TUM EI70360: Machine Learning and Optimization
Fall 2022
Lecturer: Reinhard Heckel
Teaching Assistant: Jiayu Liu

Problem Set 5
Issued: Monday, Nov. 14, 2022
Due: Tuesday, Nov. 22, 2022

Problem 1 (Linear separability)**.** Suppose we run the hard-margin SVM (without kernels) on n many d-dimensional data points belonging to two classes. The algorithm finds a solution. After which of the following transformations of the data points is the algorithms still guaranteed to find a solution (not necessarily the same solution)?

(a) Centering the data points by offseting each feature by the corresponding sample average.

(b) Transforming the data to a lower dimensional space by removing a feature.

(c) Adding a feature.

(d) Scaling all features with $\alpha > 0$.

Solution 1. Centering (a) and scaling (d) are both transformations after which linearly separable data remains linearly separable. (c) as well, to see this note that if we have a hyperplane that separates data, and we add a feature, then we can simply use the same hyperplane and ignore the additional feature (by setting the respective coordinate $\boldsymbol{\theta}$ of the hyperplane to zero), and thus the data remains linearly separable. (b) is a transformation after which the data is not necessarily linearly separable.

Problem 2 (Logistic regression)**.**

1. Show that if the (loss) function $L(y, t)$ is convex in t for every y, then the function

$$\frac{1}{n}\sum_{i=1}^{n} L(y_i, \langle \mathbf{w}, \mathbf{x}_i \rangle + b) \tag{1}$$

   is convex in $\boldsymbol{\theta} = [b\ \mathbf{w}]^T$.

2. Show that if we choose the loss function as $L(y, t) = \log(1+\exp(-yt))$, then the expression in equation (1) is proportional to the negative log-likelihood for logistic regression. Minimizing the empirical loss associated with a training set as in equation (1) is called empirical loss minimization, or commonly empirical risk minimization (ERM). This problem shows that ERM with the logistic loss is equivalent to logistic regression.

   Note: In the above expression, y is assumed to be -1 or 1. In the logistic regression notes we had $y = 0$ or 1.

Solution 2.

1. We would like to show that $\frac{1}{n}\sum_{i=1}^{n} L(y_i, \langle \mathbf{w}, \mathbf{x}_i \rangle + b)$ is convex in $\boldsymbol{\theta} = [b\ \mathbf{w}]^T$. To do this, we must show that $L(y_i, \langle \mathbf{w}, \mathbf{x}_i \rangle + b)$ is convex for any $\mathbf{x}_i$ and y_i. If this is true, then the sum of these convex functions will also be convex.

   Let us pick two arbitrary parameters $\boldsymbol{\theta}_1 = [b_1\ \mathbf{w}_1]^T$ and $\boldsymbol{\theta}_2 = [b_2\ \mathbf{w}_2]^T$ and fix any $i \in \{1, \ldots, n\}$. These have a corresponding $t_1 = \langle \mathbf{w}_1, \mathbf{x}_i \rangle + b_1$ and $t_2 = \langle \mathbf{w}_2, \mathbf{x}_i \rangle + b_2$. Note that $\boldsymbol{\theta}_1$ is a vector so that $\lambda\boldsymbol{\theta}_1 = [\lambda b_1\ \lambda \mathbf{w}_1]$ (and similarly for $\boldsymbol{\theta}_2$) for any real number λ. Then for a real number $\lambda \in (0, 1)$, we have

$$\begin{aligned}
&L(y_i, \langle \lambda \mathbf{w}_1 + (1-\lambda)\mathbf{w}_2, x_i \rangle + \lambda b_1 + (1-\lambda) b_2) \\
&= L(y_i, \lambda\left(\langle \mathbf{w}_1, x_i \rangle + b_1\right) + (1-\lambda)\left(\langle \mathbf{w}_2, x_i \rangle + b_2\right)) \\
&= L(y_i, \lambda t_1 + (1-\lambda) t_2) \\
&\leq \lambda L(y_i, t_1) + (1-\lambda) L(y_i, t_2) \\
&= \lambda L(y_i, \langle \mathbf{w}_1, x_i \rangle + b_1) + (1-\lambda) L(y_i, \langle \mathbf{w}_2, x_i \rangle + b_2).
\end{aligned}$$

   The inequality here comes from the fact that $L(y, t)$ is convex in t. Hence, we have that $L(y_i, \langle \mathbf{w}, \mathbf{x}_i \rangle + b)$ is convex in $\boldsymbol{\theta}$ for all $i = 1, \ldots, n$. Thus, $\frac{1}{n}\sum_{i=1}^{n} L(y_i, \langle \mathbf{w}, \mathbf{x}_i \rangle + b)$ is convex in $\boldsymbol{\theta}$.

