Machine Learning - FMAN45
Assignment 2, spring 2019

Support Vector Machines SVM
Dimensionality reduction PCA
Clustering with K-means

Hicham Mohamad - hi8826mo-s
hsmo@kth.se

October 20, 2019

1 Introduction

In this assignment, we are going to work on the classification machine learning algorithm, `Support Vector Machines SVM`. The first part of the assignment, we consider the theoretical aspects of SVM by studying the related optimization problem. In the second part, we deal with the experimental tasks, that involves the `MNIST` image dataset of handwritten digits, consisting of data-target pairs with images $\mathbf{X}_i \in [0,1]^{28\times 28}$ and targets $t_i \in \{0,1,\cdots,9\}$. In this part, we use first a dimensionality reduction technique PCA and then construct a classifier using both K-means clustering and SVM approaches.

2 Objectives

The objectives of this assignment are to:

1. solve the optimization problem for the hard margin SVM.
2. solve the Lagrangian dual problem for the soft margin SVM.
3. compute the dimensionality reduction PCA on MNIST.
4. implement K-means clustering and use it for classification.
5. implement a supervised classifier SVM.

3 Background - Support Vector Machines, SVM

Machine learning algorithms can be classified along two main lines: `supervised` and `unsupervised` classification. The most common supervised learning tasks are **regression** (predicting values) an **classification** (predicting classes). In supervised learning the agent observes some example input–output pairs and learns a function that maps from input to output. In supervised learning, given a `training set` of N example input-output pairs

$$(x_1, y_1), (x_2, y_2), \cdots (x_N, y_N)$$

where each y_i was generated by an unknown function $y = f(x)$, we need to discover a function h that approximates the true function f. Here x and y can be any value and they need not to be numbers. The function h is a `hypothesis`. When the output y is one of a finite set of values, the learning problem is called classification, and is called Boolean or binary classification if there are only two values. When y is a number, such as tomorrow's temperature, the learning problem is called regression.

In our tasks, we will restrict our attention to learning machines that separate the positive and negative examples using a linear function with parameters **w** and b

$$g(x) = \mathbf{w}^T \mathbf{x} + b$$

by considering the Support Vector Machines (SVM) for binary classification, as illustrated in Figures 1, which aims to find the parameters $\mathbf{w} \in \mathbb{R}^d$ and a **bias term** b such that $\mathbf{w}^T \mathbf{x}_i + b \geq 1$ if the examples $\mathbf{x}_i \in \mathbb{R}^d$ belongs to the positive class ($y_i = +1$), and $\mathbf{w}^T \mathbf{x}_i + b \leq 1$ when $\mathbf{x}_i$ belongs to the negative class ($y_i = -1$). Traditionally SVMs use the convention that class labels are +1 and -1, instead of the +1 and 0. Also, where we put the **intercept** into the weight vector w (and a corresponding dummy 1 value into xj,0), SVMs do not do that; they keep the intercept as a separate parameter, b. With that in mind, the **separator** is defined as the set of points $\{\mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = 0\}$.

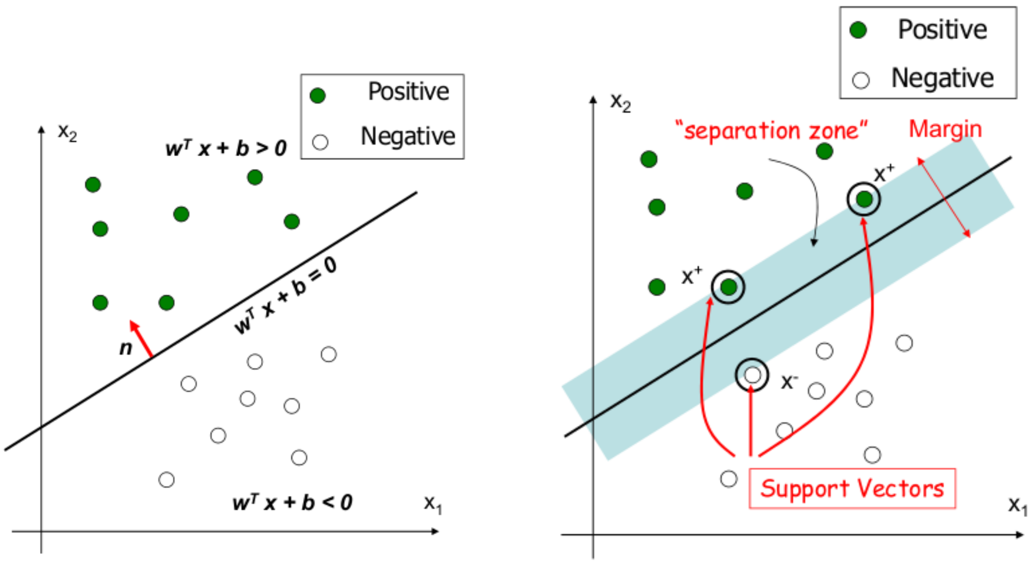


Figure 1: Support vector machine classification: a) Linear discriminant function: two classes of points (green and white circles) and three candidate linear separators. b) The maximum margin separator (heavy line), is at the midpoint of the ***margin*** *(area in blu). The* ***support vectors*** *(points with large circles) are the examples closest to the separator.*

Linear Separability

Now, it is useful to remember that a **decision boundary** is a line (or a surface, in higher dimensions) that separates two classes. In our Figures 1, the decision boundary is a straight line. A linear decision boundary is called a linear separator and data that admit such a separator are called **linearly separable**.

