^{(i)} \tag{16}$$

  - Prediction with dual variables:

$$\mathbf{w}^T\phi(x) = \frac{1}{C}\sum_i \alpha^{(i)}y^{(i)}\phi^T(x^{(i)})\phi(x) \tag{17}$$

One observes that prediction using both SVM and logistic regression and their corresponding optimization problem have many inner products $\phi^T(x^{(i)})\phi(x^{(j)})$ between the different samples. To find these inner products, we explicitly transform the input data x to a mapped feature space $\phi(x)$, and then we find the inner products. When the kernel trick is implemented, we do not need to explicitly construct $\phi(x)$, but rather directly calculate the inner products [5]. Such an operation is cheaper than operations requires explicit computations of the feature space ϕ. The kernel trick refers to replacing the inner products $\phi^T(x^{(i)})\phi(x^{(j)})$ with a kernel function $\kappa(x^{(i)}, x^{(j)})$. Both the inner product $\phi^T(x^{(i)})\phi(x^{(j)})$ and kernel function $\kappa(x^{(i)}, x^{(j)})$ return a scalar $\in \mathcal{R}$. However, we need to ensure that κ is a valid kernel. In other words, it corresponds to an inner product in some feature space.

Now let's consider an example of kernel functions. Consider a kernel function κ (defined below) which is dependent on two variables, x and z. Both x and z are two-dimensional variables.

$$\begin{aligned}
\kappa(x,z) &= (x^Tz)^2 \\
\kappa(x,z) &= x_1^2z_1^2 + 2x_1z_1x_2z_2 + x_2^2 + z_2^2 \\
\kappa(x,z) &= [x_1^2 \quad \sqrt{2}x_1x_2 \quad x_2^2][z_1^2 \quad \sqrt{2}z_1z_2 \quad z_2^2]^T \\
\kappa(x,z) &= \phi(x)^T\phi(z)
\end{aligned}$$

where $\phi(x)$ and $\phi(z)$ are defined as $[x_1^2 \quad \sqrt{2}x_1x_2 \quad x_2^2]^T$ and $[z_1^2 \quad \sqrt{2}z_1z_2 \quad z_2^2]^T$, respectively. We can check the validity of the kernel function without explicitly constructing the feature vectors using two strategies:

1. Gram matrix $(K)_{i,j} = \kappa(x^{(i)}, x^{(j)})$ is positive definite for $\forall x^{(i)}, x^{(j)}$
   Recall: Gram matrix G is the matrix of all possible inner products [3]. In other words, a Gram matrix of vectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ is a matrix G which is defined as:

$$G = \langle \mathbf{x}_{(i)}, \mathbf{x}_{(j)} \rangle \quad \forall i, j. \tag{18}$$

   From linear algebra, G is positive-definite if and only if the vectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ are linearly-independent.

2. Compose/decompose kernel from/into kernels known to be valid.

Below are three examples of valid kernel functions:

- Linear kernel:

$$\kappa(x^{(i)}, x^{(j)}) = x^{(i)T}x^{(j)}$$

- Squared exponential (Gaussian) kernel:

$$\kappa(x^{(i)}, x^{(j)}) = exp\bigg(-\frac{1}{2}(x^{(i)} - x^{(j)})^T \Sigma^{-1}(x^{(i)} - x^{(j)})\bigg)$$

  where Σ^{-1} is a positive definite matrix.

- Sigmoid kernel:

$$\kappa(x^{(i)}, x^{(j)}) = tanh(a.x^{(i)T}x^{(j)} + b)$$

To construct new kernels from existing ones, we use the operations listed below:

- Positive scaling:

$$\kappa(x^{(i)}, x^{(j)}) = C\kappa_1(x^{(i)}, x^{(j)})$$

  where $C > 0$.

- Exponentiation:

$$\kappa(x^{(i)}, x^{(j)}) = exp\bigg(\kappa_1(x^{(i)}, x^{(j)})\bigg)$$

- Addition:

$$\kappa(x^{(i)}, x^{(j)}) = \kappa_1(x^{(i)}, x^{(j)}) + \kappa_2(x^{(i)}, x^{(j)})$$

- Multiplication with function:

$$\kappa(x^{(i)}, x^{(j)}) = f(x^{(i)})\kappa_1(x^{(i)}, x^{(j)})f(x^{(j)})$$

- Multiplication:

$$\kappa(x^{(i)}, x^{(j)}) = \kappa_1(x^{(i)}, x^{(j)})\kappa_2(x^{(i)}, x^{(j)})$$

References

[1] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[2] C. Forbes, M. Evans, N. Hastings, and B. Peacock. *Statistical distributions*. John Wiley & Sons, 2011.

[3] R. A. Horn and C. R. Johnson. Matrix analysis, 2nd, 2013.

[4] K. P. Murphy. *Machine learning: a probabilistic perspective*. Cambridge, MA, 2012.

[5] S. Theodoridis, K. Koutroumbas, et al. Pattern recognition. *IEEE Transactions on Neural Networks*, 19(2):376, 2008.