MMD PnP3

Tobias Winter

April 2025

1 Task 1: Logistic Loss

The support vector machine is not the only classifier with a convex loss function. For instance, there is also the so called logistic regression (LR). We consider a multi-class version of logistic regression in which x ∈ R d and y $\in \{1, ..., K\}$, i.e., the input space is d-dimensional and there are K different classes. The probability of predicting class y according to the LR model is

$$p(Y = y|x) = \frac{\exp(\boldsymbol{w}_y^T \boldsymbol{x})}{\sum_{i=1}^K \exp(\boldsymbol{w_i^T x})}$$

where $\boldsymbol{w_1}, ..., \boldsymbol{w}_k \in \mathbb{R}^d$ are the parameters of the model. As a loss function, one typically considers the negative log-likelihood, i.e.,

$$l(\boldsymbol{x}, y; \boldsymbol{w}, ..., \boldsymbol{w}_k) = -log(p(Y = y|x))$$

1.1 (a)

Prove that the negative log-likelihood is convex in $w_1, ..., w_K$. Don't use the fact that the log-sum-exp function is convex. But you can use the fact that functions linear in w_i are convex.

Substitute the function p into the log likelihood

$$l(\boldsymbol{x}, y; \boldsymbol{w}, ..., \boldsymbol{w}_k) = -log\left(\frac{\exp(\boldsymbol{w}_y^T \boldsymbol{x})}{\sum_{i=1}^K \exp(\boldsymbol{w_i^T x})}\right)$$

use $\log(\frac{u}{v}) = \log(u) - \log(v)$

I also assume that we use the natural log here so this also applies $a = \ln(\exp(a))$

$$l(\boldsymbol{x}, y; \boldsymbol{w}, ..., \boldsymbol{w}_k) = -log\left(\frac{\exp(\boldsymbol{w}_y^T \boldsymbol{x})}{\sum_{i=1}^K \exp(\boldsymbol{w_i^T x})}\right)$$

$$-(\log(\exp(\boldsymbol{w}_y^T \boldsymbol{x}) - \log(\sum_{i=1}^K \exp(\boldsymbol{w_i^T x}))))$$

$$-\log(\exp(\boldsymbol{w}_y^T\boldsymbol{x}) + \log(\sum_{i=1}^{K}\exp(\boldsymbol{w_i^T x})))$$

$$-(\boldsymbol{w}_y^T\boldsymbol{x}) + \log(\sum_{i=1}^{K}\exp(\boldsymbol{w_i^T x})))$$

The frist term $\boldsymbol{w}_y^T\boldsymbol{x}$ is a linear function of the form $f(x) = m \cdot x + t$. Linear functions are convex and concave.

In the second term we have the sum of exponentiations. We already know that $\boldsymbol{w}_y^T\boldsymbol{x}$ is convex so $\boldsymbol{w}_i^T\boldsymbol{x}$ is also convex, so taking the exponential of a already convex function gives us again a convex function since the exp function is convex and monotonically increasing. The sum of convex functions is also again convex. If we now take the log, we still have a convex second term in our function. Since the first trem and the second term are both convex, the likelihood function is convex.

1.2 (b)

Suppose $\boldsymbol{x} = 0 \in \mathbb{R}^d$ and $K = 2$ and we observe $y = 1$

$$l(\boldsymbol{x}, y; \boldsymbol{w}_1, \boldsymbol{w}_2) = -\boldsymbol{w}_1\boldsymbol{x} + \log(\exp(\boldsymbol{w}_1^T\boldsymbol{x}) + \exp(\boldsymbol{w}_2\boldsymbol{x}))$$

sub $\boldsymbol{x} = 0$

$$l(\boldsymbol{x}, y; \boldsymbol{w}_1, \boldsymbol{w}_2) = -\boldsymbol{w}_1\mathbf{0} + \log(\exp(\boldsymbol{w}_1^T\mathbf{0}) + \exp(\boldsymbol{w}_2\mathbf{0}))$$

$$l(\boldsymbol{x}, y; \boldsymbol{w}_1, \boldsymbol{w}_2) = -\boldsymbol{w}_1 0 + \log(\exp(\boldsymbol{w}_1^T\mathbf{0}) + \exp(\boldsymbol{w}_2\mathbf{0}))$$

$$l(\boldsymbol{x}, y; \boldsymbol{w}_1, \boldsymbol{w}_2) = \log(1+1) = \log(2)$$

With $\boldsymbol{x} = 0$ the function is constant $\rightarrow$ it is flat $\rightarrow$ it is convex but not strongly convex

2 Task 3: Projection on the L1-norm

Suppose we want to train a classifier or regression model with a 1-norm constraint on the parameter vector to encourage sparsity (and hence improve computation / interpretability). For using projected gradient descent like approaches, we need to compute a projection of the parameter vector $\boldsymbol{w} \in \mathbb{R}^d$ on the 1-norm ball i.e

$$\boldsymbol{w}^P = Proj_S(\boldsymbol{w}) = argmin\|\boldsymbol{w}' - \boldsymbol{w}\|_2$$

where $\boldsymbol{w}^P$ is the projection of $\boldsymbol{w}$, and where $S = \{\boldsymbol{w}' \in \mathbb{R}^d : \|\boldsymbol{w}'\| \leq 1\}$

2.1 (a)

Write down the Karush-Kuhn-Tucker conditions for the projection problem.