Kernelization

What if the examples are not linearly separable? In general, if data are mapped into a space of sufficiently high dimension, then they will almost always be linearly separable. Actually, if we look at a set of points from enough directions, we'll find a way to make them line up. For instance, four dimensions suffice for linearly separating a circle anywhere in the plane (not just at the origin), and five dimensions suffice to linearly separate any ellipse. In general (with some special cases excepted) if we have N data points then they will always be separable in spaces of N-1 dimensions or more.

In this way, we would not usually expect to find a linear separator in the input space **x**, but we can find linear separators in the high-dimensional **feature space** $\phi(\mathbf{x})$, i.e. simply by replacing **x** by $\phi(\mathbf{x})$, as illustrated in Figure 2.

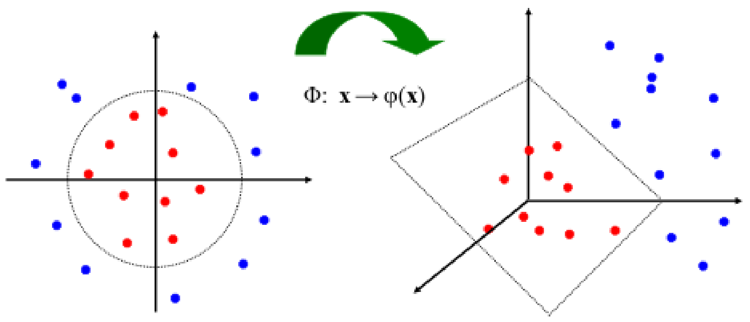


Figure 2: Non-linear SVMs - Feature Space: the original input space can be mapped to some higher-dimensional feature space where the training set is separable. The same data after mapping into a three-dimensional input space. The circular decision boundary on the left becomes a linear decision boundary in three dimensions on the right.

Properties of the SVM

The support vector machine or SVM framework is one of the most popular approach for "off-the-shelf" supervised learning. An important property of support vector machines is that the determination of the model parameters corresponds to a **convex optimization** problem, and so any local solution is also a global optimum. There are other three properties that make SVMs attractive:

1. SVMs construct a **maximum margin separator** — a decision boundary with the largest possible distance to example points. This helps them generalize well.
2. SVMs create a **linear separating hyperplane**, but they have the ability to embed the data into a higher-dimensional space, using the so-called **kernel trick**. Often, data that are not linearly separable in the original input space are easily separable in the higher-dimensional space. The high-dimensional linear separator is actually nonlinear in the original space. This means the hypothesis space is greatly expanded over methods that use strictly linear representations.
3. SVMs are a **nonparametric method** — they retain training examples and potentially need to store them all. On the other hand, in practice they often end up retaining only a small fraction of the number of examples — sometimes as few as a small constant times the number of dimensions. Thus SVMs combine the advantages of nonparametric and parametric models: *they have the flexibility to represent complex functions, but they are resistant to overfitting.*

The key insight of SVMs is that some examples are more important than others, and that paying attention to them can lead to better **generalization**. SVMs address this issue: Instead of minimizing expected *empirical loss* on the training data, SVMs attempt to minimize expected *generalization loss.* we minimize generalization loss by choosing the separator that is farthest away from the examples we have seen so far. As mentioned above, we call this separator, shown in Figure 1b the maximum margin separator.

4 Solving a nonlinear kernel SVM with hard constraints

The size of the margin is inverse proportional to the length of $\|\mathbf{w}\|$. The margin should be as large as possible, which corresponds to minimizing $\|\mathbf{w}\|$, or rather $\frac{1}{2}\|\mathbf{w}\|^2$ for computational reasons. In this section, we need to examine the linear hard margin SVM, which is defined as the solution to the minimization problem:

$$\text{Hard margin SVM} \begin{cases} \min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 \\ \\ \text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \end{cases} \tag{1}$$

4.1 Task T1 - Computing the kernel matrix K

In this task, we consider the one-dimensional training set with positive and negative examples, $(y_i = +1)$ and $(y_i = -1)$ respectively, as illustrated in Table 1. The task is to compute the kernel matrix $\mathbf{K}$ using this dataset.

i	1	2	3	4
x_i	-2	-1	1	2
y_i	+1	-1	-1	+1

Table 1: The dataset given by the one-dimensional binary classification problem with classes +1 and -1.

Clearly, there is no linear separator for this problem because the dataset is not linearly separable. Thus, to solve this problem we need to perform *kernelization*, i.e. we re-express the input data by mapping the input vector $\mathbf{x}$ to a new vector of feature values

$$\phi(x) = \begin{pmatrix} x \\ x^2 \end{pmatrix} \tag{2}$$

In this way, the data are mapped into a two-dimensional input space and they becomes linearly separable. The corresponding **kernel** is given by $k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ and the computed **kernel matrix** by

$$\mathbf{K} = [(x_i, x_j)]_{1 \le i,j \le 4} = \begin{pmatrix} 20 & 6 & 2 & 12 \\ 6 & 2 & 0 & 2 \\ 2 & 0 & 2 & 6 \\ 12 & 2 & 6 & 20 \end{pmatrix} \tag{3}$$

