Statistical Techniques in Robotics 16-831 Sping 2021 - The Robotics Institute

Homework 2: Online Supervised Learning (Perceptron, SVM, Boosting)

Deadline: March 17th, 2021 11:59 PM

Instructor: Kris Kitani TA: Xingyu Lin
Total points: 100

1 Introduction

In this project, we will deal with online supervised learning techniques, particularly for classification. You will describe the mistake and regret bounds of foundational classification techniques like the Perceptron and AdaBoost. You will also cover some of the basics of convex optimization in subgradients and implement Soft SVM.

1.1 Instructions

Submit this homework on Gradescope. Make sure your Gradescope student account is made using your **Andrew email ID** and your official name in the CMU system - this is essential to correctly record grades. Remember, you can only use a maximum of 2 late days (out of 5 for the semester) for any assignment. Beyond that, you will be penalized by a $1/3^{\text{rd}}$ deduction in points per day. Detailed instructions on what to submit are given in Section 5.

Note: This homework is longer than the previous one. So, again, remember to start **early** as you can only use a maximum of 2 late days (out of 3 for the semester) for any assignment. Beyond that, you will be penalized by a $1/3^{\text{rd}}$ deduction in points per day.

The programming portion of the project is designed to give you some practice with implementing some of these techniques on real data. You are required to use **Python** for this homework. It is mandatory to **type-set** your answers; images of hand-written documents will not be accepted. Detailed instructions on what to submit are given in Section 5.

In the homework folder, besides this problem description, you should also find the following files:

- `mnist_feature_label.npz`

- `perceptron.py`
- `svm.py`

2 Perceptron (45 points)

In this problem, we are going to (1) study the Perceptron in the non-separable setting, (2) study the Perceptron for multi-class classification, and (3) implement the Perceptron.

First, let us summarize the Perceptron we studied in class. Let us define $\mathbf{x}_t \in \mathbb{R}^d$ as our feature vector, $y_t \in \{-1, 1\}$ as our label, and $\mathbf{w}_t \in \mathbb{R}^d$ as our linear separator. Let us define $\mathbf{u}_t = y_t \mathbf{x}_t \mathbf{1}[\hat{y}_t \neq y_t]$, where $\hat{y}_t = sign(\mathbf{x}_t^T \mathbf{w}_t)$ is the prediction the Perceptron will make after seeing the feature $\mathbf{x}_t$. In every iteration t, the Perceptron updates as $\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{u}_t$. And initially, $\mathbf{w}_1 = \mathbf{0}$.

We will use M to denote the number of mistakes the algorithm makes: $M = \sum_{t=1}^{T} \mathbf{1}[\hat{y}_t \neq y_t]$. Throughout this section, we assume that we deal with bounded $\mathbf{x}_t$. Explicitly, $\|\mathbf{x}_t\|_2 \leq R \in \mathbb{R}^+$, for all t and some $R > 0$.

2.1 Perceptron in the Binary Non-Separable Setting

In class, we assumed that our data falls into two classes (i.e., binary classification), and that the data is linearly separable. This means that the two sets of datapoints are organized in such a way that you can place a line (or a generalized line, called a "hyperplane") in between the classes, so that each class lies entirely on one side of the line. In other words, there exists a $\mathbf{w}^*$ such that for all t, $y_t(\mathbf{x}_t^T \mathbf{w}^*) \geq \gamma$, for some $\gamma > 0$. Here, γ is the margin: the larger γ is, the wider the space between the classes.

Linear separability is a very strong assumption. In practice, we usually find that the data is not linearly separable. In this problem, we are going to eliminate this assumption. Surprisingly, without any further modifications to the Perceptron we saw in class, we can achieve a non-trivial mistake bound!

2.1.1 Hinge Loss

We measured the performance of Perceptron using the zero-one loss. However, this loss is usually hard to optimize as it is non-convex and non-continuous. In class we learned that a common trick (*i.e.*, convexification trick) to get around this is to upper bound the zero-one loss by some *surrogate loss*. An example of this for the zero-one loss is the hinge loss:

$$\ell(\mathbf{w}; (\mathbf{x}_t, y_t)) = \max\{0, 1 - y_t \mathbf{x}_t^T \mathbf{w}\} \tag{1}$$

(5 points) Question: Let us study the hinge loss. Show that the following inequality holds regardless of the value of $\hat{y}_t$:

$$\ell(\mathbf{w}; (\mathbf{x}_t, y_t)) \geq \mathbf{1}[\hat{y}_t \neq y_t] - \mathbf{w}^T \mathbf{u}_t. \tag{2}$$

Hint: Consider two cases: (1) $\hat{y}_t \neq y_t$ and (2) $\hat{y}_t = y_t$.

