[TC1] Notes on Machine Learning

Caio Corro

Université Paris-Saclay, CNRS, Laboratoire Interdisciplinaire des Sciences du Numérique, 91400, Orsay, France

Contents

1 Introduction

The goal of this course is to give tools to understand how to train machine learning models. This problem is an optimization problem therefore we will treat it as such. We will only consider linear models, switching between different tasks: regression, binary classification, multiclass classification and structured prediction.

Many important topics will not be covered, e.g. proof of convergence rates of different optimization techniques. We will also skip some definitions, e.g. lower semi-continuity, and some proofs. The goal is that you get the main idea behind different concepts, so definitions and theorems may be "handwavy" to not burden the material. The main reason is that time is limited and I prefer to focus on things that are simpler to understand and that you can code. However, I hope that overall this course will provide you a strong background to read the literature and explore optimization techniques for machine learning.

2 Linear models

We consider the following scenario: we are given feature values and we want to predict an output. The feature values will be represented as a vector $\boldsymbol{x} \in \mathbb{R}^d$ where d is the number of features. In this course, feature values will always be real numbers. However, it may be that a given feature takes only values 0 and 1 to indicate the presence or absence of the feature. The output will be either a scalar $y \in \mathbb{R}$ (regression, binary classification) or a vector $\boldsymbol{y} \in \mathbb{R}^k$ (multiclass classification and structured prediction). Again, we write that the output is in $\mathbb{R}$ but, for example, for binary classification it will actually be strict subspace of $\mathbb{R}$: $y \in \{-1, 1\}$.

The general framework we will study is machine learning models that compute the output in two steps:

- a parameterized scoring function $s(\boldsymbol{x})$ that maps from the input space to the score space (also called weights or logits). In this course, we will only focus on functions that applies a linear transformation to the input, e.g. $s(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$ where $\boldsymbol{A}$ and $\boldsymbol{b}$ are the parameters of the scoring function.
- a prediction function $q(\boldsymbol{w})$ that maps a value $\boldsymbol{w}$ from the score space to the output space.

Note that, as usual, we will often ignore the bias term in the linear transformation and assume there is a special feature that is always set to one. In this course, we will consider both the hard decision and the likelihood estimation cases. When the prediction function outputs a hard decision, it doesn't give any information about its own uncertainty. This setting can be understood as a probability distribution over the output space where all mass is concentrated in a single. Hence, we won't really make the difference between the two cases and we will actually show that they can be understood under the same framework.

The main problem we will address in this course is the training problem, i.e. how do we fix the parameters of the scoring function. We will focus on the supervised training scenario where we assume we have access to a labeled training set $\{\langle \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)} \rangle\}_{i=1}^n$ of n pairs $\langle \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)} \rangle$. The idea is that we want to select the element of a restricted set of functions that:

- classifies correctly all (or most of) the datapoints in the training data,
- or that maximizes the probability of the training data (probabilistic setting).

It is often important to also regularize the training objective to avoid overfitting on the training data. In our case, the set of function will be a set of parameterized linear models of a given dimension: $\mathcal{F}(d, k) = \{s(\cdot; \boldsymbol{A}) | \boldsymbol{A} \in \mathbb{R}^{k \times d}\}$ where d is the input dimension (the number of features including bias), k the output dimension (1 for regression and binary classification, larger for other problems) and $s(\boldsymbol{x}; \boldsymbol{A}) = \boldsymbol{A}\boldsymbol{x}$. The training problem, that is selecting the best function in $\mathcal{F}(d, k)$ is an optimization problem so we will treat it as such. In particular, we will study the following topics:

- Understand the difference between primal and dual optimization for training a linear model.
- Build sparse models, i.e. models where a subset of values in the matrix $\boldsymbol{A}$ are set to zero. This is important for interpretability (which features are important to discriminate between outputs?) and computational efficiency (reduces the dimensionality of the problem).
- Learn sparse distribution, i.e. learn a model where the prediction function $q(\boldsymbol{x})$ outputs a probability distribution where some of the outputs may have a zero probability.
- Build structured prediction models, for example models that predict trees or sequences.
- Inject prior knowledge about the features in the model (**Will probably be skipped!**).

A valid question is whether should we care about this as deep learning became the *de facto* technique in machine learning. The answer is yes:

- the output layer of a neural network is a linear model,
- parameter sparsity is a very important topic in deep learning because of the cost of running these networks, especially on embedded devices,
- loss functions are the same, so it is important to study them,
- structured prediction methods are the same,
- Bayesian deep learning is a hot topic right now,
- last but not least, understand the optimization of neural networks is one of the most important question facing machine learning researchers.

In sum, understanding linear models is essential before digging into deep learning techniques.

2.1 Regression

In a regression problem, we want to predict a scalar value given the feature values. This a simple linear model defined as follows:

$$y = \boldsymbol{a}^\top \boldsymbol{x}$$

where $\boldsymbol{a} \in \mathbb{R}^d$ is the parameters of the model and $\boldsymbol{x} \in \mathbb{R}^d$ is the input feature values. The output y is a scalar. We assume here that there are d features, including the bias feature. To reformulate the problem in our framework, we have:

- a parameterized scoring function $s : \mathbb{R}^d \to \mathbb{R}$ defined as $s(\boldsymbol{x}; \boldsymbol{a}) = \boldsymbol{a}^\top \boldsymbol{x}$,
- a prediction function which is just the identity, i.e. $q : \mathbb{R} \to \mathbb{R}$ is defined as $q(w) = w$.

During training, we usually minimize the mean squared error between the model prediction and the gold outputs.

$$\boldsymbol{a}^* = \arg\min_{\boldsymbol{a} \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \left(\boldsymbol{a}^\top \boldsymbol{x}^{(i)} - y^{(i)} \right)^2 + \alpha r(\boldsymbol{a})$$

where $\{\langle \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)} \rangle\}_{i=1}^n$ is the set of n training points and $r : \mathbb{R}^d \to \mathbb{R}$ is a regularizer weighted by the scalar α. However, we could more generally write the problem as:

$$\boldsymbol{a}^* = \arg\min_{\boldsymbol{a} \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} l \left(y^{(i)}, \boldsymbol{a}^\top \boldsymbol{x}^{(i)} \right) + \alpha r(\boldsymbol{a})$$

where $l : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a loss function that compares gold outputs to the model outputs. We will study loss functions more formally later in the course.

The linear regression model we just defined outputs a single value for a given input and does not model its own uncertainty. Often, we may want to either the model uncertainty *per se* (is the model confident about its own output?) or to sample a value during prediction. For example, one of the most popular regression task is to predict the height of a person given the height of their parents. If we work on a simulation game, we may want to sample the height of generated persons instead of giving to everyone their most probable height, to take into account unknown external factor or the inherent source of randomness of the task (this is neither a biology or philosophy class, so let's not discuss about the existence of randomness). To be able to do that in our machine learning algorithm, we can design the prediction function q to output a distribution over outputs instead of returning a single value. Then, the user is free to use this distribution either to retrieve the most probable value or to sample from the distribution. One way to do this is to design the prediction function to return a Gaussian distribution whose probability density function (PDF) is defined as follows:

$$\mathcal{N}(y|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2} \left(\frac{y - \mu}{\sigma} \right)^2 \right)$$

where $\mu \in \mathbb{R}$ is the mean of the distribution and $\sigma^2 > 0$ is the variance. The Gaussian distribution is a well studied distribution and many packages/libraries are available in practice to sample from its PDF. It can also be shown that the most probable value under this distribution is the mean μ. Therefore, it makes sense to define μ as the output of the scoring function. In other word, we can define the prediction function as:

$$q(w) = \mathcal{N}(\cdot|w, \sigma^2)$$

where $w = s(\boldsymbol{x})$ is the output of the scoring function and σ^2 is a fixed pre-defined variance. Of course, fixing the variance may seem like a bad idea, we should instead estimate it from the data. However, we won't study regression too much in this course so we will focus on the this simple example. The most important point to remember here is that the prediction function does not returns a scalar value anymore but the PDF of a Gaussian distribution, which can be characterized by its mean and variance.

We can now turn to the training problem: how can we fix the parameters of the scoring function under a probabilistic model, i.e. a linear regression model whose prediction function returns a Gaussian PDF? Remember that we want to consider the training problem as an optimization problem, that is minimizing or maximizing an objective function with respect to the model parameters. To this end, we need a way to compare the model distribution to the data distribution. Let's denote $\tilde{p}(y, \boldsymbol{x})$ be the data distribution and $p(y, \boldsymbol{x})$ be the model distribution (implicitly parameterized by $\boldsymbol{a}$). A common measure used to compare probability distributions is the Kullback–Leibler (KL) divergence defined as:

$$KL\left[\tilde{p}|p\right] = \int\int \tilde{p}(y, \boldsymbol{x}) \log \frac{\tilde{p}(y, \boldsymbol{x})}{p(y, \boldsymbol{x})} dydx$$

The KL divergence is not a metric because it is asymmetric in general, i.e. $KL\left[\tilde{p}|p\right] \neq KL\left[p|\tilde{p}\right]$. However, it has the following property:

- It is positive or null, i.e. $KL\left[\tilde{p}|p\right] \geq 0$,
- it is null if and only if the two distribution are equal, i.e. $KL\left[\tilde{p}|p\right] \Leftrightarrow \tilde{p} = p$.

Let's keep in mind that our model doesn't directly modelize the joint distribution but only the conditional distribution $p(y|\boldsymbol{x}) = q(s(\boldsymbol{x}; \boldsymbol{a}))$. We can define our training optimization problem as minimizing the KL divergence between the true distribution and the model distribution:

$$\begin{aligned}
\boldsymbol{a}^* &= \arg\min_{\boldsymbol{a}} KL\left[\tilde{p}|p\right] \\
&= \arg\min_{\boldsymbol{a}} \int\int \tilde{p}(y, \boldsymbol{x}) \log \frac{\tilde{p}(y, \boldsymbol{x})}{p(y, \boldsymbol{x})} dydx \\
&= \arg\min_{\boldsymbol{a}} \underbrace{\int\int \tilde{p}(y, \boldsymbol{x}) \log \tilde{p}(y, \boldsymbol{x}) dydx}_{\text{constant wrt } \boldsymbol{a}} - \int\int \tilde{p}(y, \boldsymbol{x}) \log p(y, \boldsymbol{x}) dydx
\end{aligned}$$

Note that the first term in constant wrt to the model so it can be ignored without changing the arg min:

$$= \arg\min_{\boldsymbol{a}} - \int\int \tilde{p}(y, \boldsymbol{x}) \log p(y, \boldsymbol{x}) dydx$$

which is simply an expectation under the true ditribution:

$$\begin{aligned}
&= \arg\min_{\boldsymbol{a}} \mathbb{E}_{\tilde{p}(y,\boldsymbol{x})}[-\log p(y, \boldsymbol{x})] \\
&= \arg\min_{\boldsymbol{a}} \mathbb{E}_{\tilde{p}(y,\boldsymbol{x})}[-\log p(y|\boldsymbol{x})] + \underbrace{\mathbb{E}_{\tilde{p}(y,\boldsymbol{x})}[-\log p(\boldsymbol{x})]}_{\text{constant wrt } \boldsymbol{a}}
\end{aligned}$$

The second term is constant wrt to $\boldsymbol{a}$ as our model does not modelize the prior distribution on the data (we actually didn't even defined it! but it doesn't really matter...):

$$= \arg\min_{\boldsymbol{a}} \mathbb{E}_{\tilde{p}(y,\boldsymbol{x})}[-\log p(y|\boldsymbol{x})]$$

In general, we cannot compute the expectation over the data distribution because we don't know it. However, we have access to a training set of n samples $\{\langle \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)} \rangle\}_{i=1}^n$ which we can use to approximate the true expectation:

$$= \arg\min_{\boldsymbol{a}} \frac{1}{n} \sum_{i=1}^{n} -\log p(y|\boldsymbol{x})$$

which is a Monte-Carlo approximation using n samples. Finally, note that we rewritte the objective as:

$$= \arg\min_{\boldsymbol{a}} \frac{1}{n} \sum_{i=1}^{n} l(y, q(s(\boldsymbol{x}; \boldsymbol{a})))$$

where l is the negative log-likehood loss function. Note the the $\frac{1}{n}$ term could be removed without changing the arg min. As such, we note that:

- the objective to train the probabilistic variant of the linear regression model is highly similar to the first objective, the difference only lies in the definition of the loss function,
- we can obviously add a regularization terme $\alpha r(\boldsymbol{a})$ to the objective in this case too.