4.2 Task T2 - Solving the maximization problem for α

In this task, we need to solve the given Lagrangian dual problem for the hard margin SVM in 4. We could actually search the space of $\mathbf{w}$ and b with **gradient descent** to solve the minimization problem in 1 and consequently find the parameters that maximize the margin while correctly classifying all the examples. However, it turns out there is another approach to solving this problem by considering an alternative representation called the **dual representation**, in which the optimal solution is found by solving

$$\text{Dual Hard margin SVM} \begin{cases} \max_{\alpha_1, \cdots, \alpha_4} \quad \sum_{i=1}^4 \alpha_i - \frac{1}{2} \sum_{i=1}^4 \sum_{j=1}^4 \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ \\ \text{subject to} \quad \alpha_i \ge 0 \quad \text{and} \quad \sum_{i=1}^4 y_i \alpha_i = 0 \end{cases} \tag{4}$$

This takes the form of a **quadratic programming** problem in which we optimize a quadratic function of α subject to a set of inequality constraints. Here the kernel function is defined by $k(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$, where $\phi(x)$ corresponds to the **feature map** defined above. In order to solve this optimization problem for α, we consider the objective function (the lagrangian)

$$L = \sum_{i=1}^4 \alpha_i - \frac{1}{2} \sum_{i=1}^4 \sum_{j=1}^4 \alpha_i \alpha_j y_i y_j k(x_i, x_j) = 4\alpha - \frac{1}{2}\alpha^2 \sum_{i=1}^4 \sum_{j=1}^4 y_i y_j \phi(x_i)^T \phi(x_j)$$

where we take in consideration the fact that the solution satisfies $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4$, then, we need to set the derivative of the lagrangian with respect to α to zero, giving

$$\begin{aligned} \frac{\partial L}{\partial \alpha} &= 4 - \alpha \sum_{i=1}^4 \sum_{j=1}^4 y_i y_j \phi(x_i)^T \phi(x_j) \\ &= 4 - \alpha \sum_{i=1}^4 \sum_{j=1}^4 y_i y_j \mathbf{K}(i,j) = 0 \implies \alpha = \frac{4}{\sum_{i=1}^4 \sum_{j=1}^4 y_i y_j \mathbf{K}(i,j)} \end{aligned} \tag{5}$$

By implementing the obtained expression of α in Matlab and running it on the dataset using the computed $\mathbf{K}$ matrix in 3 or the kernel in 2, we get

$$\alpha = 1/9 = 0.11111$$

Properties of the Lagrangian dual representation

There are some important properties of the Lagrangian dual representation in Equation 4:

1. Once we have found the vector α we can get back to $\mathbf{w}$ with the equation $\mathbf{w} = \sum_j \alpha_j \mathbf{x}_j$, or we can stay in the dual representation.
2. The expression is convex; it has a single global maximum that can be found efficiently.
3. The data enter the expression only in the form of **dot products** of pairs of points. This property is also true of the equation for the **separator** itself.
4. A final important property is that the weights α_j associated with each data point are zero except for the **support vectors** — *the points closest to the separator. They are called "support" vectors because they "hold up" the separating plane.* Because there are usually many fewer support vectors than examples, SVMs gain some of the advantages of **parametric** models.

4.3 Task T3 - Reduction of the classifier function

Once the optimal α_j have been calculated, we can express the data in the form of dot products of pairs of points. Using the properties mentioned above, we can see that for any support vector x_s, we have

$$y_s\left(\mathbf{w}^T\mathbf{x}_s + b\right) = 1 \implies y_s\left(\sum_{j\in S}\alpha_j y_j k(x_j, x_s) + b\right) = y_s\left(\sum_{j=1}^{4}\alpha_j y_j k(x_j, x_s) + b\right) = 1 \tag{6}$$

where s denotes the set of indices of the support vectors. In our task, we notice that all the four data points in Figure 3a can be considered support vectors. To get the second term on the right in 6, we use one of the KKT conditions which gives

$$\mathbf{w} = \sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i \tag{7}$$

In addition, we can also see that the equation for the classifier can be given by

$$g(x) = \sum_{j=1}^{4}\alpha_j y_j k(x_j, x) + b \tag{8}$$

For the data-target pairs in Table 1, we need to reduce now the **classifier function** in 8 to the simplest possible form, leading to a simple polynomial in x. For solving this task, we can solve the equation in 6 for b using an arbitrarily chosen support vector x_s:

$$b = \frac{1}{y_s} - \sum_{j=1}^{4}\alpha_j y_j k(x_j, x_s) = \frac{-5}{3} = -1.666 \tag{9}$$

That's why by plugging in the value of $\alpha = 1/9$ obtained before and anyone of the data points we get the same result for b. Now, we can reduce the expression of the classifier function in 8, by plugging in the values of α, b

$$\begin{aligned}
g(x) &= \alpha\sum_{j=1}^{4} y_j(x_j x + x_j^2 x^2) + b \\
&= \alpha[y_1(x_1 x + x_1^2 x^2) + y_2(x_2 x + x_2^2 x^2) + y_3(x_3 x + x_3^2 x^2) + y_4(x_4 x + x_4^2 x^2)] + b \\
&= \alpha[(-2x + 4x^2) - (-x + x^2) - (x + x^2) + (2x + 4x^2)] + b \\
&= \alpha(6x^2) + b = \frac{1}{9}(6x^2) - \frac{5}{3} = \frac{2x^2 - 5}{3}
\end{aligned} \tag{10}$$

i	1	2	3	4
x_i	-2	-1	1	2
y_i	$+1$	-1	-1	$+1$

(a)

i	1	2	3	4	5	6	7
x_i	-3	-2	-1	0	1	2	4
y_i	$+1$	$+1$	-1	-1	-1	$+1$	$+1$

(b)

Figure 3: a) The binary classification of the data in Table 1, where we can see the decision boundary in red line. The support vectors are the examples closest to the separators, i.e. all the points in the dataset. b) The binary classification of the data in Table 2 where the support vectors are the same examples as those in Table 1, the points which are closest to the red separators.