2.1.2 Lower Bound on the Potential Function

Recall the generic strategies for deriving mistake bounds we learned in class. Here let us use the potential function $\Phi(\mathbf{w}_t) = \mathbf{w}_t^T \mathbf{w}^*$, where we assume $\mathbf{w}^*$ is a linear vector with bounded norm $\|\mathbf{w}^*\|_2 \leq D \in \mathbb{R}^+$. Note that this is different from what we learned in class; here we do not assume that $\mathbf{w}^*$ can perfectly separate the data. Namely, there is no $\lambda > 0$ such that $y_t \mathbf{x}_t^T \mathbf{w}^* \geq \lambda$ for all t.

We measure the performance of $\mathbf{w}^*$ using the cumulative hinge loss:

$$L = \sum_{t=1}^{T} \ell(\mathbf{w}^*; (\mathbf{x}_t, y_t)) \tag{3}$$

Using the strategy learned in class and the above potential function, let's try to derive the lower bound. We'll see later how this will be used to derive the mistake/regret bound.

(5 points) Question: Prove the following lower bound for $\Phi(\mathbf{w}_{T+1})$:

$$\Phi(\mathbf{w}_{T+1}) \geq M - L \tag{4}$$

Hint: Use induction with $\mathbf{u}_t^\top \mathbf{w}^* = \Phi(\mathbf{w}_{t+1}) - \Phi(\mathbf{w}_t)$ and the hinge loss inequality (2) from subsection 2.1.1.

2.1.3 Upper Bound of the Potential Function

Next, we also need to derive the upper bound of the potential function. The first step to upper bound the potential function is to apply Cauchy-Schwartz inequality on $\Phi(\mathbf{w}_{T+1})$:

$$\mathbf{w}_{T+1}^\top \mathbf{w}^* \leq \|\mathbf{w}_{T+1}\|_2 \|\mathbf{w}^*\|_2 \leq D \|\mathbf{w}_{T+1}\|_2, \tag{5}$$

where we assume that $\|\mathbf{w}^*\|_2 \leq D$. So, if we derive an upper bound on $\|\mathbf{w}_{T+1}\|_2$, we will also have an upper bound on $\Phi(\mathbf{w}_{T+1})$."

(5 points) Question: Show that $\|\mathbf{w}_{T+1}\|_2^2$ is linearly proportional to the number of mistakes M. Specifically, show that:

$$\|\mathbf{w}_{T+1}\|_2^2 \leq MR^2. \tag{6}$$

Hint: Think about the sign of $\mathbf{w}_t^\top \mathbf{u}_t$ and use induction with $\mathbf{w}_{T+1} = \mathbf{w}_T + \mathbf{u}_T$.

2.1.4 Chain the Upper and Lower Bounds

Now, we are ready to derive the mistake bound. Let us chain the upper and lower bounds we derived together.

(10 points) Question: Derive the mistake bound. It should look like this:

$$M \leq L + O(\sqrt{L}) + O(1) \tag{7}$$

The O hides constants that do not depend on L and T.

Hint: You need to solve a quadratic inequality. You may find this following inequality will be useful: $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}, \forall a \geq 0, b \geq 0$

To connect this to the linearly separable setting, note that when $\mathbf{w}^*$ separates $(\mathbf{x}_t, y_t)$ with margin $\lambda = 1$, then $L = 0$.

2.2 Multi-class Perceptron

Multi-class classification is a natural extension of binary classification and is quite common in practice. As usual, let us define our features $\mathbf{x}_t \in \mathbb{R}^d, \forall t$, and labels $y_t \in \{1, 2, 3, .., k\}$. We use the notation $\mathbf{W} \in \mathbb{R}^{k \times d}$ to represent a matrix. $\mathbf{W}^i$ represents the i^{th} row of the matrix $\mathbf{W}$ and $\mathbf{W}^{i,j}$ represents the entry in i^{th} row and j^{th} column.