We can go further and try to compute $-\log p(y|\boldsymbol{x})$ to compare the two loss functions. This is actually straightforward as $p(y|\boldsymbol{x})$ is simply the value of the PDF parameterize with mean $\mu = s(\boldsymbol{x}; \boldsymbol{a})$ at point y:

$$\begin{aligned}
-\log p(y|\boldsymbol{x}) &= -\log\left(\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{1}{2}\left(\frac{y - s(\boldsymbol{x};\boldsymbol{a})}{\sigma}\right)^2\right)\right) \\
&= -\log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) + \frac{1}{2}\left(\frac{y - s(\boldsymbol{x};\boldsymbol{a})}{\sigma}\right)^2 \\
&= \underbrace{-\log\left(\frac{1}{\sigma\sqrt{2\pi}}\right)}_{1} + \underbrace{\frac{1}{2\sigma^2}}_{2}\left(y - s(\boldsymbol{x};\boldsymbol{a})\right)^2
\end{aligned}$$

which highlights an interesting fact: if we plug this back in the arg min and ignore the regularization term, terms 1 and 2 could be removed without impacting the arg min solution, hence we end up with the same loss function as the non-probabilistic case!

2.2 Binary classification

The second problem we will study is binary classification: given a vector of feature values $\boldsymbol{x} \in \mathbb{R}^d$ we want to predict $y \in \{-1, 1\}$, i.e. either class -1 or 1. Similarly to the linear regression model, we assume there is special bias feature so our classifier has a single parameter $\boldsymbol{a} \in \mathbb{R}^d$. Given values $\boldsymbol{x}$, the predicted class y is defined as follows:

$$y = \begin{cases} 1 & \text{if } \boldsymbol{a}^\top \boldsymbol{x} \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

or, in other words, we predict the class which has the same sign as $\boldsymbol{a}^\top \boldsymbol{x}$ (the choice of setting $\boldsymbol{a}^\top \boldsymbol{x} = 0$ to class 1 is arbitrary). In our framework, we have the following scoring and prediction function:

$$\begin{aligned}
s(\boldsymbol{x}; \boldsymbol{a}) &= \boldsymbol{a}^\top \boldsymbol{x} \\
q(w) &= \begin{cases} 1 & \text{if } w \geq 0 \\ -1 & \text{otherwise} \end{cases}
\end{aligned}$$

We will consider two cases for training a binary classifier:

- a model based on class separation called Support Vector Machine (SVM) which includes the perceptron model as a special case,
- a probabilistic model, where instead of only returning the most probable output we return a distribution over the output. Similarly to the linear regression case, the user could just want to retrieve the most probable output from this model and the result will defined the same way as the non probabilistic case.

2.2.1 Support Vector Machine (SVM)

Given a training set $\{\langle \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)} \rangle\}_{i=1}^n$of n examples, we want to find an hyperplane that separates the two classes. This hyperplane corresponds to the points solving the equation $\boldsymbol{a}^\top \boldsymbol{x} = 0$, that is the set of points $\boldsymbol{x}$ that are orthogonal to $\boldsymbol{a}$. The vectors labeled 1 will be the one that have an acute angle with $\boldsymbol{a}$ and the vectors labeled -1 will be the one that have an obtuse angle with $\boldsymbol{a}$. Given a dataset $\{\langle \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)} \rangle\}_{i=1}^n$, the regularized SVM training problem can be formalized as follows:

$$\begin{aligned}
\min_{\boldsymbol{a}} \quad & \alpha r(\boldsymbol{a}) \\
\text{s.t.} \quad & \boldsymbol{a}^\top \boldsymbol{x}^{(i)} y^{(i)} \geq m \qquad \forall 1 \leq i \leq n
\end{aligned}$$

where m is the minimum expected distance between the separating hyperplane and datapoints, e.g. we usually set $m = 1$. If $r(\boldsymbol{a}) = 0$ (or $\alpha = 0$), i.e. no regularization, the problem is a constraint satisfaction problem

where we aim to find any $\boldsymbol{a}$ that separates the datapoints. If $r(\boldsymbol{a}) = \frac{1}{2}\|\boldsymbol{a}\|_2^2$, i.e. L2 regularization, the problem aim to find the hyperplane that maximizes the margin [Boser et al., 1992, Section 2.1] between the separating hyperplane and datapoints. One important problem is to know whether a solution exists for this problem. It can be shown that if the intersection of the convex hulls of points of the two class is empty, then there exists a value of $m > 0$ for which this problem has a solution [Boyd and Vandenberghe, 2004, Section 2.5]. However, it may be that the class are not separable, i.e. that such hyperplane does not exists and the problem is infeasible. To take into account this problem, a common trick is to introduce slack variables that allows margin violation. Moreover, we add a penalty term to the object to minimize this margin violation. The resulting problem is defined as follows:

$$\begin{aligned} \min_{\boldsymbol{a},\epsilon} \quad & \alpha r(\boldsymbol{a}) + \sum_{i=1}^{n} \epsilon_i \\ \text{s.t.} \quad & \boldsymbol{a}^\top \boldsymbol{x}^{(i)} y^{(i)} \geq m - \epsilon_i \qquad \forall 1 \leq i \leq n \\ & \epsilon \geq 0 \end{aligned}$$

where the vector $\epsilon \in \mathbb{R}^n$ is the vector of slack variables, containing one value per datapoint. The slack variables are constrained to be positive or null. Note that we now optimize both over $\boldsymbol{a}$ and ϵ. We can rewrite the margin constraints as follows:

$$\begin{aligned} \boldsymbol{a}^\top \boldsymbol{x}^{(i)} y^{(i)} &\geq m - \epsilon_i && \forall 1 \leq i \leq n \\ \boldsymbol{a}^\top \boldsymbol{x}^{(i)} y^{(i)} - m &\geq -\epsilon_i && \forall 1 \leq i \leq n \\ m - \boldsymbol{a}^\top \boldsymbol{x}^{(i)} y^{(i)} &\leq \epsilon_i && \forall 1 \leq i \leq n \\ \max(0, m - \boldsymbol{a}^\top \boldsymbol{x}^{(i)} y^{(i)}) &= \epsilon_i && \forall 1 \leq i \leq n \end{aligned}$$

where the last line is valid because:

- ϵ is contrained to be positive or null, therefore if we introduce the $\max(0, \cdot)$ operator on the left-hand side it doesn't change the constraint,
- we minimize over ϵ and each ϵ_i is independent (i.e. appears in a single constraint). Therefore, the minimum value that ϵ_i is exactly the left-hand side after the introduction of the $\max(0, \cdot)$ operator.

Now that we have an equality constraint, we just plug back the left-hand side in the objective function instead of using slack variables:

$$\begin{aligned} & \min_{\boldsymbol{a}} \quad \sum_{i=1}^{n} \max(0, m - \boldsymbol{a}^\top \boldsymbol{x}^{(i)} y^{(i)}) + \alpha r(\boldsymbol{a}) \\ = & \min_{\boldsymbol{a}} \quad \sum_{i=1}^{n} l(y^{(i)}, \boldsymbol{a}^\top \boldsymbol{x}^{(i)}; m) + \alpha r(\boldsymbol{a}) \end{aligned}$$

where in the last line we introduced the loss function $l(y, y'; m) = \max(0, m - yy')$ parameterized by the margin parameter $m \geq 0$:

- if $m > 0$ (we usually simply set $m = 1$), this loss function is called the hinge loss;
- if $m = 0$, this loss function is called the perceptron loss.[1]

2.2.2 Probabilistic model

We turn to the following problem: instead of making a hard decision choice, how can we return a probability distribution over the two classes. First, we are going to make a notation change: instead of having $y \in \{-1, 1\}$ we will have $y \in \{0, 1\}$. We only introduce this change to simplify notation.

A Bernoulli distribution is a discrete distribution over a random variable taking values in $\{0, 1\}$ whose probability mass function (PMF) is defined as follows:

$$\mathcal{B}(y; \mu) = \mu^y (1 - \mu)^{1-y}$$

where $\mu \in [0, 1]$ is the mean parameter of the distribution. Alternatively, we can write down the probability of each value explicitly:

$$\begin{aligned} \mathcal{B}(y = 1; \mu) &= \mu \\ \mathcal{B}(y = 0; \mu) &= 1 - \mu \end{aligned}$$

[1] If you don't know the original perceptron model, you can check [Daumé III, 2012, Chapter 4] to understand why it is called the perceptron loss when $m = 0$.

In our framework, we can rely on a prediction function that returns the PMF of a Bernoulli distribution:

$$s(\boldsymbol{x}; \boldsymbol{a}) = \boldsymbol{a}^\top \boldsymbol{x}$$
$$q(w) = \mathcal{B}(y; \sigma(w))$$

where $\sigma(w) = \frac{1}{1+\exp(-w)} = \frac{\exp(w)}{1+\exp(w)}$ is the sigmoid function. The reason we introduce the sigmoid function in the prediction function is to ensure that the Bernoulli parameter μ is in $[0, 1]$. Indeed, $s(\boldsymbol{x}; \boldsymbol{a})$ is unconstrained and could return values inside this bound. Note that $\sigma(w) \in (0, 1)$, i.e. we will never have $\mu = 0$ or $\mu = 1$. In other word, the predicted Bernoulli distribution will have a full support: there is no value in its domain that has a null probability.

To train the probabilistic model, we rely on the same methodology that we used for regression: we minimize the negative log-likelihood of the training data (we could do the same derivation as in Section 2.1 where integrals are basically replaced by sums over $\{0, 1\}$). Given a training set $\{\langle \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)} \rangle\}_{i=1}^n$, the training objective is defined as:

$$\min_{\boldsymbol{a}} \frac{1}{n} \sum_{i=1}^{n} - \log p(y^{(i)}|\boldsymbol{x}) + \alpha r(\alpha)$$

We can rewrite the training loss as follows:

$$- \log p(y|\boldsymbol{x}) = - \log \left(\sigma(\boldsymbol{a}^\top \boldsymbol{x})^y (1 - \sigma(\boldsymbol{a}^\top \boldsymbol{x}))^{1-y} \right)$$

If $y = 1$ we have:

$$\begin{aligned} - \log \sigma(\boldsymbol{a}^\top \boldsymbol{x}) &= - \log \frac{\exp(\boldsymbol{a}^\top \boldsymbol{x})}{1 + \exp(\boldsymbol{a}^\top \boldsymbol{x})} \\ &= -\boldsymbol{a}^\top \boldsymbol{x} + \log(1 + \exp(\boldsymbol{a}^\top \boldsymbol{x})) \end{aligned}$$

If $y = 0$ we have:

$$\begin{aligned} - \log(1 - \sigma(\boldsymbol{a}^\top \boldsymbol{x})) &= - \log \left(1 - \frac{\exp(\boldsymbol{a}^\top \boldsymbol{x})}{1 + \exp(\boldsymbol{a}^\top \boldsymbol{x})} \right) \\ &= - \log \left(\frac{1 + \exp(\boldsymbol{a}^\top \boldsymbol{x})}{1 + \exp(\boldsymbol{a}^\top \boldsymbol{x})} - \frac{\exp(\boldsymbol{a}^\top \boldsymbol{x})}{1 + \exp(\boldsymbol{a}^\top \boldsymbol{x})} \right) \\ &= - \log \left(\frac{1}{1 + \exp(\boldsymbol{a}^\top \boldsymbol{x})} \right) \\ &= - \log 1 + \log \left(1 + \exp(\boldsymbol{a}^\top \boldsymbol{x}) \right) \\ &= \log \left(1 + \exp(\boldsymbol{a}^\top \boldsymbol{x}) \right) \end{aligned}$$

So the negative log-likelihood loss is simply:

$$- \log p(y|\boldsymbol{x}) = -y \boldsymbol{a}^\top \boldsymbol{x} + \log(1 + \exp(\boldsymbol{a}^\top \boldsymbol{x}))$$

2.3 Multiclass classification

In multiclass classification, we are interested in predicting a 1-in-k class given a set of feature values $\boldsymbol{x}$. It is usual to consider that $k > 2$, but the approach we are going to describe also work for binary classification (but has more parameters than the binary classifier of previous section!). Note that there exists several approaches to multiclass classification based on combining several binary classifiers. We will not consider them in this course.