4.4 Task T4 - Another binary classification problem

In this task, we consider another binary classification problem with the dataset shown in Table 2. With the same kernel $k(x, y)$ as above, we need to find the solution g(x) of the nonlinear kernel SVM with hard constraint on this dataset.

i	1	2	3	4	5	6	7
x_i	-3	-2	-1	0	1	2	4
y_i	+1	+1	-1	-1	-1	+1	+1

Table 2: Another dataset given by the one-dimensional binary classification problem with classes +1 and -1.

In order to solve this task, it is enough to look at Figure 3b and notice that we have the support vectors at the data points $x = \{-2, -1, 1, 2\}$ which correspond to the same support vectors in Task 3. At the other data points the weights α should be zero. In this way, the solution g(x) on this dataset is also given by

$$g(x) = \alpha \sum_{j=1}^{4} y_j (x_j x + x_j^2 x^2) + b = \frac{2x^2 - 5}{3} \tag{11}$$

5 The Lagrangian dual of the soft margin SVM

In the case of inherently noisy data, we may not want a linear separator in some high-dimensional space. Rather, we'd like a decision surface in a lower-dimensional space that does not cleanly separate the classes, but reflects the reality of the noisy data. That is possible with the **soft margin** classifier, which *allows examples to fall on the wrong side of the decision boundary*, but assigns them a **penalty** proportional to the distance required to move them back on the correct side. To do this, we introduce **slack variables**, $\xi_i \geq 0$, with one slack variable for each training data point.

We therefore need a way to modify the support vector machine so as to allow some of the training points to be misclassified. This is sometimes described as relaxing the hard margin constraint to give a soft margin. Thus, in the soft-margin SVM, we allow errors ξ_i in this classification tasks for each data point x_i. We denote the collection of all ξ_i by ξ. The **primal** formulation of the linear soft margin classifier is given by

$$\text{Soft margin SVM} \begin{cases} \min_{\mathbf{w},b,\xi} & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i \\ & \\ \text{subject to} & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{cases} \tag{12}$$

where the parameter $C > 0$ controls the trade-off between the slack variable penalty and the margin. Because any point that is misclassified has $\xi_i > 1$, it follows that $\sum_i \xi_i$ is an upper bound on the number of misclassified points. The parameter C is therefore analogous to (the inverse of) a **regularization coefficient** because it controls the trade-off between minimizing training errors and controlling model complexity. Small C allows constraints to be easily ignored, i.e. *large margin.* Large C makes constraints hard to ignore, i.e. *narrow margin.* In the limit $C \to \infty$, we will recover the earlier support vector machine for separable data, i.e. *hard margin.*

5.1 Task 5 - Dual representation of the soft margin problem

Our goal is now to maximize the margin while softly penalizing points that lie on the wrong side of the margin boundary. We therefore need to minimize the objective function

$$f(\mathbf{w}, b, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i \tag{13}$$

subject to the constraints in the second line in 27 together with $\xi \geq 0$. In order to solve this constrained optimization problem, we introduce Lagrange multipliers $\alpha, \beta \geq 0$, with one multiplier for each of the constraints in 27, giving the Lagrangian function

$$\begin{aligned} L(\mathbf{w}, b, \xi_i; \alpha_i, \beta_i) &= f(\mathbf{w}, b, \xi) - \sum_{i=1}^{n} \alpha_i \left[y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i \right] - \sum_{i=1}^{n} \beta_i \xi_i \\ &= \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i \left[y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i \right] - \sum_{i=1}^{n} \beta_i \xi_i \end{aligned} \tag{14}$$

This Lagrangian need to be minimized with respect to $\mathbf{w}, b, \xi_i$ and maximized w.r.t. α_i, β_i

$$\min_{\mathbf{w},b,\xi} \quad \max_{\alpha,\beta} L(\mathbf{w}, b, \xi_i; \alpha_i, \beta_i) \tag{15}$$

By setting the derivatives of $L(\mathbf{w}, b, \xi; \alpha_i, \beta_i)$ with respect to $\mathbf{w}$, b and ξ_i to zero, we can obtain the following **KKT conditions** (Karush-Kuhn-Tucker)

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \implies \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0 \implies \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{16}$$

$$\frac{\partial L}{\partial b} = 0 \implies \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{17}$$

$$\frac{\partial L}{\partial \xi_i} = 0 \implies C - \alpha_i - \beta_i = 0 \implies \alpha_i = C - \beta_i \tag{18}$$

In addition, the KKT conditions include the following **complementary slackness conditions** CSC

$$\alpha_i \left[y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i \right] = 0 \tag{19}$$

$$\beta_i \xi_i = 0 \tag{20}$$

From the KKT condition in 19, we can see that only support vectors have $\alpha_i \neq 0$ and from KKT condition in 16 the solution has the form