The multi-class Perceptron is summarized below. Start with $\mathbf{W}_1 = \mathbf{0}$. Every iteration t, after we receive $\mathbf{x}_t$, we predict:

$$\hat{y}_t = \arg \max_{j \in 1,2,..,k} \mathbf{W}_t^j \mathbf{x}_t, \tag{8}$$

where $\mathbf{W}_t^j$ is a row vector and $\mathbf{x}_t$ is a column vector. Basically, we are trying out each of the linear classifiers (every row of $\mathbf{W}_t$) and taking the one with the greatest response. After we receive the true label y_t, we form $\mathbf{U}_t \in \mathbb{R}^{k \times d}$ as follows:

$$\mathbf{U}_t^{i,j} = \mathbf{x}_{t,j}(\mathbf{1}[y_t = i] - \mathbf{1}[\hat{y}_t = i]) \tag{9}$$

where $\mathbf{x}_{t,j}$ means the j^{th} element in the vector $\mathbf{x}_t$. Now let us look into the details of $\mathbf{U}_t$. We can verify that when $\hat{y}_t = y_t$ (i.e., no mistake), $\mathbf{U}_t$ is a matrix with zero in all entries. When there is a mistake (i.e., $\hat{y}_t \neq y_t$), $\mathbf{U}_t$ consists of zeros except that the y_t^{th} row of $\mathbf{U}_t$ is $\mathbf{x}_t$ and the $\hat{y}_t^{th}$ row is $-\mathbf{x}_t$.

We update $\mathbf{W}_{t+1} = \mathbf{W}_t + \mathbf{U}_t$, and then move to the next iteration.

2.2.1 Understanding Multi-class Hinge Loss

In the binary setting, we define the *margin* as $(y\mathbf{w}^\top \mathbf{x})$ for any predictor $\mathbf{w}$ and any pair of $(\mathbf{x}, y)$. The margin is the distance from the example $\mathbf{x}$ to the decision boundary. If margin is positive, then the prediction $\hat{y}_t$ is correct. The larger the margin, the more confident we are about the prediction.

Below is one possible definition of a multi-class hinge loss:

$$\ell(\mathbf{W}; (\mathbf{x}_t, y_t)) = \max\{0, 1 - (\mathbf{W}^{y_t}\mathbf{x}_t - \max_{j \in \{1,2,..,k\} \setminus \{y_t\}} \mathbf{W}^j \mathbf{x}_t)\}, \tag{10}$$

where $\{1, 2, ..., k\} \setminus \{y_t\}$ represents the set of all labels excluding the label y_t.

(5 points) Question: Explain why this loss function make sense: discuss what the loss would be if the algorithm does not make a mistake, and explain if it upper bounds the (non-convex) zero-one loss.

2.2.2 Connecting Hinge Loss and Update Rule (Bonus)

Recall that the update rule is $\mathbf{W}_{t+1} = \mathbf{W}_t + \mathbf{U}_t$. To understand this, we can compute the subgradient of $\ell(\mathbf{W}; (\mathbf{x}_t, y_t))$ with respect to $\mathbf{W}$ measured at $\mathbf{W}_t$. The subgradient of the hinge loss is exactly $-\mathbf{U}_t$. Hence, you can understand the update rule as doing gradient descent with a constant learning rate 1.

Well, the intuition mentioned above is not quite accurate, as simply using the gradient descent will not give us the desired mistake bound. If gradient descent is considered, you would be required to have a decaying learning rate, but we know that Perceptron uses a constant learning rate 1.

(Bonus: 15 points) Question: Recall what we did above for the binary, non-separable setting. Derive a mistake bound for the multi-class Perceptron. *Hint:* Replace $\mathbf{w}_t$ with $\mathbf{W}_t$, $\mathbf{u}_t$ with $\mathbf{U}_t$, the vector inner product $\mathbf{a}^\top \mathbf{b}$ with the matrix inner product $\langle \mathbf{A}, \mathbf{B} \rangle = \sum_{i,j} \mathbf{A}^{i,j} \mathbf{B}^{i,j}$, and the binary hinge loss with the multi-class hinge loss.

2.3 Implementing Perceptron

(15 points) Code: Using the template code in `perceptron.py`, implement the multi-class perceptron. The code indicates which parts need filling out.