Let d be the number of features, including a bias feature, and k the number of classes. The score function of multiclass classifier is defined as follows:

$$s(\boldsymbol{x}; \boldsymbol{A}) = \boldsymbol{A}\boldsymbol{x}$$

where $\boldsymbol{A} \in \mathbb{R}^{k \times d}$ is a matrix of parameters, contrary to regression and binary classification where the parameter of the scoring function is a vector. An output will be denoted with vector $\boldsymbol{y} \in \mathbb{R}^k$ of size k. In other words, a prediction of the model is represented as a basis vector (also called a one-hot vector), that is a vector with a single element equal to 1 (associated to a class) and all other elements equal to 0. Let $\mathcal{Y}(k)$ be the set of the k standard basis vectors of size k, the prediction function of the model can be defined as follows:

$$q(\boldsymbol{w}) = \arg\max_{\boldsymbol{y} \in \mathcal{Y}(k)} \boldsymbol{w}^\top \boldsymbol{y}$$

i.e. we "pick the index" with maximum weight in the vector returned by the scoring function.

2.3.1 Support Vector Machine (SVM)

We can build a SVM model similarly to the binary classification one. For the multiclass classification problem, we want to find a matrix $\boldsymbol{A}$ such that, for each training example, the gold class has a score higher to all other classes with a margin of at least $m \geq 0$. Given a training set $\{\langle \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)} \rangle\}_{i=1}^{n}$, we can write the optimization problem as follows:

$$\begin{aligned} \arg\min_{\boldsymbol{A}} \quad & \alpha r(\boldsymbol{A}) \\ \text{s.t.} \quad & \boldsymbol{y}^\top \boldsymbol{A}\boldsymbol{x}^{(i)} + m \leq \boldsymbol{y}^{(i)\top} \boldsymbol{A}\boldsymbol{x}^{(i)} \qquad \forall 1 \leq i \leq n, \boldsymbol{y} \in \mathcal{Y}(k) \setminus \{\boldsymbol{y}^{(i)}\} \end{aligned}$$

We have $k-1$ constraints per training point with this formulation. Note that if highest scoring class that is not the gold class satisfies the constraint, all other non-gold classes will also satisfy it. We can therefore replace the constraints with the following ones:

$$m + \max_{\boldsymbol{y} \in \mathcal{Y}(k) \setminus \{\boldsymbol{y}^{(i)}\}} \left(\boldsymbol{y}^\top \boldsymbol{A}\boldsymbol{x}^{(i)} \right) \leq \boldsymbol{y}^{(i)\top} \boldsymbol{A}\boldsymbol{x}^{(i)} \qquad \forall 1 \leq i \leq n$$

Similarly, we can add slack variables $\epsilon \in \mathbb{R}_+^n$ in case the problem is not separable:

$$\begin{aligned} m + \max_{\boldsymbol{y} \in \mathcal{Y}(k) \setminus \{\boldsymbol{y}^{(i)}\}} \left(\boldsymbol{y}^\top \boldsymbol{A}\boldsymbol{x}^{(i)} \right) &\leq \boldsymbol{y}^{(i)\top} \boldsymbol{A}\boldsymbol{x}^{(i)} + \epsilon_i \qquad \forall 1 \leq i \leq n \\ -\boldsymbol{y}^{(i)\top} \boldsymbol{A}\boldsymbol{x}^{(i)} + m + \max_{\boldsymbol{y} \in \mathcal{Y}(k) \setminus \{\boldsymbol{y}^{(i)}\}} \left(\boldsymbol{y}^\top \boldsymbol{A}\boldsymbol{x}^{(i)} \right) &\leq \epsilon_i \qquad \forall 1 \leq i \leq n \\ \max\left(0, -\boldsymbol{y}^{(i)\top} \boldsymbol{A}\boldsymbol{x}^{(i)} + m + \max_{\boldsymbol{y} \in \mathcal{Y}(k) \setminus \{\boldsymbol{y}^{(i)}\}} \left(\boldsymbol{y}^\top \boldsymbol{A}\boldsymbol{x}^{(i)} \right)\right) &= \epsilon_i \qquad \forall 1 \leq i \leq n \end{aligned}$$

So we can rewrite our original training problem as an unconstrained optimization problem:

$$\arg\min_{\boldsymbol{A}} \quad \sum_{i=1}^{n} \max\left(0, -\boldsymbol{y}^{(i)\top} \boldsymbol{A}\boldsymbol{x}^{(i)} + m + \max_{\boldsymbol{y} \in \mathcal{Y}(k) \setminus \{\boldsymbol{y}^{(i)}\}} \left(\boldsymbol{y}^\top \boldsymbol{A}\boldsymbol{x}^{(i)} \right)\right) + \alpha r(\boldsymbol{A})$$

where the first term is the hinge loss for multiclass classification.

2.3.2 Probabilistic model

In order to return a probability distribution over classes instead of a hard decision, we rely on a categorical distribution. A categorical distribution can be represented by a vector of size k with strictly positive values that sum to 1. Contrary to the Bernoulli case, the categorical distribution has a full support by essence, even if the definition is sometimes abused to allow sparse distributions (i.e. with at least one class with 0 probability). As the matrix $\boldsymbol{A}$ is unconstrained, the prediction function must renormalize the score vector given by the scoring function so that they represent the parameters of a valid categorical distribution. We can achieve this with the softmax function. Let $\boldsymbol{y} = \text{softmax}(\boldsymbol{w})$ where $\boldsymbol{w} \in \mathbb{R}^k$. Then, each element of $\boldsymbol{y}$ is equal to:

$$y_i = \frac{\exp(w_i)}{\sum_{j=1}^{k} \exp(w_j)} \qquad \forall 1 \leq i \leq k$$

This transformation ensures that $\forall 1 \leq i \leq k : y_i > 0$ and $\sum_{i=1}^{k} \exp(w_j) = 1$. The denominator is called the partition function and is often denoted as $Z = \sum_{j=1}^{k} \exp(w_j)$. The scoring and prediction function are therefore defined as follows:

$$\begin{aligned} s(\boldsymbol{x}; \boldsymbol{A}) &= \boldsymbol{A}\boldsymbol{x} \\ q(\boldsymbol{w}) &= \text{softmax}(\boldsymbol{w}) \end{aligned}$$

Where we return the parameters of the categorical distribution instead of distribution itself to simplify notation. During training, we minimize the dataset log-likelihood. Given a training set $\{\langle \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)} \rangle\}_{i=1}^{n}$, the training optimization problem is defined as:

$$\boldsymbol{A}^* = \arg\min_{\boldsymbol{A}} \sum_{i=1}^{n} -\log p(\boldsymbol{y}^{(i)} | \boldsymbol{x}^{(i)})$$

where:

$$\begin{aligned} -\log p(\boldsymbol{y}|\boldsymbol{x}) &= -\log \frac{\exp(\boldsymbol{y}^\top \boldsymbol{w})}{Z} \\ &= -\boldsymbol{y}^\top \boldsymbol{w} + \log Z \\ &= -\boldsymbol{y}^\top \boldsymbol{w} + \log \sum_{i=1}^{k} \exp(w_k) \end{aligned}$$

where $\boldsymbol{w} = s(\boldsymbol{x}; \boldsymbol{A})$.

3 Convex analysis

If we focus on the binary classification problem, during training we have an optimization problem to solve to fix the parameters $\boldsymbol{a}$ of the model. Given a training set $\{\langle \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)} \rangle\}_{i=1}^{n}$, the learning problem aim to solve an optimization problem of the form:

$$\min_{\boldsymbol{a}} \quad \sum_{i=1}^{n} l(y^{(i)}, \boldsymbol{a}^\top \boldsymbol{x}^{(i)}) + \alpha r(\boldsymbol{a})$$

where l is the loss function and r is a regularizer. We considered two different loss function:

$$\begin{aligned} &(1) \quad \text{Hinge}(y, \boldsymbol{a}^\top \boldsymbol{x}) = \max(0, m - \boldsymbol{a}^\top \boldsymbol{x} y) \\ &(2) \quad \text{BCE}(y, \boldsymbol{a}^\top \boldsymbol{x}) = -y \boldsymbol{a}^\top \boldsymbol{x} + \log(1 + \exp(\boldsymbol{a}^\top \boldsymbol{x})) \end{aligned}$$

where Hinge is the hinge loss used for training a SVM and BCE is the binary cross-entropy loss used to train the probabilistic variant. Depending on the loss function we use, the learning algorithm will be different. In a certain sense, we could argue that (2) is easier to optimize as the function is differentiable, hence we can rely on the well known gradient descent algorithm. On the other hand, (1) is not differentiable everywhere so the gradient of the function is not defined everywhere. However, it is sub-differentiable. We will explain these terms and the implication for optimization in this section.

Most of this section is based on [Boyd and Vandenberghe, 2004].

3.1 Convex functions

Definition 3.1: Convex sets
A set $X \subseteq \mathbb{R}^n$ is convex if and only if:

$$\forall \boldsymbol{x}, \boldsymbol{x}' \in X, \epsilon \in [0, 1] : \epsilon \boldsymbol{x} + (1 - \epsilon)\boldsymbol{x}' \in X,$$

or, in other words, any point which is a convex combination of points in X must also be in X.

Definition 3.2: Convex and concave functions
Let $X \subseteq \mathbb{R}^n$ be a convex set. A function $f : X \to \mathbb{R}$ is convex if and only if:

$$\forall \boldsymbol{x}, \boldsymbol{x}' \in X, \epsilon \in [0, 1] : f(\epsilon \boldsymbol{x} + (1 - \epsilon)\boldsymbol{x}') \leq \epsilon f(\boldsymbol{x}) + (1 - \epsilon) f(\boldsymbol{x}')$$

Note that the domain of the function is required to be convex so that the left-hand side is well defined. A function f is concave if and only if $-f$ is convex.

The reason we are interested in convex functions is because they have a unique minimum (eventually $-\infty$ in some cases, i.e. affine functions). Note that this does not mean that there exists an unique minimizer! Non-convex optimization is also a really important topic in machine learning, especially when dealing with neural networks, but we won't consider this problem in the course.

It is often useful to consider functions whose domain is extended to the full real space. Let $X \subseteq \mathbb{R}^n$ be a convex set and $f : X \to \mathbb{R}$ be a convex function. The extended-value extension of f is the function $\tilde{f} : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ defined as follow:

$$\tilde{f}(\boldsymbol{x}) = \begin{cases} f(\boldsymbol{x}) & \text{if } \boldsymbol{x} \in \mathbf{dom}\, f \\ \infty & \text{otherwise} \end{cases}$$

It is easy to show that $\tilde{f}$ is also convex. In the following, we will use f to refer both to the function and the extended-value extension $\tilde{f}$ and write $\mathbf{dom}\, f = \{\boldsymbol{x} \in \mathbb{R}^n | f(\boldsymbol{x}) \neq \infty\}$ to simplify notation.

Definition 3.3: Graph
Given a function $f : X \to \mathbb{R}$, its graph is the set of tuples in $X \times \mathbb{R}$ defined by:

$$\{(\boldsymbol{x}, f(\boldsymbol{x})) | \boldsymbol{x} \in X\}$$

Definition 3.4: Epigraph
Given a function $f : X \to \mathbb{R}$, its epigraph is the set of tuple in $X \times \mathbb{R}$ defined by:

$$\mathbf{dom}\, f = \{(\boldsymbol{x}, r) | \boldsymbol{x} \in X \text{ and } f(\boldsymbol{x}) \leq r\}$$

Definition 3.5: Supporting hyperplane
Given a set X, a supporting hyperplane of X is a hyperplane such that:

- X is entirely contained in one of the half-spaces separated by the hyperplane;
- X has at least one boundary point with the hyperplane.

Definition 3.6: Subgradient
Given a function $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$, a subgradient of f at $\boldsymbol{x} \in X$ is a vector $\boldsymbol{g} \in \mathbb{R}^n$ such that:

$$\forall \boldsymbol{x}' \in X : \quad f(\boldsymbol{x}') \geq f(\boldsymbol{x}) + \boldsymbol{g}^\top (\boldsymbol{x}' - \boldsymbol{x})$$

The set of subgradients at point $\boldsymbol{x}$ is called the subdifferential and is denoted $\partial f(\boldsymbol{x})$.