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = \sum_{i \in SV} \alpha_i y_i \mathbf{x}_i \tag{21}$$

Now we rewrite the Lagrangian function by substituting $\|\mathbf{w}\|^2 = \mathbf{w}^T\mathbf{w}$ and the obtained KKT conditions

$$\begin{aligned} L(\mathbf{w}, b, \xi_i; \alpha_i, \beta_i) &= \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i \left[y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i \right] - \sum_{i=1}^{n} \beta_i \xi_i \\ &= \frac{1}{2}(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i)^T(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j) + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i \left[y_i((\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_i)^T \mathbf{x}_i + b) - 1 + \xi_i \right] - \sum_{i=1}^{n} \beta_i \xi_i \\ &= \frac{1}{2}(\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j) + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j - b\sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \alpha_i \xi_i - \sum_{i=1}^{n} \beta_i \xi_i \\ &= -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i \end{aligned} \tag{22}$$

where $\mathbf{w}$ and b and ξ_i become eliminated from $L(\mathbf{w}, b, \xi_i; \alpha_i, \beta_i)$ using these conditions then gives the **dual representation** of the maximum margin problem in which we maximize

$$\text{Dual soft margin SVM} \begin{cases} \max_{\alpha_1, \cdots, \alpha_4} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j \ \mathbf{x}_i^T \cdot \mathbf{x}_j \\ \\ \text{subject to} \quad 0 \leq \alpha_i \leq C \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \end{cases} \tag{23}$$

This takes the form of a **convex quadratic programming** problem in which we optimize a quadratic function of α subject to a set of inequality constraints. Such problem has no extra local minima. Using a QP solver, gives us the uniquely best α_i .

5.2 Task 6 - Using the complementary slackness

Now we need to use complementary slackness (of the KKT conditions) to show that support vectors with $y_i(\mathbf{w}^T\mathbf{x}_i + b) < 1$ have coefficient $\alpha_i = C$.

As we have seen in 19 and 20, the complementary slackness conditions are give by

$$\alpha_i \left[y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i\right] = 0 \quad \text{and} \quad \beta_i\xi_i = 0 \tag{24}$$

From this we can write that $\alpha_i = 0$ or $y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i = 0$. But we know that for support vectors $\alpha_i \neq 0$. This gives that $y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1 - \xi_i < 1$ only if $\xi_i > 0$. This with 20, imply that $\beta_i = 0$.

Finally, using the obtained KKT condition in 18, we can write

$$\alpha_i = C - \beta_i \implies \alpha_i = C \tag{25}$$

6 Dimensionality reduction on MNIST using PCA

In the following experimental tasks of this assignment, we work on the `MNIST` image dataset of handwritten digits, consisting of data-target pairs with images $\mathbf{X}_i \in [0,1]^{28\times 28}$ and targets $t_i \in \{0, 1, \cdots, 9\}$. We consider only a part of the data set, specifically only images with targets $t_i \in \{0, 1\}$. The dataset `MNIST_01.mat` contains 12665 training data-target pairs {`train_data, train_labels`} and 2115 test data-target pairs {`test_data, test_labels` }, where for both sets the images have been stacked column-wise, i.e., $\mathbf{x}_i = vec(\mathbf{X}_i) \in \mathbb{R}^{28\times 28}$.

PCA - Finding maximum variance directions

The MNIST images are of quite low resolution, but the dimensionality is still quite large, i.e., $D = 28^2 = 784$. *One way of visualizing a collection of high-dimensional data examples is to use a dimensionality reduction technique.* **Principal component analysis** (PCA) is a method for reducing the dimensionality of a dataset, while retaining as much of the data's variability as possible.

Principal Component Analysis (PCA) is about finding ***directions*** *in a data set where we have* ***large variance***. Suppose we have data $\mathbf{D} = \{\mathbf{x}_i, t_i\}$ and $\mathbf{x}_i \in \mathbb{R}^P$. **PCA algorithm** can be summarized by the following steps:

1. $\mathbf{X} \leftarrow$ create $D \times N$ data matrix, with one column vector x_i per data point
2. $\mathbf{X} \leftarrow$ subtract mean μ from each column vector $\mathbf{x}_i$ in $\mathbf{X}$.
3. $\Sigma \leftarrow$ covariance matrix of $\mathbf{X}$
4. Find eigenvectors and eigenvalues of Σ
5. PC's $\leftarrow$ the M eigenvectors with largest eigenvalues

These eigenvectors of the covariance matrix have the property that they point along the major directions of variation in the data. Thus, as defined in Wikipedia, *PCA method is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables.* In other words, it is a technique that reduces a dataset to its most important components by removing correlated characteristics, i.e. reducing the data to its "essence."

SVD - Problems and limitations using PCA algorithm

What if we have very large dimensional data? For example, when we have images of dimension $d \geq 10^4$ we get covariance matrix Σ of size $d^2 = 10^8$. In this case, it is nice to use *Singular Value Decomposition* (SVD) which is an efficient algorithm and available in Matlab.