If we have a optimization problem such as

$$\max_x f(x) \quad s.t. \quad g(x) = 0$$

1. $g_i(x) \leq 0$ for all i.
2. non negative Lagrange multiplicators $\lambda_i \geq 0$
3. $\lambda_i g_i(x) = 0$ for all i
4. $\nabla f(x) + \sum_i \lambda_i \nabla g_i(x) = 0$

If we now find a optimal solution

$$\max_x f(x) = x^*$$

then x^* is a KKT-point.

If f(x) is now convex and all g_i are also conves then $x^* \Leftrightarrow$ KKT-Point.

2.2 (b)

What insights about the optimal solution can you get from the KarushKuhn-Tucker conditions? Devise an algorithm computing the projection (aim to exploit the insights you made).

As stated in (a) if target function and all constrains are convex in addition to all other KKT conditions, if we then find a KKT-Point, it is a global optimal solution.

Our problem from above is subject to all the contains mentioned. This means if we solve for a KKT-Point, we have a optimal solution, which gives us the projection on or in the l_1-ball.

If we now find a way to project w_i^P into or on the l_1-ball then we know it is subject to all KKT conditions, so we found a optimum. We now just need to find such a projection. Soft thresholding helps us with this. The idea is to snap all entries in w^P to 0 that are small enough and do not contribute to the projection process. For this we need to find a threshold τ. τ will reduce all entries to be either reduced by τ or set to 0. If we do this with all entries, with the right τ, then we found our projection, thous our optimal solution, that is a KKT-Point.

2.3 (c)

Consider a 2-dimensional problem. Compute the projection of the point (2, 1) onto the l_1 unit-ball using your devised algorithm. How does this projection compare to the projection of the same point onto the l_2 unit-ball?

If we use the algorithm form (b) for the vector (2,1) it would go like this:

Algorithm 1 Soft Thresholding

```
Require: : $w \in \mathbb{R}^d$
1: if $\|w\|_1 \leq 1$ then
2:     we is already in or on the $l_1$-ball. No need for projection.
3:     return w
4: end if
5:
6: sort all entries of $w$ such that $|w_1| \geq |w_2| \geq |w_3| \geq ... \geq |w_d|$
7:
8: $\tau_{max} = 0$
9: for k=1,2,..., d do
10:    $v$ = all preceding $w_k$'s
11:    $\tau_k = \frac{w_k+v-1}{k}$
12:    if $\tau_k > \tau_{max}$ then
13:        $\tau_{max} = \tau_k$
14:    end if
15: end for
16:
17: for k=1,2,...,d do
18:    $w_k = w_k$ - $\tau_{max}$
19: end for
20: return w =0
```

1. the vector is already sorted
2. $\|(2,1)\|_1 = 3 > 1$ so we need to run the algorithm.
3. $\tau_1 = \frac{2-1}{1} = 1$
4. $\tau_2 = \frac{2+1-1}{2} = 1$
5. so $\tau = 1$
6. now we need to use τ as our soft threshold
7. $(2-1, 1-1) = (1, 0)$
8. the projection is $(1, 0)$

For the onto the l_2 ball we would just norm the vector by its two norm which is

$$\sqrt{2^2 + 1^2} = \sqrt{5} \quad \rightarrow \left(\frac{2}{\sqrt{5}}, \frac{1}{\sqrt{5}}\right)$$

I don't really know what the key difference here is beside that we lose a entrie in w if we project onto l_1

3 Task 4: Parallel Mini-Batch Gradient Descent

Consider a convex optimization problem where we minimize a convex loss function

$$f(w) = \mathbb{E}_{x\sim D}[l(w; x)]$$

where $w \in \mathbb{R}^d$ is the parameter vector and $l(w; x)$) is a convex loss function with respect to w for any data point x drawn i.i.d. from some distribution D.