The graph of f can be understood as the set of points which are usually used in order to draw it. The epigraph of f is the set of points which are, roughly speaking, "*above*" the graph, including the graph itself. Thus, a function is convex if its domain and its epigraph are convex sets. Let X be a convex set and $f : X \to \mathbb{R}$ be a convex function. Given a point $\boldsymbol{x} \in X$, a subgradient $\boldsymbol{g} \in \partial f(\boldsymbol{x})$ can be used to define a supporting hyperplane of $\mathbf{dom}\, f$ with $(\boldsymbol{x}, f(\boldsymbol{x}))$ being a boundary point. Alternatively and equivalently, the graph $\{(\boldsymbol{x}', f(\boldsymbol{x}) + \boldsymbol{g}^\top (\boldsymbol{x}' - \boldsymbol{x})) | \boldsymbol{x}' \in X\}$ candefines a supporting hyperplane of f. It is easy to see that the function $f'(\boldsymbol{x}'; \boldsymbol{g}) = f(\boldsymbol{x}) + \boldsymbol{g}^\top (\boldsymbol{x}' - \boldsymbol{x})$ is a global subestimator of the function. As such, moving along the subgradient of a given point may be a good idea in order to minimize a function. However, such an approach is appealing only if a subgradient exists at all points of the domain of f. Figure 1 shows an example of a non-convex function which has at least one point where its subgradient is undefined. It can be proven that a subgradient exists at any point of a proper convex function via the hyperplane separation theorem [Boyd and Vandenberghe, 2004].

Finally, it is worth noting the following point: Let X be a convex set and $f : X \to \mathbb{R}$ be a convex function. If f is differentiable at $\boldsymbol{x} \in X$, then subdifferential of f at $\boldsymbol{x}$ contains a single element which is the gradient of f at $\boldsymbol{x}$, i.e. $\partial f(\boldsymbol{x}) = \{\nabla f(\boldsymbol{x})\}$

3.2 (Sub-)gradient based optimization

Let's consider the following optimization problem:

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x})$$

where $f : \mathbb{R} \to \mathbb{R}$ is a convex function which we assume as a well defined minimum, i.e. $\min_{\boldsymbol{x} \in \mathbb{R}^n} f(\boldsymbol{x}) \neq -\infty$ exists.

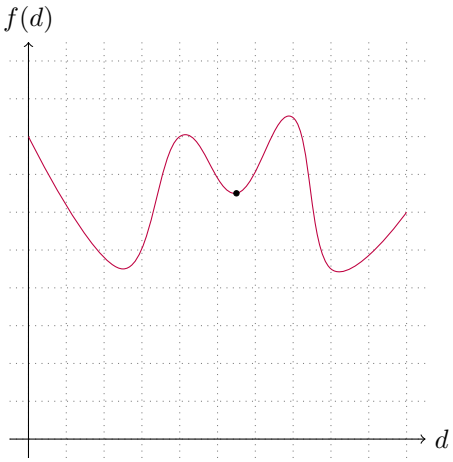

Figure 1: No supporting hyperlane to the epigraph of the function exists at the black dot. Thus, the subgradient is undefined at this point.

3.2.1 Differentiable case

In this Section, we assume f is differentiable everywhere in its domain. The gradient descent algorithm is an iterative optimization algorithm that searches for the minimum of f by considering a sequence of points as follows:

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \epsilon^{(t)} \nabla f(\boldsymbol{x}^{(t)})$$

where $\epsilon^{(t)}$ is the stepsize at time step t. The initial point $\boldsymbol{x}^{(0)}$ can be chosen randomly.

Theorem 3.7: Descent direction
Let $\boldsymbol{x}$ be a non optimal point, i.e. $f(\boldsymbol{x}) \neq \min_{\boldsymbol{x}' \in \mathbb{R}^n} f(\boldsymbol{x}')$. Then, there exist ϵ such that:

$$f\left(\boldsymbol{x} - \epsilon^{(t)} \nabla f(\boldsymbol{x})\right) < f(\boldsymbol{x})$$

We say that $-\nabla f(\boldsymbol{x})$ is a descent direction.

Proof. See [Boyd and Vandenberghe, 2004, Sections 9.2 and 9.3] □

This highlight a method to optimize the binary classifier with the BCE loss (if the regularization term is differentiable too!): follow the gradient direction. We just need a way to choose the stepsize $\epsilon^{(t)}$ at each timestep:

- line search: search for the best stepsize, i.e. solve (potentially approximately) the problem $\epsilon^{(t)} = \arg\min_{\epsilon} f(\boldsymbol{x} - \epsilon^{(t)} \nabla f(\boldsymbol{x}))$. Sometimes this problem has a simple closed form solution. One popular line search method is the backtracking line search algorithm.
- constant stepsize: define a stepsize value that will be used at each step. This is the main approach we will use in the lab exercises.
- diminishing stepsize: start with a given stepsize and decrease its value each t steps or according to the function evaluation.

In general, we can use the following fact: we know that there exists a stepsize that would give us a point with a lower function value. Hence, we can easily check manually if the stepsize is too big or not.

3.2.2 Non-differentiable case

Although gradient descent is an appealing technique, it is however not applicable directly to non-differentiable functions. Let's consider the function $f(x) = \max(0, x)$ (called rectifier linear unit or relu in machine learning), this function is differentiable everywhere except at the point 0, where the sub-differential set contains several sub-gradients. This motivates a similar but different optimization algorithm: sub-gradient descent. The idea is to simply replace the gradient with a sub-gradient at each update:

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \epsilon^{(t)} \boldsymbol{g} \qquad \text{where } \boldsymbol{g} \in \partial f(\boldsymbol{x}^{(t)})$$

Note that to simplify notation, the sub-gradient is often denoted $\nabla f(\boldsymbol{x})$ as if it was the gradient of a differentiable function (especially in the deep learning literature).

Theorem 3.8: The negative sub-gradient is not necessarly a descent direction
Everything is in the title.

Proof. We just need a counter example. Let $f : \mathbb{R}^2 \to \mathbb{R}$ be the function $f(\boldsymbol{x}) = |x_1| + 2|x_2|$. Let:

$$\boldsymbol{x} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \boldsymbol{g} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

The vector $\boldsymbol{g}$ is a sub-gradient of f at $\boldsymbol{x}$, i.e. $\in \partial f(\boldsymbol{x})$. Remember that the stepsize is strictly positive, i.e. $\epsilon > 0$.

$$\begin{aligned} f(\boldsymbol{x}) &= |x_1| + 2|x_2| \\ &= |1| + 2|0| \\ &= 1 \end{aligned}$$

$$\begin{aligned} f(\boldsymbol{x} - \epsilon\boldsymbol{g}) &= |x_1 - \epsilon g_1| + 2|x_2 - \epsilon g_2| \\ &= |1 - \epsilon 1| + 2|0 - \epsilon 2| \\ &= |1 - \epsilon| + |4\epsilon| \end{aligned}$$

By subadditivity of the absolute function, i.e. $|a + b| \leq |a| + |b|$:

$$\begin{aligned} &\geq |1 - \epsilon + 4\epsilon| \\ &= |1 + 3\epsilon| \end{aligned}$$

Therefore we have $\forall \epsilon > 0 : f(\boldsymbol{x} - \epsilon\boldsymbol{g}) \geq |1 + 3\epsilon| > f(\boldsymbol{x})$. □

This highlight a major issue with the subgradient descent algorithm: we cannot check the quality of the stepsize. Is it an issue in practice? Yes and no. For neural networks, this is not an issue in practice. For convex optimization problem, this can be an issue.

3.3 Karush–Kuhn–Tucker conditions

Assume we have a maximization problem defined as follows:

$$\begin{aligned} \max_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & g^{(i)}(\boldsymbol{x}) \geq 0 && \forall 1 \leq i \leq m \\ & h^{(i)}(\boldsymbol{x}) = 0 && \forall 1 \leq i \leq n \end{aligned}$$

where $\boldsymbol{x} \in \mathbb{R}^k$, $g^{(i)} \geq 0$ is a set of m inequality and $h^{(i)}(\boldsymbol{x}) = 0$ is a set of n equality. An optimal solution $\boldsymbol{x}^* \in \mathbb{R}^k$ of the mathematical program satisfies the following constraints:

(stationarity)	$\forall i :$	$\frac{\partial}{\partial x_i} f(x^*) + \sum_j \mu_j \frac{\partial}{\partial x_i} g^{(j)}(\boldsymbol{x}^*) - \sum_j \lambda_j \frac{\partial}{\partial x_i} h^{(j)}(\boldsymbol{x}^*) = 0$
(primal feasibility)	$\forall i :$	$g^{(i)}(\boldsymbol{x}) \geq 0$
	$\forall i :$	$h^{(i)}(\boldsymbol{x}) = 0$
(dual feasibility)	$\forall i :$	$\mu_i \geq 0$
(complementary slackness)		$\sum_i \mu_i g^{(i)}(\boldsymbol{x}^*) \geq 0$

where $\mu \in \mathbb{R}^m$ and $\lambda \in \mathbb{R}^n$ are dual variables associated with primal inequalities and equalities, respectively. These set of equation, in the general case, are necessary constraints: there may exists points $\boldsymbol{x}$ that satisfies these constraints that are not optimal solution. However, if:

- f is a concave function,
- all $g^{(i)}$ are concave functions,
- annd all $h^{(i)}$ are affine functions,

then the KKT are sufficient conditions too. In particular, we will see that in some special cases we can solve this set of equations to find the optimal $\boldsymbol{x}^*$, i.e. the optimization problem as a closed form solution.

3.4 Fenchel conjugate

Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ be a function. The (Fenchel) conjugate of f is the function $f^* : \mathbb{R}^n \to \mathbb{R}$ defined as follows:

$$f^*(\boldsymbol{y}) = \max_{\boldsymbol{x} \in \mathbf{dom}\, f} \boldsymbol{y}^\top \boldsymbol{x} - f(\boldsymbol{x})$$

The biconjugate of f is the function $f^{**} : \mathbb{R}^n \to \mathbb{R}$ defined as follows:

$$f^{**}(\boldsymbol{x}) = \max_{\boldsymbol{y} \in \mathbf{dom}\, f^*} \boldsymbol{x}^\top \boldsymbol{y} - f^*(\boldsymbol{y})$$

If f is proper, closed and convex, then $f^{**} = f$. This important property is often used to build variational formulation of functions (later in the course).

The Fenchel conjugate concept may seem abstract and useless for now. It is however a very important topic that will appear several time in the course and that appears in many places in machine learning and optimization. Instead of introducing everything you need to know about conjugates right now, I will introduce select subjects with examples along the course.

4 Dual optimization

We consider the following L2 regularized binary SVM training problem:

$$\min_{\boldsymbol{a}} \quad \sum_{i=1}^{n} \max(0, 1 - \boldsymbol{a}^\top \boldsymbol{x}^{(i)} y^{(i)}) + \frac{1}{2}\|\boldsymbol{x}\|_2^2$$

We showed that optimizing this optimization problem via subgradient descent can be problematic: the sub-gradient is not necessarily a descent direction. The sub-gradient allows use to move closer to the optimal solution in term of euclidean distance, but it is difficult to check if an iteration improves this distance as the optimal point is obviously unknown. In this section, we will show an alternative way to train a SVM in its dual formulation. This formulation will have two advantages:

- it highlights what support vectors in SVM means,
- it allows us to use a hyper-parameter free optimization algorithm (no stepsize!)

We first need to bring in some theory

4.1 Lagrangian duality

Let $\mathcal{X} \subseteq \mathbb{R}^n$ be a convex set and $f : \mathcal{X} \to \mathbb{R}$ be a function. We consider the following primal mathematical program:

$$\begin{aligned} (P) \quad & \min_{\boldsymbol{x} \in \mathcal{X}} \quad f(\boldsymbol{x}) \\ & \text{s.t.} \quad \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \end{aligned}$$

where $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$ defines a set of m linear inequalities. The Lagrangian of (P) is the function $L : \mathcal{X} \times \mathbb{R}_+^m \to \mathbb{R}$ defined as follows:

$$L(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$$

where $\boldsymbol{x}$ are the primal variables and λ the Lagrangian multipliers also called the dual variables. We build the following relaxation of the primal problem:

$$L(\lambda) = \min_{\boldsymbol{x} \in \mathcal{X}} L(\boldsymbol{x}, \lambda)$$

where L is a function $L : \mathbb{R}_+^m \to \mathbb{R}$. We use the same name for two different functions, we hope it won't confuse the reader. We call this problem relaxed because the constraints on the primal variables are replaced by penalties in the objective. It can be the case that this problem is simpler to solve than the primal problem.