Thus, in our task we can compute the PCA from the **Singular Value Decomposition** (SVD) on $\mathbf{X}$, which is a matrix of N zero-mean data examples of dimension D, i.e. $\mathbf{X} \in \mathbb{R}^{D\times N}$. SVD breaks down the data set into eigenvectors and eigenvalues, using the following equation:

$$\mathbf{X} = \mathbf{U} \;\; \mathbf{S} \;\; \mathbf{V}^T \tag{26}$$

where $\mathbf{U} \in \mathbb{R}^{D \times D}$, $\mathbf{S} \in \mathbb{R}^{D \times N}$, $\mathbf{V} \in \mathbb{R}^{N \times N}$.

$\mathbf{U}$, $\mathbf{V}$ are orthogonal matrices, and $\mathbf{S}$ is a diagonal matrix where singular values are positive, and sorted in decreasing order. $\mathbf{U}$, the *left singular vectors* of $\mathbf{X}$, contains the *eigenvectors* of the covariance $\Sigma = \mathbf{X}\ \mathbf{X}^T$, whereas $\mathbf{S}$ contains the square roots of the **eigenvalues** of Σ. $\mathbf{V}$ is the right singular vectors of $\mathbf{X}$ and contains the eigenvectors of $\mathbf{X}^T\mathbf{X}$.

The dimensionality reduction of $\mathbf{X}$ to dimension d is then the projection of $\mathbf{X}$ onto the d **left singular vectors** with the largest absolute singular values.

6.1 Task E1 - Computing a linear PCA using SVD

In this task, we need to compute a linear PCA and vizualize the whole training data in d = 2 dimensions. For fullfilling the **necessary condition** to apply PCA, we should substract the mean from the data. The output results is displayed in Figure 4.

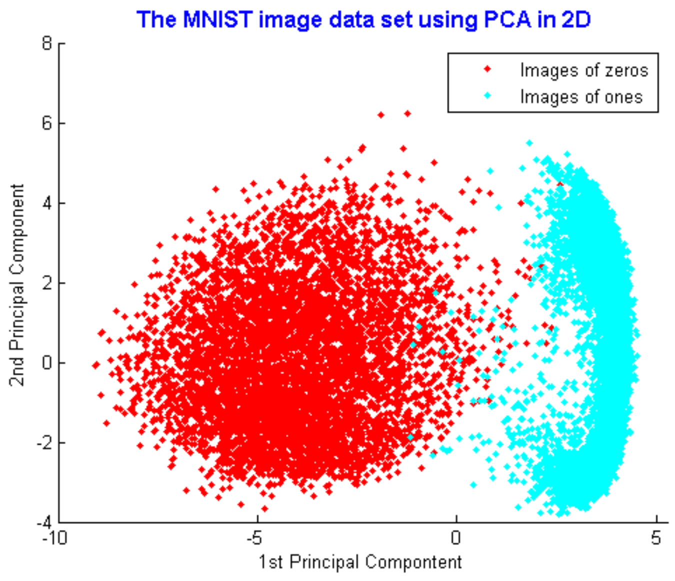


Figure 4: The resulting plot for the MNIST images 0,1 using PCA method for reducing the dimensionality of the dataset. The data shows the ***largest variability*** *along the first principal component axis. This is the largest possible variance among all possible choices of the first axis. The variability along the second principal component axis is the largest among all possible remaining choices of the second axis.*

7 Clustering of unsupervised data using K-means

In this section, we shouldl cluster the MNIST images without using their target values. *Kmeans clustering is an unsupervised approach for computing K many clusters by iteratively determining the so-called centroids of each cluster and assigning each sample to a cluster.* The **centroid** of a cluster is defined as the mean location of all samples within the cluster. In the *assignment step*, each sample gets assigned to the closest centroid. Algorithm 1, in Figure 5, summarizes the K-means clustering approach.

7.1 Task E2 - K-means clustering algorithm implementation

In this task, the function `K_means_clustering()` is implemented according to the algorithm illustrated in Figure 5. The implementation is done by completing and implementing the following functions:

1. `f_xdist(x, C)`, computes the euclidean distance between a single example and the K centroids.
2. `f_cdist(C, C̃)`, computes the euclidean distance between two cluster centroids, such that the distance is exactly zero when the pairs of centroids agree.
3. `step_assign_cluster()`, the first step of the K-means clustering algorithm which assigns each sample to one of the K clusters.

4. `step_compute_mean()`, the second step of the K-means clustering algorithm which assigns new centroids by computing the means of the newly assigned clusters, outputting both the new centroids and a measure of the distance which the centroids have moved. The latter is used as a stopping criterion; the algorithm will stop when the centroids have changed less than a specificed threshold.