2. Consider the negative log-likelihood for logistic regression, $-\sum_{i=1}^{n} \log(p(y_i|\boldsymbol{\theta}, \mathbf{x}_i))$, with the probability defined as

$$p(y_i|\boldsymbol{\theta}, \mathbf{x}_i) = \begin{cases} g(\boldsymbol{\theta}^T \tilde{\mathbf{x}}_i) & : y_i = 1 \\ 1 - g(\boldsymbol{\theta}^T \tilde{\mathbf{x}}_i) & : y_i = -1 \end{cases} \tag{2}$$

   where $g(t) = 1/(1 + e^{-t})$. Here we use the new variable $\tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$ to simplify notation. However, observe that

$$\begin{aligned}
1 - g(t) &= \frac{1 + e^{-t} - 1}{1 + e^{-t}} \\
&= \frac{e^{-t}}{1 + e^{-t}} \\
&= \frac{1}{1 + e^{t}} \\
&= g(-t)
\end{aligned} \tag{3}$$

   So we can rewrite Equation 2 as

$$p(y_i|\boldsymbol{\theta}, \mathbf{x}_i) = g(y_i \boldsymbol{\theta}^T \tilde{\mathbf{x}}_i). \tag{4}$$

   Then

$$\begin{aligned}
-\sum_{i=1}^{n} \log(p(y_i|\boldsymbol{\theta}, \mathbf{x}_i)) &= \sum_{i=1}^{n} \log(1 + e^{-y_i \boldsymbol{\theta}^T \tilde{\mathbf{x}}_i}) \\
&\propto \frac{1}{n}\sum_{i=1}^{n} \log(1 + e^{-y_i t_i})
\end{aligned} \tag{5}$$

   where $t_i = \boldsymbol{\theta}^T \tilde{\mathbf{x}}_i$.

Problem 3 (Kernelized perceptrons)**.**

1. Consider a binary classification problem with $y \in \{-1, 1\}$. Suppose that g is a subgradient of a convex loss function $L(y, \langle \boldsymbol{\theta}, \mathbf{x} \rangle)$ with respect to $\boldsymbol{\theta}$. If we have

$$g(\boldsymbol{\theta}) = \begin{cases} -y\mathbf{x} & \langle \boldsymbol{\theta}, \mathbf{x} \rangle \, y < 0 \\ 0 & \text{otherwise} \end{cases}$$

then what is $L(y, \langle \boldsymbol{\theta}, \mathbf{x} \rangle)$?

Hint: A vector $g \in \mathbb{R}^d$ is a subgradient of $f : \mathbb{R}^d \to \mathbb{R}$ at $\mathbf{x}$ if for all $\mathbf{z}$,

$$f(\mathbf{z}) \geq f(\mathbf{x}) + g^T(\mathbf{z} - \mathbf{x}).$$

If f is convex and differentiable, then its gradient at $\mathbf{x}$ is a subgradient. But a subgradient can exist even when f is not differentiable at $\mathbf{x}$, as illustrated in Figure 1.

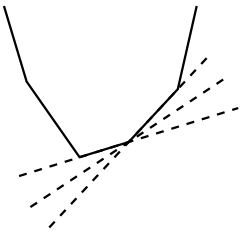

Figure 1: The slope of each dased line is a subgradient.

2. The perceptron uses a hypothesis of the form $h_{\boldsymbol{\theta}}(\mathbf{x}) = \text{sign}(\langle \boldsymbol{\theta}, \mathbf{x} \rangle)$, where $\text{sign}(t) = 1$ if $t \geq 0$ and -1 otherwise. In this problem we will consider a stochastic subgradient method-like implementation of the perceptron algorithm where each update to the parameters $\boldsymbol{\theta}$ is made using only one training example. However, unlike the stochastic subgradient method, the perceptron algorithm will only make one pass through the entire training set, which has n observations. The update rule for this version of the perceptron algorithm is given by

$$\boldsymbol{\theta}^{(i+1)} := \begin{cases} \boldsymbol{\theta}^{(i)} + \alpha y^{(i+1)} \mathbf{x}^{(i+1)} & \text{if } h_{\boldsymbol{\theta}^{(i)}}(\mathbf{x}^{(i+1)}) y^{(i+1)} < 0 \\ \boldsymbol{\theta}^{(i)} & \text{otherwise,} \end{cases}$$

where $\boldsymbol{\theta}^{(i)}$ is the value of the parameters after the algorithm has seen the first i training examples. Here, we have that α is the step size and $i \in \{1, 2, \ldots, n\}$. Prior to seeing any training examples, $\boldsymbol{\theta}^{(0)}$ is initialized to 0.