Weak Lagrangian duality Let $\hat{\boldsymbol{x}}$ be the optimal solution of the primal problem (P). Then:

$$\forall \lambda \in \mathbb{R}_+^m : \quad f(\hat{\boldsymbol{x}}) \geq L(\lambda)$$

In other word, for any set of dual variables λ, the relaxed problem gives use a lower bound to the optimal solution. To prove this, first note that $\hat{\boldsymbol{x}}$ satisfies the primal constraints by definition, therefore $\lambda^\top(\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{b}) \leq 0$:

$$f(\hat{\boldsymbol{x}}) \geq f(\hat{\boldsymbol{x}}) + \lambda^\top(\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{b})$$

If this inequality is true for $\hat{\boldsymbol{x}}$, it will also be true if we try to find the $\boldsymbol{x}$ that minimizes the right-hand side:

$$\begin{aligned} &\geq \min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x}) + \lambda^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \\ &= L(\lambda) \end{aligned}$$

which ends the proof.

Lagrangian dual problem By weak Lagrangian duality, we know that for any λ the relaxed problem is a lower bound to the primal problem. Therefore, we may want to search for the best lower bound possible, that is maximizing the lower bound. This problem is called the Lagrangian dual problem and is defined as follows:

$$(D) \quad \max_{\lambda \in \mathbb{R}_+^m} L(\lambda)$$

Note that λ must be in the non-negative orthant. The difference between the primal optimal value and dual optimal value is called the duality gap. One important problem to know whether the primal and dual optimal value match or not, i.e is the duality gap null or not? We won't study this problem here, we can just note that if $\mathcal{X}$ is a convex set, f is convex function, the constraints are linear (i.e. of the form $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ as described here) and the problem is well defined (there exists at least one optimal solution), then there is no duality gap. The conditions that ensures that there is no duality gap are described in [Borwein and Lewis, 2010, Section 4.3] An important property of the duality gap is that it lets us estimate the quality of a primal feasible solution. However, this may not come for free as it requires to build a dual feasible solution. Finally, we show below how we can test if the duality gap is null.

Concavity One important property of the Lagrangian dual problem is that its objective function is concave. Moreover, this property does not depends on the the convexity of the primal objective function. Remember that a function is concave if its opposite is convex. Therefore, the following properties must be satisfied:

1. the domain of $L(\lambda)$ must be convex
2. $\forall \lambda^{(1)}, \lambda^{(2)} \in \lambda \in \mathbb{R}_+^m$ and $\epsilon \in [0, 1] : L(\epsilon\lambda^{(1)} + (1-\epsilon)\lambda^{(2)}) \geq \epsilon L(\lambda^{(1)}) + (1-\epsilon)L(\lambda^{(2)})$

The first property is trivially satisfied for the domain $\mathbb{R}_+^m$.

For the second property, lets write $\lambda = \epsilon\lambda^{(1)} + (1-\epsilon)\lambda^{(2)}$ and $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x} \in \mathcal{X}} L(\boldsymbol{x}, \lambda)$. The following inequality are satisfied by definition:

$$\begin{aligned} L(\boldsymbol{x}^*, \lambda^{(1)}) &\geq L(\lambda^{(1)}) \\ L(\boldsymbol{x}^*, \lambda^{(2)}) &\geq L(\lambda^{(2)}) \end{aligned}$$

as the right-hand sides are minimizations over the primal variables. We multiply and sum both inequalities to obtain:

$$\epsilon L(\boldsymbol{x}^*, \lambda^{(1)}) + (1-\epsilon)L(\boldsymbol{x}^*, \lambda^{(2)}) \geq \epsilon L(\lambda^{(1)}) + (1-\epsilon)L(\lambda^{(2)})$$

To simplify notations, we write $c(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$. The left-hand side of the inequality can be rewritten as:

$$\begin{aligned} \epsilon L(\boldsymbol{x}^*, \lambda^{(1)}) + (1-\epsilon)L(\boldsymbol{x}^*, \lambda^{(2)}) &= \epsilon \left(f(\boldsymbol{x}^*) + \lambda^{(1)\top} c(\boldsymbol{x}^*) \right) + (1-\epsilon) \left(f(\boldsymbol{x}^*) + \lambda^{(2)\top} c(\boldsymbol{x}^*) \right) \\ &= \epsilon f(\boldsymbol{x}^*) + \epsilon\lambda^{(1)\top} c(\boldsymbol{x}^*) + (1-\epsilon) f(\boldsymbol{x}^*) + (1-\epsilon)\lambda^{(2)\top} c(\boldsymbol{x}^*) \\ &= f(\boldsymbol{x}^*) + \left(\epsilon\lambda^{(1)} + (1-\epsilon)\lambda^{(2)} \right)^\top c(\boldsymbol{x}^*) \\ &= f(\boldsymbol{x}^*) + \lambda^\top c(\boldsymbol{x}^*) \\ &= L(\boldsymbol{x}^*, \lambda) \\ &= L(\lambda) \\ &= L(\epsilon\lambda^{(1)} + (1-\epsilon)\lambda^{(2)}) \end{aligned}$$

Hence, we obtain the inequality:

$$L(\epsilon\lambda^{(1)} + (1-\epsilon)\lambda^{(2)}) \geq \epsilon L(\lambda^{(1)}) + (1-\epsilon)L(\lambda^{(2)})$$

which proves that the Lagrangian dual objective is concave.

Strong Lagrangian duality Let $\lambda \in \mathbb{R}_+^m$ be a dual feasible solution and $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}\in\mathcal{X}} L(\boldsymbol{x}, \lambda)$. If:

- $\boldsymbol{A}\boldsymbol{x}^* \leq \boldsymbol{b}$ (primal feasibility condition)
- and $\lambda^\top(\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b}) = 0$ (complementary slackness condition)

then $\boldsymbol{x}^*$ is a primal optimal solution. To prove this, let $\hat{\boldsymbol{x}}$ be a primal optimal solution. By weak Lagrangian duality, we know that:

$$\begin{aligned} f(\hat{\boldsymbol{x}}) &\geq L(\lambda) \\ &= L(\boldsymbol{x}^*, \lambda) \\ &= f(\boldsymbol{x}^*) + \lambda^\top(\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b}) \end{aligned}$$

From the prerequisites the second term in null, therefore we have:

$$f(\hat{\boldsymbol{x}}) \geq f(\boldsymbol{x}^*)$$

Moreover, we know that $f(\hat{\boldsymbol{x}}) \leq f(\boldsymbol{x}^*)$ because $\boldsymbol{x}^*$ is primal feasible, and therefore $f(\hat{\boldsymbol{x}}) = f(\boldsymbol{x}^*)$, which ends the proof.

Equality constraints As of now we only considered linear inequality constraints. If we want to introduce equality constraints $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ instead, note that the problem can be formulated as follows:

$$\begin{aligned} (P) \quad \min_{\boldsymbol{x}\in\mathcal{X}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ & -\boldsymbol{A}\boldsymbol{x} \leq -\boldsymbol{b} \end{aligned}$$

Then, the Lagrangian is a function $L(\boldsymbol{x}, \lambda, \lambda')$ with two vectors of dual variables $\lambda \in \mathbb{R}_+^m$ and $\lambda' \in \mathbb{R}_+^m$ defined as follows:

$$\begin{aligned} L(\boldsymbol{x}, \lambda, \lambda') &= f(\boldsymbol{x}) + \lambda^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) + \lambda'^\top(-\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}) \\ &= f(\boldsymbol{x}) + \lambda^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) - \lambda'^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \\ &= f(\boldsymbol{x}) + (\lambda - \lambda')^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \end{aligned}$$

By defining $\lambda'' = \lambda - \lambda'$ we could instead formulate the Lagrangian with a single vector of dual variable:

$$L(\boldsymbol{x}, \lambda'') = f(\boldsymbol{x}) + \lambda''^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$$

In this formulation, the dual variables $\lambda'' \in \mathbb{R}^m$ is unconstrained. Equality constraints with this formulation simplifies the problem in two ways:

- First, the maximization in the Lagrangian dual problem is simpler as there is no constraint on the dual variables,
- Second, the strong duality condition simplifies as primal feasibility implies complementary slackness.

4.2 SVM dual problem optimization

We rewrite this problem by introducing a new vector of variables $\boldsymbol{v} \in \mathbb{R}^n$ such that $v_i = \boldsymbol{a}^\top \boldsymbol{x}^{(i)} y^i$:

$$\begin{aligned} \min_{\boldsymbol{a},\boldsymbol{v}} \quad & \sum_{i=1}^{n} \max(0, 1 - v_i) + \frac{1}{2}\|\boldsymbol{a}\|_2^2 \\ \text{s.t.} \quad & v_i = \boldsymbol{a}^\top \boldsymbol{x}^{(i)} y^i \qquad \forall i \in \{1...n\} \end{aligned}$$

It is more convenient to write this expression in term of matrix operations. Let $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ be a matrix where each row consists of a training point, i.e. $X_{i,j} = x_j^{(i)}$, and $\boldsymbol{Y} \in R^{n \times n}$ be a diagonal matrix containing labels in its diagonal, i.e. $Y_{i,i} = y^{(i)}$ and $\forall i \neq j : Y_{i,j} = 0$. Then we have:

$$\begin{aligned} \min_{\boldsymbol{a},\boldsymbol{v}} \quad & \sum_{i=1}^{n} \max(0, 1 - v_i) + \frac{1}{2}\|\boldsymbol{a}\|_2^2 \\ \text{s.t.} \quad & \boldsymbol{v} = \boldsymbol{Y}\boldsymbol{X}\boldsymbol{a} \end{aligned}$$

The Lagrangian of the problem is:

$$L(\boldsymbol{a}, \boldsymbol{v}, \lambda) = \sum_{i=1}^{n} \max(0, 1 - v_i) + \frac{1}{2}\|\boldsymbol{a}\|_2^2 + \lambda^\top (\boldsymbol{Y}\boldsymbol{X}\boldsymbol{a} - \boldsymbol{v})$$

from which we build the following relaxed primal problem:

$$L(\lambda) = \min_{\boldsymbol{a},\boldsymbol{v}} \quad \sum_{i=1}^{n} l(v_i) + r(\boldsymbol{a}) + \lambda^\top (\boldsymbol{Y}\boldsymbol{X}\boldsymbol{a} - \boldsymbol{v})$$

where $\lambda \in \mathbb{R}^n$ is the vector of dual variables, $l(v) = \max(0, 1 - v)$ is the hinge loss function and $r(\boldsymbol{a}) = \frac{1}{2}\|\boldsymbol{a}\|$ is the L2 regularization term. We can observe that the problem decomposes as two independent minimization problems over $\boldsymbol{a}$ and $\boldsymbol{v}$

$$= \min_{\boldsymbol{v}} \sum_{i=1}^{n} (l(v_i) - \lambda_i v_i) \qquad + \qquad \min_{\boldsymbol{a}} r(\boldsymbol{a}) + \lambda^\top \boldsymbol{Y}\boldsymbol{X}\boldsymbol{a}$$

We now consider the two terms independently. We first note that the minimization over each coordinate of $\boldsymbol{v}$ can be done independently.

$$\begin{aligned} \min_{\boldsymbol{v}} \sum_{i=1}^{n} l(v_i) - \lambda_i v_i &= \sum_{i=1}^{n} \min_{v_i} l(v_i) - \lambda_i v_i \\ &= -\sum_{i=1}^{n} \max_{v_i} \lambda_i v_i - l(v_i) \\ &= -\sum_{i=1}^{n} l^*(\lambda_i) \end{aligned}$$

where $l^*(\lambda_i)$ is the Fenchel conjugate of the hinge loss which is defined as follows:

$$l^*(\lambda_i) = \begin{cases} \lambda_i & \text{if } \lambda_i \in [-1, 0] \\ \infty & \text{otherwise} \end{cases}$$

For the second term, we have:

$$\begin{aligned} \min_{\boldsymbol{a}} r(\boldsymbol{a}) + \lambda^\top \boldsymbol{Y}\boldsymbol{X}\boldsymbol{a} &= -\max_{\boldsymbol{a}} -\lambda^\top \boldsymbol{Y}\boldsymbol{X}\boldsymbol{a} - r(\boldsymbol{a}) \\ &= -\max_{\boldsymbol{a}} \boldsymbol{a}^\top(-\boldsymbol{X}^\top \boldsymbol{Y}\lambda) - r(\boldsymbol{a}) \\ &= -r^*(-\boldsymbol{X}^\top \boldsymbol{Y}\lambda) \end{aligned}$$

where r^* is the Fenchel conjugate of the regularizer r. For L2 regularization, we have $r = r^*$:

$$\begin{aligned} &= -\frac{1}{2}\| - \boldsymbol{X}^\top \boldsymbol{Y}\lambda\|_2^2 \\ &= -\frac{1}{2}\left(-\boldsymbol{X}^\top \boldsymbol{Y}\lambda\right)^\top \left(-\boldsymbol{X}^\top \boldsymbol{Y}\lambda\right) \\ &= -\frac{1}{2}\lambda^\top \boldsymbol{Y}\boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{Y}\lambda \end{aligned}$$