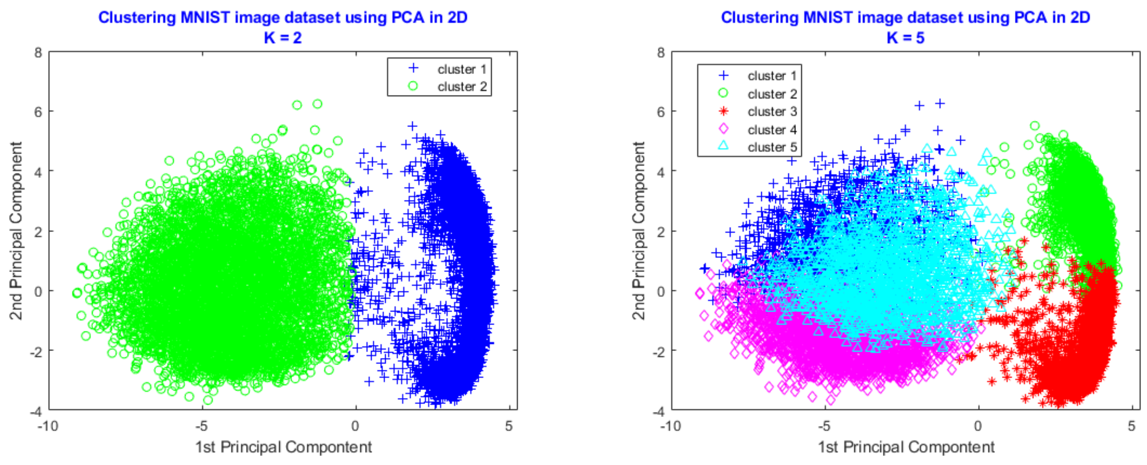
Algorithm 1 K-means clustering

```
Select number of clusters $K$, convergence threshold $\epsilon_{\text{tol}} > 0$,
and maximum iterations $j_{\max}$.
Initialize $K$ centroids $C^{(0)} = \left[\mathbf{c}_1^{(0)}, \ldots, \mathbf{c}_K^{(0)}\right]$ randomly.
for $j = 1, \ldots, j_{\max}$ do
    Step 1: Assign examples to clusters:
    $d^i = f_{\text{xdist}}(\mathbf{x}_i, \mathbf{C}^{(j-1)}), i = 1, \ldots, N$
    $y_i = \arg\min_k d_k^i, i = 1, \ldots, N$
    Step 2: Assign new cluster centroids:
    $N_k = \sum_{i=1}^{N} 1\{y_i = k\}$, k = 1,...,K
    $\mathbf{c}_k = N_k^{-1} \sum_{i=1}^{N} 1\{y_i = k\}\mathbf{x}_i$, k = 1,...,K
    Check convergence:
    if $f_{\text{cdist}}(\mathbf{C}^{(j)}, \mathbf{C}^{(j-1)}) < \epsilon_{\text{tol}}$ then
        return $\mathbf{y}^{(j)}, \mathbf{C}^{(j)}$
    end if
end for
return $\mathbf{y}^{(j_{\max})}, \mathbf{C}^{(j_{\max})}$
```

Figure 5: The algorithm that outlines the implementation of a K-means clustering approach

Results and discussion

After implementing and running the algorithm on the training data, a plot illustrating the results in two dimensions for K = 2 and K = 5 are shown in Figure 6. Looking at Figure 6b, it seems that the clusters overlap. This can be explained because we have 5 clusters visualized in only 2 dimensions.



(a) (b)

Figure 6: The output results after running the algoritm on the training data. a)K-means clustering for $K = 2$. b) K-means clustering for $K = 5$.

7.2 Task E3 - Displaying the centroids as images

We could now check only a few samples from each cluster to assign a suitable label to the centroid. Having assigned a label to a centroid, we assign the same label to all samples in the corresponding cluster.

Alternatively, we can plot the centroid image and assign a suitable class manually. These approaches are advantageous if labeling of all samples individually is expensive. Here, we have the *privileged situation* to know target labels for all samples, so we may also assign the label to a centroid that is the most frequent in the cluster. Assigning the same label to all samples in the cluster will lead to some misclassifications. By summing the number of misclassifications in each cluster, a missclassification rate may be calculated for the dataset.

In this task, we need to display the K centroids as images. First we should stack back the data $\mathbf{x}_i$ into $\mathbf{X}_i \in \mathbb{R}^{28\times 28}$ using the Matlab function `reshape()`. The output results are illlustrated in Figure 7.

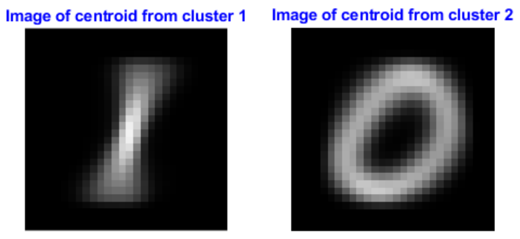
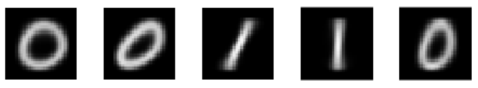


Image of centroid from cluster 1
Image of centroid from cluster 2

(a)



(b)

Figure 7: a) plot illustrating the 2 centroids as images. b) Plot displaying 1×5 *centroids from each cluster .*

7.3 Task E4 - Using K-means clustering for classification

Now you shall use K-means clustering for classification, by completing the following steps:

1. `K_means_classifier`, which assigns to a given example the label of the closest centroid using the distance function calculated in Task E2. Then we assign each cluster centroid the label of which it has the most examples of in the training data.
2. Evaluation: Here we need to evaluate how many **misclassifications** occur for the train and test set `train_data` and `test_data` by comparing the computed cluster labels to the true labels in `train_labels` and `test_labels`.

After implementing the function `K_means_classifier()`, and running it on the train and test set, `train_data` and `train_data`, we obtained the results illustrated Table 3.