In stochastic gradient descent (SGD), at each iteration t, we sample a single data point x_t and update the parameter vector via:

$$w_{t+1} = w_t - \eta_t \nabla f(w_t; x)$$

To reduce variance and make use of parallel computation, we can instead draw a minibatch of B independent samples $\{x_t^{(1)}, ..., x_t^{(B)}\}$ at each step and compute the average gradient:

$$w_{t+1} = w_t - \eta_t \cdot \frac{1}{B}\sum_{i=1}^{B} \nabla l(w_t; x_t^{(i)})$$

3.1 (a)

Show that the variance of the mini-batch gradient estimator is reduced by a factor of 1/B compared to the variance of the single-sample estimator. To this end, let $g_i := \nabla l(w; x^{(i)})$ and assume that the stochastic gradients g_i are iid with:

$\mathbb{E}[g_i] = \nabla f(w)$, and $\mathbb{E}[\|g_i - \nabla f(w)\|^2] = \sigma^2$

We always draw a batch of B samples and then take the average of it. So the estimator would be:

$$\bar{g} = \frac{1}{B}\sum_{i=1}^{B} g_i$$

If we now take the variance of it we get:

$$Var(\bar{g}) = Var(\frac{1}{B}\sum_{i=1}^{B} g_i)$$

NOTE: $Var(X) = \mathbb{E}[(X - \mathbb{E}(X)^2]$ and $Var(cX) = c^2 Var(X)$

$$Var(\bar{g}) = \frac{1}{B^2}\sum_{i=1}^{B} Var(g_i)$$

We know each $Var(g_i) = \mathbb{E}[\|g_i - \nabla f(w)\|^2] = \sigma^2$ and we do this B times, therefore:

$$Var(\bar{g}) = \left(\frac{1}{B^2}\right) \cdot B \cdot \sigma^2 = \frac{\sigma^2}{B}$$

3.2 (b)

Explain how this variance reduction impacts the convergence rate of SGD. Discuss under what conditions it is computationally beneficial to use large mini-batches in parallel implementations.

Hint: Note that a common convergence rate for SGD with step size $\eta_t = \frac{1}{\sqrt{t}}$ bounded variance σ^2 is

$$\mathbb{E}[f(\bar{\boldsymbol{w}}_T)] - f(\boldsymbol{w}^*) \leq O\left(\frac{\sigma}{\sqrt{T}}\right)$$

where $\bar{\boldsymbol{w}}_T$ is the average of iterates, and $\boldsymbol{w}^*$ is the optimal solution.

The variance in SGD can be interpreted as noise while we draw each point x_t. This means that if the variance is high the converges rate is slow. We see this in the Note from above.

We also saw in (a) that if we use mini batching the variance is reduced by a factor of $\frac{1}{B}$, meaning we converge faster by this factor which gives us the new bound of

$$O\left(\frac{\frac{\sigma}{B}}{\sqrt{T}}\right)$$

So we want to choose a large B to reduce the variance, which in turn speeds up the convergence rate, so we need less iterations. But if we choose B to large we lose the benefits of parallelism, so we don't want to choose B to large.

So while in theory if we want the convergence rate to be as fast as possible we want to have a large B. But if we want to benefit from parallelism, we need to choose such a B is not to expansive to be computed in parallel. (try by error i guess)

4 Task 5:Decision Trees

[Reading exercise] Decision trees are commonly used classifiers and provide a humanreadable explanation of how to derive decisions from them.

4.1 (a)

Read and understand sections 12.5.1-12.5.5 of the MMDS book [1]. Prepare an explanation of what decision trees are, how they work, and how they are constructed. Of course, you can use all the material from the book.

All inputs of the same class should reach the same nodes. We also want as few leafs as possible, to be able to classify new data quick. At each decision we make, the group should be pure. Which impurity measure we use, does not matter.

- high impurity $\rightarrow$ classes are mostly mixed
- low impurity $\rightarrow$ most of the examples belong to the same class

4.1.1 Designing a decision three

Goal: the avg. impurity should be as small as possible and the test should have a binary outcome. The test should have two conditions:

1. features to be tested should be numerical so they can be compared with a constant
2. test should also check if input vector is in a possible set

4.1.2 Selecting a test based on numerical features

Steps:

1. sort the data according to the features in ascending order
2. find all gaps (possible splits) between tow features of the same feature class, between two data points. E.g.: Spain has x mio. inhabitants and Italy has y mio.
3. calculate the impurtiy for each split
4. select the split with the least impurity

4.1.3 Selecting a test using categorical features

Now the same. For a node we want to design a test based on a categories.
Steps:

1. assume that there are only two classes for a feature. Like in the book S (soccer) and N (no-Soccer)
2. sort the features based on the accumelated sum of the classes. (how often a feature see the categorie). E.g.: Sothamerica sees S more often then Asia
3. create splits based on the based on the accumelated sums
4. calculate the impurieties of the splitts
5. select the best split.

4.2 (b)

Possible way to parallelizes creation of a decision in a node.

- at each node we need to find the best split by calculating the impurieries. This can be done in parallel.
- we expect that the amount of work at each level is about the same. This is because either a item reached a leaf, and did not go into the next level or it hit a node at the level we now are. So all the nodes at one level can be designed in parallel.
- Grouping of a feature by value can be done in parallel.
- sorting can be done in parallel