We now rewrite the relaxed problem using the Fenchel conjugate notation:

$$L(\lambda) = -\sum_{i=1}^{n} l^*(\lambda_i) - r^*\left(-\boldsymbol{X}^\top \boldsymbol{Y}\lambda\right)$$

The associated Lagrangian dual problem is:

$$\begin{aligned}\max_{\lambda} L(\lambda) = &\max_{\lambda} \quad -\sum_{i=1}^{n} l^*(\lambda_i) - r^*\left(-\boldsymbol{X}^\top \boldsymbol{Y}\lambda\right) \\ = &\max_{\lambda} \quad -\sum_{i=1}^{n} \lambda_i - \frac{1}{2}\lambda^\top \boldsymbol{Y}\boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{Y}\lambda \quad \text{s.t.} \quad \forall 1 \leq i \leq n : -1 \leq \lambda_i \leq 0\end{aligned}$$

From this formulation, we can observe that:

- the objective is concave and differentiable! It should be easy to optimize
- the dual variables λ are constrained to be in a convex set (linear constraints)

Before considering the optimization problem, we should ask ourselves: how do we get the model weights $\boldsymbol{a}$, i.e. the primal variables, from the dual variables λ? To do that, we can rely on the Lagrangian. The optimal solution is a saddle point of the Lagrangian therefore its gradient is null at optimal point. If we check the gradient wrt $\boldsymbol{a}$:

$$\begin{aligned}\nabla_{\boldsymbol{a}} L(\boldsymbol{a}, \boldsymbol{v}, \lambda) &= 0 \\ \nabla_{\boldsymbol{a}} \left(\sum_{i=1}^{n} \max(0, 1 - v_i) + \frac{1}{2}\|\boldsymbol{a}\|_2^2 + \lambda^\top \left(\boldsymbol{Y}\boldsymbol{X}\boldsymbol{a} - \boldsymbol{v}\right)\right) &= 0 \\ \boldsymbol{a} + \left(\lambda^\top \boldsymbol{Y}\boldsymbol{X}\right)^\top &= 0 \\ \boldsymbol{a} &= -\boldsymbol{X}^\top \boldsymbol{Y}\lambda\end{aligned}$$

4.2.1 Project gradient ascent optimization

One way we can solve is via projected gradient ascent. If we rewrite the objective function as:

$$f(\lambda) = -\sum_{i=1}^{n} \lambda_i - \frac{1}{2}\lambda^\top \boldsymbol{Y}\boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{Y}\lambda$$

each step of gradient descent will compute new dual variable values as follows:

$$\lambda^{(t+1)} = \text{Proj}_{[-1,0]^n}\left[\lambda^{(t)} + \epsilon \nabla f(\lambda^{(t)})\right]$$

where Proj is the projection operator. The gradient has a very simple form:

$$\begin{aligned}\nabla f(\lambda^{(t)}) &= -1 - \frac{1}{2}\left(\boldsymbol{Y}^\top \boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{Y} + \left(\boldsymbol{Y}^\top \boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{Y}\right)^\top\right)\lambda^{(t)} \\ &= -1 - \boldsymbol{Y}^\top \boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{Y}\lambda^{(t)}\end{aligned}$$

See Equation 97 in the Matrix Cookbook [Petersen et al., 2008]. In our case, we want to project into the convex set $[-1, 0]^n$. This is straightforward: just clip each coordinate to $[-1, 0]$, i.e.:

$$\text{Clip}_{[0,1]}(w) = \begin{cases} -1 & \text{if } w \leq -1 \\ 0 & \text{if } w \geq 0 \\ w & \text{otherwise} \end{cases}$$

4.2.2 Box-constrained coordinate ascent optimization

The SVM dual problem is a specific case of a quadratic program that can written as follows:

$$\begin{aligned}\max_{\lambda} \quad & \frac{1}{2}\lambda^\top \boldsymbol{Q}\ \lambda + \boldsymbol{b}^\top \lambda \\ \text{s.t.} \quad & l \leq \lambda \leq u\end{aligned}$$

where $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{b} \in \mathbb{R}^n$ are weights associated with quadratic and linear terms. The matrix $\boldsymbol{Q}$ must be symmetric, i.e. $\forall i, j : Q_{i,j} = Q_{j,i}$. This problem is called a box constrained quadratic problem. In our case, we have:

- $\boldsymbol{Q} = -\boldsymbol{Y}\boldsymbol{X}\boldsymbol{X}^\top\boldsymbol{Y}$
- $\boldsymbol{b} = -1$
- $l = -1$ and $u = 0$.

The idea behind coordinate ascent is that we iteratively solve the problem wrt to one single element (coordinate) of λ only. If the objective is concave (or convex), solving wrt to λ_k only is trivial:

- Solve $\frac{\partial}{\partial\lambda_k} f(\lambda) = 0$
- If the solution does not satisfy the constraints, clip it.

Therefore the algorithm is quite simple. Note that when you solve for one coordinate, you immediately use the new coordinate solution for the next coordinate! One may visit the coordinates of λ in any order.

Note that we can rewrite the objective as:

$$\begin{aligned} f(\lambda) &= \frac{1}{2}\lambda^\top\boldsymbol{Q}\lambda + \boldsymbol{b}^\top\lambda \\ &= \frac{1}{2}\sum_{i=1}^{n}\lambda_i\sum_{j=1}^{n}\lambda_j Q_{j,i} + \sum_{i=1}^{n} b_i\lambda_i \\ &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\lambda_i\lambda_j Q_{j,i} + \sum_{i=1}^{n} b_i\lambda_i \\ &= \frac{1}{2}\sum_{i\neq j}\lambda_i\lambda_j Q_{j,i} + \frac{1}{2}\sum_{i}\lambda_i^2 Q_{i,i} + \sum_{i=1}^{n} b_i\lambda_i \end{aligned}$$

whose partial derivative is:

$$\frac{\partial}{\partial\lambda_k} f(\lambda) = \sum_{i\neq k}\lambda_i Q_{k,i} + \lambda_k Q_{k,k} + b_k$$

Solving for the derivate equals to zero gives:

$$\begin{aligned} \sum_{i\neq k}\lambda_i Q_{k,i} + \lambda_k Q_{k,k} + b_k &= 0 \\ \lambda_k &= \frac{-b_k - \sum_{i\neq k}\lambda_i Q_{k,i}}{Q_{k,k}} \end{aligned}$$

Therefore, solving for coordinate k is simply setting:

$$\lambda_k = \text{Clip}_{[l,u]}\left[\frac{-b_k - \sum_{i\neq k}\lambda_i Q_{k,i}}{Q_{k,k}}\right]$$

5 (TODO) Sparsity inducing norms

6 Sparsemax

In Section 2, we introduced two different prediction function for multiclass classification. Let $\boldsymbol{w} \in \mathbb{R}^k$ be a score vector. In the probabilistic case, we defined:

$$q(\boldsymbol{w}) = \text{softmax}(\boldsymbol{w})$$

The predicted vector $\boldsymbol{y} = q(\boldsymbol{w}), \boldsymbol{y} \in \mathbb{R}^k$ contains the probability associated with each output class defined as follows:

$$y_i = \frac{\exp(w_i)}{\sum_j \exp(w_j)}$$

Therefore, by definition, the probability of a given class is strictly positive, i.e. $\forall i : y_i > 0$. We say that this distribution has full support. In the non-probabilistic case, we defined the prediction function as follows:

$$q(\boldsymbol{w}) = \arg\max_{\boldsymbol{y}\in\mathcal{Y}(k)} \boldsymbol{y}^\top\boldsymbol{w}$$

where $\mathcal{Y}(k)$ is the set of standard basis vectors of dimension k. We can consider each $\boldsymbol{y} \in \mathcal{Y}(k)$ as probability distribution which contains a single event with probability one and all other have a null probability. These probability distributions are all sparse because there is always at least one element with null probability, i.e. $\exists i : y_i = 0$. If we ignore ties in the scores, the same prediction function can be formalized as follows:

$$\begin{aligned} q(\boldsymbol{w}) = \arg\max_{\boldsymbol{y}} \quad & \boldsymbol{y}^\top \boldsymbol{w} \\ \text{s.t.} \quad & \sum_{i=1}^{k} y_i = 1 \\ & \boldsymbol{y} \geq 0 \end{aligned}$$

The constraints defines what is called the $k-1$ probability simplex:

$$\triangle^k = \left\{ \boldsymbol{y} \in \mathbb{R}_+^k \middle| \sum_{i=1}^{k} y_i = 1 \right\}$$

6.1 Variational formulation of softmax

Even if we can interpret their output as probability distributions, the two prediction functions for multiclass classification seems rather different: one is defined a function and the second as the result of an optimization problem. In this section we show that the two can actually be formulated in a similar framework.

To show this, we must first understand that the softmax function is result of the following optimization problem:

$$\begin{aligned} \arg\max_{\boldsymbol{y}} \quad & \boldsymbol{y}^\top \boldsymbol{w} - \sum_{i=1}^{k} y_i \log y_i \\ \text{s.t.} \quad & \sum_{i=1}^{k} y_i = 1 \\ & \boldsymbol{y} \geq 0 \end{aligned}$$

The objective is convex and the constraints are linear, therefore the KKT conditions necessary and sufficient conditions to characterize the optimal solution. The Lagrangian is defined as follows:

$$L(\boldsymbol{y}, \lambda, \mu) = \boldsymbol{y}^\top \boldsymbol{w} - y_i \sum_{i=1}^{k} \log y_i + \lambda \left(\sum_{i=1}^{k} y_i - 1 \right) - \mu^\top (-\boldsymbol{y})$$

and the KKT conditions are:

$$\begin{aligned} &\text{(stationarity)} && \nabla_{\boldsymbol{y}} L(\boldsymbol{w}, \lambda, \mu) = 0 \\ &\text{(primal feasibility)} && \sum_{i=1}^{k} y_i = 1 \\ & && \boldsymbol{y} \geq 0 \\ &\text{(dual feasibility)} && \mu \geq 0 \\ &\text{(complementary slackness)} && \mu^\top (-\boldsymbol{y}) = 0 \end{aligned}$$

From primal feasibility we have:

$$\begin{aligned} \frac{\partial}{\partial y_i} L(\boldsymbol{y}, \lambda, \mu) &= 0 \\ w_i - \log y_i - 1 + \lambda + \mu_i &= 0 \\ \log y_i &= w_i - 1 + \lambda + \mu_i \\ y_i &= \exp\left(w_i - 1 + \lambda + \mu_i\right) \end{aligned}$$

meaning we have $y_i > 0$ and by complementary slackness we have $\mu_i = 0$:

$$y_i = \frac{\exp\left(w_i\right)}{\exp(1 - \lambda)}$$

If we plug this expression into primal feasibility we have:

$$\sum_{i=1}^{k} y_i = 1$$
$$\sum_{i=1}^{k} \frac{\exp(w_i)}{\exp(1-\lambda)} = 1$$
$$\exp(1-\lambda) = \sum_{i=1}^{k} \exp(w_i)$$

Plugging this back into the definition of y_i we obtain:

$$y_i = \frac{\exp(w_i)}{\sum_{j=1}^{k} \exp(w_j)}$$

hence $\boldsymbol{y} = \text{softmax}(\boldsymbol{w})$ is the optimal solution.

We can use this fact to define a framework for prediction function for the multiclass classifiction:

$$q_\Omega(\boldsymbol{w}) = \underset{\boldsymbol{y} \in \mathbf{dom}\,\Omega}{\arg\max}\, \boldsymbol{y}^\top \boldsymbol{w} - \Omega(\boldsymbol{y})$$

where Ω is a convex regularization function. Note that the regularization is not applied to the parameters of the scoring function but to the output of the prediction function. We denote $I_{\triangle^k}$ the probability simplex indicator function, i.e.:

$$I_{\triangle^k}(\boldsymbol{y}) = \begin{cases} 0 & \text{if } \boldsymbol{y} \in \triangle^k \\ \infty & \text{otherwise} \end{cases}$$

We can obtain our two prediction functions as follows:

- $\Omega(\boldsymbol{y}) = I_{\triangle^k}(\boldsymbol{y})$ gives us the hard choice variant,
- $\Omega(\boldsymbol{y}) = \sum_i y_i \log y_i + I_{\triangle^k}(\boldsymbol{y})$ (i,e. the negative Shannon entropy) gives us the softmax variant.