Table 3: K-means classification results

Training data	Cluster	# '0'	# '1'	Assigned to class	# misclassified
	1	114	6736	1	114
	⋮	⋮	⋮	⋮	⋮
	2	5809	6	0	6
$N_{\text{train}} = 12665$				Sum misclassified:	120
				Misclassification rate (%):	0.95
Testing data	**Cluster**	**# '0'**	**# '1'**	**Assigned to class**	**# misclassified**
	1	12	1135	1	12
	⋮	⋮	⋮	⋮	⋮
	2	968	0	0	0
$N_{\text{test}} = 2115$				Sum misclassified:	12
				Misclassification rate (%):	0.57

7.4 Task E5 - Lowering the misclassification rate

By trying different values of clusters K, we see that it is possible to lower the misclassification rate further on test data.

8 Classification of MNIST digits using SVM

In the previous section, we implemented a classifier using unsupervised data, by virtue of the data structure itself. In this section, we consider a **supervised classifier**, namely the *support vector machine* (SVM). Thus, the task is to use the *soft-margin SVM* for binary classification, which solves the optimization problem

$$\text{Soft margin SVM} \begin{cases} \min_{\mathbf{w},b,\xi_i} & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ & \\ \text{subject to} & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \quad \forall i \end{cases} \tag{27}$$

where $y_i = 1$ for target $t_i = 1$, and $y_i = -1$ for target $t_i = 0$

8.1 Task E6 - Training a linear SVM classifier

In this task, we need to train a *linear SVM classifier* by using the supervised training data {`training_data, training_labels`}. One can train the *soft-margin SVM* by solving the *Langrangian dual problem* derived in Task T5 by using **numerical methods**. Then we need to evaluate the misclassification rate on both the training and test data.

We can train the binary soft-margin SVM using Matlab's built-in function `fitcsvm(X,T)` on the training data `train_data` and targets `train_labels`, where **X** denotes the $N \times D$ matrix of data examples, and T the target vector of length N.

Then, we can calculate **class predictions** using the built-in Matlab function `predict(model,X)`, where model is the output from `fitcsvm()`, and X is the $N \times D$ dataset matrix to do predictions on. Finally to evaluate the misclassification rate on both the training and test data, the built-in Matlab function `confusionmat(X, predictions)` is used. The results of the misclassifications are illustrated in Table 4.

Table 4: Linear SVM classification results

Training data	Predicted class	True class: # '0'	# '1'
	'0'	5923	0
	'1'	0	6742
$N_{\text{train}} = 12665$		Sum misclassified:	0
		Misclassification rate (%):	0
Testing data	**Predicted class**	**True class: # '0'**	**# '1'**
	'0'	974	1
	'1'	1	1134
$N_{\text{test}} = 2115$		Sum misclassified:	2
		Misclassification rate (%):	9.4563e-04

8.2 Task E7 - Training a non-linear kernel SVM classifier

In this task, you shall look at SVM with kernels, more specifically a **Gaussian kernel**. The kernel SVM solves the SVM optimization problem not on the data directly, but in a **feature space** $\mathbf{z} = \phi(\mathbf{x})$, such that the classifier is linear in $\mathbf{z}$. In practice, this is done using the so called **kernel trick**, such that data is never mapped into this feature space explicitly, but by modifying the dual problem appropriately

Now, we need to use the supervised data again to train a non-linear kernel SVM classifier, using a **Gaussian kernel**. This is done by using the Matlab's built-in function:

```
fitcsvm(X, T, 'KernelFunction', 'gaussian')
```

The Gaussian kernel has a *scaling parameter* σ^2, which in Matlab's SVM estimator is set to $\beta = \sqrt{1/\sigma^2} = 1$ by default, but it may be modified, for some value beta, by specifying

```
fitcsvm(X, T, 'KernelFunction', 'gaussian', 'KernelScale', beta)
```

After training the non-linear kernel SVM classifier using the Gaussian kernel, we get the results as shown in Table 5.

After tuning the value of $\beta = 5$, we could lower the misclassification rate on the test data and get the perfect confusing matrix, i.e. getting misclassification rate = 0.

Table 5: Gaussian kernel SVM classification results

Training data	Predicted class	True class: # '0'	# '1'
	'0'	5923	0
	'1'	0	6742
$N_{\text{train}} = 12665$		Sum misclassified:	0
		Misclassification rate (%):	0
Testing data	**Predicted class**	**True class: # '0'**	**# '1'**
	'0'	980	0
	'1'	388	747
$N_{\text{test}} = 2115$		Sum misclassified:	388
		Misclassification rate (%):	0.1835

References

[1] Bishop, C. M.: Pattern Recognition and Machine Learning. Springer, 2006. Online version available at `https://www.microsoft.com/en-us/research/people/cmbishop/#!prml-book` (March 2019).

[2] Ian Goodfellow, Yoshua Bengio, Aaron Courville. *Deep Learning.* MIT Press, 2016, ISBN: 9780262035613. HTLM version available at `https://www.deeplearningbook.org/` (March 2019).

[3] T. Hastie, R. Tibshirani, J. Friedman: The elements of statistical learning, data mining, inference and prediction, 2nd edition, Springer, 2009, online version available at `https://web.stanford.edu/~hastie/Papers/ESLII.pdf` (March 2019).

[4] Andrew Ng, Stanford course CS229: Machine Learning. Available online at `cs229.stanford.edu/notes/cs229-notes1.pdf`

[5] Roger Grosse, UToronto course CSC 411 Fall 2018: Machine Learning and Data Mining. Available online at `http://www.cs.toronto.edu/~rgrosse/courses/csc411_f18/`