Let k be a positive definite kernel corresponding to some very high-dimensional feature mapping ϕ. Suppose ϕ is so high-dimensional (say, infinite dimensional) that it is infeasible to ever represent $\phi(\mathbf{x})$ explicitly. We would like to apply the "kernel trick" to the perceptron to make it work in the high-dimensional feature space ϕ, but without ever explicitly computing $\phi(\mathbf{x})$. [Note: You do not have to worry about the intercept term. If you like, think of ϕ as having the property that $\phi_0(\mathbf{x}) = 1$ so that this is taken care of.]

How would you (implicitly) represent the high-dimensional parameter vector $\boldsymbol{\theta}^{(i)}$, including how the initial value of $\boldsymbol{\theta}^{(0)} = 0$ is represented? (note that $\boldsymbol{\theta}^{(i)}$ is now a vector whose dimension is the same as the feature vectors $\phi(\mathbf{x}^{(i)})$)

3. How would you efficiently make a prediction on a new input $\mathbf{x}^{(i+1)}$? In other words, how would you compute $h_{\boldsymbol{\theta}^{(i)}}(\mathbf{x}^{(i+1)}) = \text{sign}\left(\left\langle \boldsymbol{\theta}^{(i)}, \phi(\mathbf{x}^{(i+1)})\right\rangle\right)$, using your representation of $\boldsymbol{\theta}^{(i)}$?

4. How would you modify the update rule given above to perform an update to $\boldsymbol{\theta}$ on a new training sample $(\mathbf{x}^{(i+1)}, y^{(i+1)})$? In other words, how do you compute

$$\boldsymbol{\theta}^{(i+1)} := \begin{cases} \boldsymbol{\theta}^{(i)} + \alpha y^{(i+1)}\phi(\mathbf{x}^{(i+1)}) & \text{if } h_{\boldsymbol{\theta}^{(i)}}(\phi(\mathbf{x}^{(i+1)}))y^{(i+1)} < 0 \\ \boldsymbol{\theta}^{(i)} & \text{otherwise,} \end{cases}$$

feasibly?

Solution 3.

1. Let

$$L(y, \langle \boldsymbol{\theta}, \mathbf{x}\rangle) = \max\{0, -\langle \boldsymbol{\theta}, \mathbf{x}\rangle\, y\}$$

so that L is a convex function of $\boldsymbol{\theta}$. It is easily seen that g is a subgradient of L with respect to $\boldsymbol{\theta}$.

2. We note that given the form of the updates to $\boldsymbol{\theta}^{(i)}$, $\boldsymbol{\theta}^{(i)}$ will always be a linear combination of the training examples, which means that for any $\mathbf{z}$ we can express the inner product between $\boldsymbol{\theta}^{(i)}$ and $\phi(\mathbf{z})$ as

$$\left\langle \boldsymbol{\theta}^{(i)}, \phi(\mathbf{z})\right\rangle = \sum_{j=1}^{i} \beta_j k(\mathbf{x}^{(j)}, \mathbf{z})$$

for some coefficients $\beta_1, \ldots, \beta_i$. So, we can implicitly represent $\boldsymbol{\theta}^{(i)}$ using the kernel functions $k(\mathbf{x}^{(j)}, \cdot)$ and coefficients β_j. For $\boldsymbol{\theta}^{(0)} = 0$, we will simply have that $\left\langle \boldsymbol{\theta}^{(0)}, \phi(\mathbf{z})\right\rangle = 0$, which is covered by the linear combination presented above since the empty sum evaluates to 0.

3. To make a new prediction, we just need to use the linear combination of kernel functions,

$$h_{\boldsymbol{\theta}^{(i)}}(\mathbf{x}^{(i+1)}) = \text{sign}\left(\sum_{j=1}^{i} \beta_j k(\mathbf{x}^{(j)}, \mathbf{x}^{(i+1)})\right),$$

and we are done.

4. Recall that computing $h_{\boldsymbol{\theta}^{(i)}}(\phi(\mathbf{x}^{(i+1)}))y^{(i+1)} < 0$ is feasible because we can use the kernel k. If based on our update rule we need to set $\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} + \alpha y^{(i+1)}\phi(\mathbf{x}^{(i+1)})$, we let $\beta_{i+1} = \alpha y^{(i+1)}$. If we need to set $\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)}$, then using our previous linear combination representation, we simply let $\beta_{i+1} = 0$.

With this update rule, we never have to explicitly compute $\boldsymbol{\theta}^{(i)}$ or $\phi(\mathbf{x}^{(i)})$ for any $i \in \{1, 2, \ldots, n\}$.