6.2 Sparse distribution via projection into the simplex

In this section we present a third prediction function for multiclass classification called Sparsemax. Contrary to the two previous prediction functions:

- Sparsemax can have a sparse support, i.e. there may be $y_i = 0$ in the output.
- Sparsemax does not necessarily put all the mass on a single output contrary to the hard selection prediction function.

The sparsemax function is simply the projection of its input into the simplex:

$$\text{sparsemax}(\boldsymbol{w}) = \underset{\boldsymbol{y} \in \triangle^k}{\arg\min} \|\boldsymbol{y} - \boldsymbol{w}\|_2^2$$

If the input is already in the simplex, i.e. $\boldsymbol{w} \in \triangle^k$, then the output is $\boldsymbol{w}$ meaning that the sparsemax can return a probability distribution of full support! However, if it is outside the simplex, the projection will hit a boundary of the simplex. Therefore, the Sparsemax transformation is likely to return a sparse distribution. The sparsemax output can be computed in linear-time, see [Martins and Astudillo, 2016].

It can be showed that:

$$\text{sparsemax}(\boldsymbol{w}) = \underset{\boldsymbol{y} \in \triangle^k}{\arg\min} \|\boldsymbol{y} - \boldsymbol{w}\|_2^2 = \underset{\boldsymbol{y} \in \triangle^k}{\arg\max}\, \boldsymbol{y}^\top \boldsymbol{w} - \frac{1}{2}\|\boldsymbol{y}\|_2^2$$

therefore we can define it using our framework:

$$q_\Omega(\boldsymbol{w}) = \underset{\boldsymbol{y} \in \mathbf{dom}\,\Omega}{\arg\max}\, \boldsymbol{y}^\top \boldsymbol{w} - \Omega(\boldsymbol{y})$$

by setting the regularizer $\Omega(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{y}\|_2^2 + I_{\triangle^k}(\boldsymbol{y})$.

6.3 Training loss

The partial derivative of the log-softmax function is:

$$\begin{aligned}
\frac{\partial}{\partial w_k} - \log \frac{\exp(w_g)}{\sum_i \exp(w_i)} &= -\frac{\partial}{\partial w_k} w_g + \frac{\partial}{\partial w_k} \log \sum_i \exp(w_i) \\
&= -\mathbf{1}[k = g] + \frac{\partial}{\partial w_k} \log \sum_i \exp(w_i) \\
&= -\mathbf{1}[k = g] + \frac{1}{\sum_i \exp(w_i)} \frac{\partial}{\partial w_k} \exp(w_i) \\
&= -\mathbf{1}[k = g] + \frac{1}{\sum_i \exp(w_i)} \exp(w_k) \\
&= -\mathbf{1}[k = g] + \frac{\exp(w_k)}{\sum_i \exp(w_i)}
\end{aligned}$$

Hence:

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}, \hat{\boldsymbol{y}}) = -\hat{\boldsymbol{y}} + \text{softmax}(\boldsymbol{w})$$

Therefore, we may want to use the following gradient to train a sparse distribution model:

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{w}, \hat{\boldsymbol{y}}) = -\hat{\boldsymbol{y}} + \text{sparsemax}(\boldsymbol{w})$$

However, this formulation doesn't give us the objective function! We could artificially build an objective so that its gradient is this equation, but this seems a little bit artificial. In the next section, we will present a general framework that can be used to build loss functions for regularized prediction functions.

7 Fenchel-Young losses

In this section, we will describe a generic method to build loss functions for regularized multiclass prediction functions called Fenchel-Young losses [Blondel et al., 2019]. This approach is also suitable to other prediction problems. We assume a multiclass prediction function defined as follows:

$$q_\Omega(\boldsymbol{w}) = \underset{\boldsymbol{y} \in \mathbf{dom}\, \Omega}{\arg\max}\, \boldsymbol{y}^\top \boldsymbol{w} - \Omega(\boldsymbol{y})$$

where $\Omega(\boldsymbol{y})$ is a regularizer. This family of prediction and loss functions is motivated the use of sparse output distributions whose parameters are computed via the sparsemax function, which relies on the regularizer $\Omega(\boldsymbol{y}) = \frac{1}{2}\|\boldsymbol{y}\| + I_{\triangle^k}(\boldsymbol{y})$. As we showed, minimizing the negative log-likelihood is ill-defined in this case. We will take the same approach than in the previous section: we first analyze the negative log-likelihood for the "softmax distribution" and then use a variational formulation to generalize this loss function to other regularizers.

7.1 Variational formulation of the negative log-likelihood loss for the softmax prediction function

In this section we assume the prediction function of a multiclass classifier is the softmax function:

$$q(\boldsymbol{w}) = \text{softmax}(\boldsymbol{w})$$

In Section 2, we argued that training this classifier can be done via the negative log-likelihood loss function. Let $\langle \boldsymbol{x}, \hat{\boldsymbol{y}} \rangle$ be a tuple from our training data and $\boldsymbol{w} = s(\boldsymbol{x})$ be the score vector of our model for the feature vector $\boldsymbol{x}$. Then, the negative log-likelihood is given by:

$$\begin{aligned}
L(\boldsymbol{w}; \hat{\boldsymbol{y}}) &= -\log \hat{\boldsymbol{y}}^\top \text{softmax}(\boldsymbol{w}) \\
&= -\log \hat{\boldsymbol{y}}^\top \frac{\exp(\boldsymbol{w})}{\sum_i \exp(w_i)} \\
&= -\log \frac{\exp(\hat{\boldsymbol{y}}^\top \boldsymbol{w})}{Z(\boldsymbol{w})} \\
&= -\hat{\boldsymbol{y}}^\top \boldsymbol{w} + \log Z(\boldsymbol{w})
\end{aligned}$$

where $Z(\boldsymbol{w}) = \sum_i \exp(w_i)$ is the partition function. The function $f(\boldsymbol{w}) = \log Z(\boldsymbol{w})$ is convex, see [Boyd and Vandenberghe, 20 Examples 3.1.5]. As f is convex and closed, we have $f^{**} = f$:

$$f(\boldsymbol{w}) = f^{**}(\boldsymbol{w}) = \max_{\boldsymbol{y}} \boldsymbol{y}^\top \boldsymbol{w} - f^*(\boldsymbol{y})$$

that is, we can replace the $\log Z(\boldsymbol{w})$ term in the loss by the result of an optimization problem, i.e. we build a variational formulation of the log-partition function. To understand what this variational formulation is exactly, let's first compute the Fenchel conjugate of f:

$$f^*(\boldsymbol{y}) = \max_{\boldsymbol{w}} \boldsymbol{w}^\top \boldsymbol{y} - f(\boldsymbol{w}) = \max_{\boldsymbol{w}} \boldsymbol{w}^\top \boldsymbol{y} - \log Z(\boldsymbol{w})$$

which is concave function. To compute the optimal $\boldsymbol{w}$, we can set the equation $\nabla_{\boldsymbol{w}} \boldsymbol{w}^\top \boldsymbol{y} - \log Z(\boldsymbol{w}) = 0$. For each partial derivative we have:

$$\begin{aligned}
\frac{\partial}{\partial w_k} \left(\boldsymbol{w}^\top \boldsymbol{y} - \log Z(\boldsymbol{w}) \right) &= 0 \\
y_k - \frac{1}{Z} \frac{\partial}{\partial w_k} Z &= 0 \\
y_k - \frac{1}{Z} \frac{\partial}{\partial w_k} \sum_i \exp(w_i) &= 0 \\
y_k - \frac{\exp(w_k)}{Z} &= 0 \\
y_k &= \frac{\exp(w_k)}{Z}
\end{aligned}$$

Note that this equation is only solvable if $\sum_i y_i = 1$ and $\forall i : y_i > 0$, which imposes constraint on the domain of the conjugate f^*. It is often usefull to consider extended formulations of exp and log where we set $\exp(-\infty) = 0$, $\log 0 = -\infty$ and $0 \log 0 = 0$. Then we can set $\mathbf{dom}\, f^* = \triangle^k$. We have:

$$\begin{aligned}
\exp(w_k) &= y_k Z \\
w_k &= \log y_k + \log Z
\end{aligned}$$

We replace this formulation of w in the conjugate formulation:

$$\begin{aligned}
f^*(\boldsymbol{y}) &= \max_{\boldsymbol{w}} \boldsymbol{w}^\top \boldsymbol{y} - \log Z(\boldsymbol{w}) \\
&= \max_{\boldsymbol{w}} \sum_i w_i y_i - \log Z(\boldsymbol{w}) \\
&= \sum_i \left(\log y_i + \log Z \right) y_i - \log Z(\boldsymbol{w}) \\
&= \sum_i y_i \log y_i + \sum_i y_i \log Z - \log Z(\boldsymbol{w})
\end{aligned}$$

as $y \in \triangle^k$ we have $\sum_i y_i \log Z = \log Z$, therefore

$$= \sum_i y_i \log y_i$$

which is the negative entropy of the discrete distribution parameterize by $\boldsymbol{y}$. In other word, if we consider the extended real formulation of f^*, where $\mathbf{dom}\, f^* = \triangle^k$ we have:

$$\begin{aligned}
f^*(\boldsymbol{y}) &= \begin{cases} \sum_i y_i \log y_i & \text{if } \boldsymbol{y} \in \triangle^k \\ \infty & \text{otherwise} \end{cases} \\
&= \sum_i y_i \log y_i + I_{\triangle^k}(\boldsymbol{y})
\end{aligned}$$

Therefore, the variational formulation of the log-partition function can be written as:

$$\begin{aligned}
f(\boldsymbol{w}) &= f^{**}(\boldsymbol{w}) \\
&= \max_{\boldsymbol{y}} \boldsymbol{y}^\top \boldsymbol{w} - f^*(\boldsymbol{y}) \\
&= \max_{\boldsymbol{y}} \boldsymbol{y}^\top \boldsymbol{w} - \sum_i y_i \log y_i + I_{\triangle^k}(\boldsymbol{y})
\end{aligned}$$

where we know how to compute the optimal $\boldsymbol{y}$: it is the softmax (see Section 6.1). So our loss function can be equivalently formulated as:

$$\begin{aligned}
L(\boldsymbol{w}; \hat{\boldsymbol{y}}) &= -\boldsymbol{y}^\top \boldsymbol{w} + \log Z(\boldsymbol{w}) \\
&= -\boldsymbol{y}^\top \boldsymbol{w} + f^{**}(\boldsymbol{w}) \\
&= -\boldsymbol{y}^\top \boldsymbol{w} + \max_{\boldsymbol{y}} \left(\boldsymbol{y}^\top \boldsymbol{w} - f^*(\boldsymbol{y}) \right)
\end{aligned}$$

7.2 Generalization to other prediction functions

In the previous section we showed how we can build a variational formulation of the negative log-likelihood loss when the prediction function is the softmax function. In this section, we generalize this approach to other multiclass prediction functions that can be written as follows:

$$q_\Omega(\boldsymbol{w}) = \arg\max_{\boldsymbol{y} \in \mathbf{dom}\, \Omega} \boldsymbol{y}^\top \boldsymbol{w} - \Omega(\boldsymbol{y})$$

where $\Omega(\boldsymbol{y})$ is a regularizer. This formulation includes the three prediction functions that we introduced in this course:

- Hard choice: $\Omega(\boldsymbol{y}) = I_{\triangle^k}(\boldsymbol{y})$,
- Softmax: $\Omega(\boldsymbol{y}) = \sum_i y_i \log y_i + I_{\triangle^k}(\boldsymbol{y})$,
- Sparsemax: $\Omega(\boldsymbol{y}) = \frac{1}{2}\|\boldsymbol{y}\|_2^2 + I_{\triangle^k}(\boldsymbol{y})$.

Definition 7.1: Fenchel-Young loss
Let $\Omega : \mathbb{R}^k \to \mathbb{R} \cup \{\infty\}$ be a regularization function, $\boldsymbol{w} \in \mathbb{R}^k$ a score vector and $\hat{\boldsymbol{y}}$ a gold label. The Fenchel-Young loss associated with Ω is defined as:

$$L_\Omega(\boldsymbol{w}; \hat{\boldsymbol{y}}) = \Omega^*(\boldsymbol{w}) + \Omega(\hat{y}) - \boldsymbol{w}^\top \hat{\boldsymbol{y}}$$

We can check that we correctly recover the negative log-likelihood loss for the softmax prediction function:

$$L_\Omega(\boldsymbol{w}; \hat{\boldsymbol{y}}) = \Omega^*(\boldsymbol{w}) + \Omega(\hat{y}) - \boldsymbol{w}^\top \hat{\boldsymbol{y}}$$

If $\Omega(\boldsymbol{y}) = \sum_i y_i \log y_i + I_{\triangle^k}(\boldsymbol{y})$ we have:

$$= \max_{\boldsymbol{y} \in \triangle^k} \left(\boldsymbol{y}^\top \boldsymbol{w} - \Omega(\boldsymbol{y})\right) + \sum_i \hat{y}_i \log \hat{y}_i - \boldsymbol{w}^\top \hat{\boldsymbol{y}}$$

The first term is exactly $Z(\boldsymbol{w})$ in our previous notation (i.e. $f(\boldsymbol{w})$). The second term is $\sum_i y_i \log y_i = 0$ because $\hat{\boldsymbol{y}}$ contains only 0 or 1 values, $1 \log 1 = 0$ and $0 \log 0 = 0$. Therefore we have:

$$= -\boldsymbol{w}^\top \hat{\boldsymbol{y}} + \log Z$$

which is exactly what we expected!

The Fenchel-young loss for the hard choice, i.e. $\Omega(\boldsymbol{y}) = I_{\triangle^k}(\boldsymbol{y})$ is:

$$\begin{aligned} L_\Omega(\boldsymbol{w}; \hat{\boldsymbol{y}}) &= \Omega^*(\boldsymbol{w}) + \Omega(\hat{y}) - \boldsymbol{w}^\top \hat{\boldsymbol{y}} \\ &= \max_{\boldsymbol{y} \in \triangle^k} \left(\boldsymbol{y}^\top \boldsymbol{w}\right) - \boldsymbol{w}^\top \hat{\boldsymbol{y}} \end{aligned}$$

which is exactly the perceptron loss (or the SVM loss with a null margin $m = 0$). If the highest scoring class is the gold class, the loss is null, which is equivalent to the notation in Section 2.

Finally, we can build a loss function for the Sparsemax prediction function. Let $\Omega(\boldsymbol{y}) = \frac{1}{2}\|\boldsymbol{y}\|_2^2 + I_{\triangle^k}(\boldsymbol{y})$, we have:

$$\begin{aligned} L_\Omega(\boldsymbol{w}; \hat{\boldsymbol{y}}) &= \Omega^*(\boldsymbol{w}) + \Omega(\hat{y}) - \boldsymbol{w}^\top \hat{\boldsymbol{y}} \\ &= \max_{\boldsymbol{y} \in \triangle^k} \left(\boldsymbol{y}^\top \boldsymbol{w} - \frac{1}{2}\|\boldsymbol{y}\|_2^2\right) + \frac{1}{2}\|\hat{\boldsymbol{y}}\|_2^2 - \boldsymbol{w}^\top \hat{\boldsymbol{y}} \\ &= \text{sparsemax}(\boldsymbol{w})^\top \boldsymbol{w} - \frac{1}{2}\|\text{sparsemax}(\boldsymbol{w})\|_2^2 + \frac{1}{2}\|\hat{\boldsymbol{y}}\|_2^2 - \boldsymbol{w}^\top \hat{\boldsymbol{y}} \\ &= \text{sparsemax}(\boldsymbol{w})^\top \boldsymbol{w} - \frac{1}{2}\|\text{sparsemax}(\boldsymbol{w})\|_2^2 + \frac{1}{2}\|\hat{\boldsymbol{y}}\|_2^2 - \boldsymbol{w}^\top \hat{\boldsymbol{y}} - \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ &= \frac{1}{2}\left(\|\hat{\boldsymbol{y}}\|_2^2 + \|\boldsymbol{w}\|_2^2 - 2\boldsymbol{w}^\top \hat{\boldsymbol{y}}\right) - \frac{1}{2}\left(\|\text{sparsemax}(\boldsymbol{w})\|_2^2 + \|\boldsymbol{w}\|_2^2 - 2\text{sparsemax}(\boldsymbol{w})^\top \boldsymbol{w}\right) \\ &= \frac{1}{2}\|\hat{\boldsymbol{y}} - \boldsymbol{w}\|_2^2 - \frac{1}{2}\|\text{sparsemax}(\boldsymbol{w}) - \boldsymbol{w}\|_2^2 \end{aligned}$$

The gradient of the loss function is then defined as:

$$\begin{aligned}\nabla_{\boldsymbol{w}} L_\Omega(\boldsymbol{w}; \hat{\boldsymbol{y}}) &= \nabla_{\boldsymbol{w}} \frac{1}{2} \|\hat{\boldsymbol{y}} - \boldsymbol{w}\|_2^2 - \nabla_{\boldsymbol{w}} \frac{1}{2} \|\text{sparsemax}(\boldsymbol{w}) - \boldsymbol{w}\|_2^2 \\ &= -\hat{\boldsymbol{y}} + \boldsymbol{w} + \text{sparsemax}(\boldsymbol{w}) - \boldsymbol{w} \\ &= -\hat{\boldsymbol{y}} + \text{sparsemax}(\boldsymbol{w})\end{aligned}$$

which is the gradient we were aiming for.

7.3 Properties

A very important feature of using generic framework like Fenchel-Young losses is that we can study their global properties without checking them for each particular variant. We show a few example in this section.

Theorem 7.2: Non-negativity
Let $\Omega : \mathbb{R}^k \to \mathbb{R} \cup \{\infty\}$ be a regularization function, $\boldsymbol{w} \in \mathbb{R}^k \in \mathbf{dom}\,\Omega^*$ a score vector and $\hat{\boldsymbol{y}} \in \mathbf{dom}\,\Omega$ a gold label. The Fenchel-Young loss associated with Ω is non-negative:

$$L_\Omega(\boldsymbol{w}; \hat{\boldsymbol{y}}) \geq 0$$

Proof. To prove this, we use the Fenchel-Young inequality:

$$\begin{aligned}\Omega^*(\boldsymbol{w}) + \Omega(\hat{\boldsymbol{y}}) &\geq \boldsymbol{w}^\top \hat{\boldsymbol{y}} \\ \Omega^*(\boldsymbol{w}) + \Omega(\hat{\boldsymbol{y}}) - \boldsymbol{w}^\top \hat{\boldsymbol{y}} &\geq 0 \\ L_\Omega(\boldsymbol{w}; \hat{\boldsymbol{y}}) &\geq 0\end{aligned}$$

Theorem 7.3: Convexity
Let $\Omega : \mathbb{R}^k \to \mathbb{R} \cup \{\infty\}$ be a regularization function, $\boldsymbol{w} \in \mathbb{R}^k \in \mathbf{dom}\,\Omega^*$ a score vector and $\hat{\boldsymbol{y}} \in \mathbf{dom}\,\Omega$ a gold label. The Fenchel-Young loss associated with Ω is convex and $q_\Omega(\boldsymbol{w}) - \hat{\boldsymbol{y}} \in \partial L_\Omega(\boldsymbol{w}; \hat{\boldsymbol{y}})$ is a subgradient.

Proof. We have:

$$L_\Omega(\boldsymbol{w}; \hat{\boldsymbol{y}}) = \Omega^*(\boldsymbol{w}) + \Omega(\hat{\boldsymbol{y}}) - \boldsymbol{w}^\top \hat{\boldsymbol{y}}$$

The first term is convex as the Fenchel conjugate is the maximum of a set of affine functions and the third term is affine. Note that the second term is constant wrt $\boldsymbol{w}$ so Fenchel-Young losses are convex even for non-convex regularizers. We have:

- $q_\Omega(\boldsymbol{w}) \in \partial\Omega^*(\boldsymbol{w})$ by Danskin's theorem [Bertsekas, 1971].
- $-\hat{\boldsymbol{y}} = \nabla_{\boldsymbol{w}} - \boldsymbol{w}^\top \hat{\boldsymbol{y}}$.

so $q_\Omega(\boldsymbol{w}) - \hat{\boldsymbol{y}} \in \partial L_\Omega(\boldsymbol{w}; \hat{\boldsymbol{y}})$ by standard subgradient calculus rules.[2] □

Theorem 7.4: Zero loss
Let $\Omega : \mathbb{R}^k \to \mathbb{R} \cup \{\infty\}$ be a lower semi-continuous proper convex function regularization function, $\boldsymbol{w} \in \mathbb{R}^k \in \mathbf{dom}\,\Omega^*$ a score vector and $\hat{\boldsymbol{y}} \in \mathbf{dom}\,\Omega$ a gold label. $L_\Omega(\boldsymbol{w}; \hat{\boldsymbol{y}}) = 0$ if and only if $\hat{\boldsymbol{y}} \in \partial\Omega^*(\boldsymbol{w})$.

Proof.

$$\begin{aligned}L_\Omega(\boldsymbol{w}; \hat{\boldsymbol{y}}) &= \Omega^*(\boldsymbol{w}) + \Omega(\hat{\boldsymbol{y}}) - \boldsymbol{w}^\top \hat{\boldsymbol{y}} \\ &= \max_{\boldsymbol{y} \in \mathbf{dom}\,\Omega} \big(\boldsymbol{w}^\top \boldsymbol{y} - \Omega(\boldsymbol{y})\big) + \Omega(\hat{\boldsymbol{y}}) - \boldsymbol{w}^\top \hat{\boldsymbol{y}}\end{aligned}$$

By premises we have $\hat{\boldsymbol{y}} \in \partial\Omega^*(\boldsymbol{w})$, i.e. the maximization in $\Omega^*(\boldsymbol{w})$ is satisfied for $\boldsymbol{y} = \hat{\boldsymbol{y}}$ (Danskin's theorem):

$$\begin{aligned}&= \boldsymbol{w}^\top \hat{\boldsymbol{y}} - \Omega(\hat{\boldsymbol{y}}) + \Omega(\hat{\boldsymbol{y}}) - \boldsymbol{w}^\top \hat{\boldsymbol{y}} \\ &= 0\end{aligned}$$

[2] See Section 3.2 in `https://web.stanford.edu/class/ee364b/lectures/subgradients_notes.pdf`

7.4 (TODO) SVM loss

8 Structured prediction

In regression, binary classification and multiclass classification, we assumed that we have access to a fixed size vector of feature $\boldsymbol{x}$ and we wish to output a fixed size vector (or scalar) output $\boldsymbol{y}$. In structured prediction we are interested in classification problems where:

- the input size is not fixed
- the output space depends on the input
- an output can be decomposed into parts.

For example, assume that our input is a sentence $s = s_1 s_2 ... s_d$ of d words. The part-of-speech tagging problem aim to categorize each word into class describing its grammatical properties: noun, adjective, verb... We could consider each word independently and rely on a multiclass classification, but there is a strong interdependence between the category assigned to each word. For example, in English a determiner is mainly followed by an adjective or a noun. Hence, we want to take that into account and makes the prediction for the whole sentence at once.

Let $s = s_1 s_2 ... s_d$ be an input sentence of d words and t be the number of possible part of speech tags. A prediction of ta

9 (TODO) Bayesian Machine Learning

References

[Bertsekas, 1971] Bertsekas, D. P. (1971). *Control of uncertain systems with a set-membership description of the uncertainty.* PhD thesis, Massachusetts Institute of Technology.

[Blondel et al., 2019] Blondel, M., Martins, A., and Niculae, V. (2019). Learning classifiers with fenchel-young losses: Generalized entropies, margins, and algorithms. In Chaudhuri, K. and Sugiyama, M., editors, *Proceedings of Machine Learning Research*, volume 89 of *Proceedings of Machine Learning Research*, pages 606–615. PMLR.

[Borwein and Lewis, 2010] Borwein, J. and Lewis, A. S. (2010). *Convex analysis and nonlinear optimization: theory and examples*. Springer Science & Business Media.

[Boser et al., 1992] Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152.

[Boyd and Vandenberghe, 2004] Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

[Daumé III, 2012] Daumé III, H. (2012). A course in machine learning. *Publisher, ciml. info*, 5:69.

[Martins and Astudillo, 2016] Martins, A. and Astudillo, R. (2016). From softmax to sparsemax: A sparse model of attention and multi-label classification. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1614–1623, New York, New York, USA. PMLR.

[Petersen et al., 2008] Petersen, K., Pedersen, M., et al. (2008). The matrix cookbook, vol. 7. *Technical University of Denmark*, 15.