Test your code on the data in `mnist_feature_label.npz`. This data consists of $70,000$ samples, which are HoG features of the MNIST images. Generate a plot with time on the x-axis, from $t = 1$ to $T = 70000$, and M_t/t on the y-axis, where M_t is the number of mistakes you have made up to and including step t. If the perceptron is training correctly, the average number of mistakes should decline over time.

Only import numpy and matplotlib.

Submit your completed `perceptron.py` and show the plot in your writeup.

3 AdaBoost (20 points)

In this question, we will look at a slightly different version of AdaBoost than what was described in class, and derive its error bound. The algorithm remains largely the same as before, except with the update of the weights. We are given:

1. Dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_M, y_M)\}$, where $\mathbf{x}_m \in \mathbb{R}^n$ and $y_m \in \{-1, 1\}$.
2. Weak learner $\mathcal{W}$ which takes the dataset $\mathcal{D}$ and weights $\mathbf{p}$ over points to produce hypothesis $h = \mathcal{W}(\mathcal{D}, \mathbf{p})$. Here, $\mathbf{p}(m)$ corresponds to the weight of the point $(\mathbf{x}_m, y_m)$. The weights must sum to 1, i.e. $\sum_m \mathbf{p}(m) = 1$.

The algorithm is as follows:

`Initialize:` $\mathbf{p}_1(m) = \dfrac{1}{M}$ for $m = 1, \ldots, M$

`For` $t = 1, \ldots, T$:

`Generate hypothesis:` $h_t = \mathcal{W}(\mathcal{D}, \mathbf{p}_t)$

`Receive weighted error:` $\epsilon_t = \sum_{m=1}^{M} \mathbf{p}_t(m) \cdot \mathbb{1}(y_m \neq h_t(\mathbf{x}_m))$

`Reweight:` $\mathbf{p}_{t+1}(m) = \dfrac{\mathbf{p}_t(m)}{Z_t} \times \begin{cases} \exp(-\beta_t) \ \texttt{if} \ \ y_m = h_t(\mathbf{x}_m) \\ \exp(\beta_t) \ \texttt{if} \ \ y_m \neq h_t(\mathbf{x}_m) \end{cases}$

`Final hypothesis:` $h_F(\mathbf{x}) = sign\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)$

In the `Reweight` step, Z_t is a normalization constant to make $\mathbf{p}_{t+1}$ sum to 1 and $\beta_t = \epsilon_t/(1-\epsilon_t)$ is a weighting factor. We want to show that the error ϵ on running the final classifier h_F on the dataset $\mathcal{D}$ is bounded:

$$\epsilon = \frac{1}{M} \sum_{m=1}^{M} \mathbb{1}(y_m \neq h_F(\mathbf{x}_m)) \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t(1-\epsilon_t)} \tag{11}$$

3.1 Bounding the weight of a single point

First, let us take a look at the weight of a single point. If the final classifier h_F makes a mistake on a point, then we know that the (weighted) majority of classifiers $h_1, \ldots, h_T$ have made a mistake on that point. This means that the point's weight must be large. To make this idea explicit, let's prove a lower bound on such a point's weight.

(10 points) Question:

- Take $f(\mathbf{x}) = \sum_{t=1}^{T} \beta_t h_t(\mathbf{x})$. Show that

$$p_{T+1}(m) = \frac{1}{M} \frac{\exp(-y_m f(x_m))}{\prod_{t=1}^{T} Z_t}. \tag{12}$$

- Find a lower bound for $p_{T+1}(i)$ of point $(\mathbf{x}_i, y_i)$ which h_F gets wrong.

Hint: Rewrite the weight update as one equation with y_m and $h_t(\mathbf{x}_m)$.

3.2 Bounding error using normalizing weights

We now have a bound on the size of the weight of a mistaken point. Since the weights sum to 1, we can't have too many points with a high weight.

(5 points) Question: Show that error of h_F is bounded by

$$\epsilon \leq \prod_{t=1}^{T} Z_t. \tag{13}$$

Hint: (1) What is the relationship between $\mathbb{1}(\alpha \leq 0)$ and $\exp(-\alpha)$? (2) Use Eqn. 12 to simply derivation.

3.3 Choosing β_t to minimize error bound

Finally, we want to show that $\prod_{t=1}^{T} Z_t \to 0$, if we select β_t appropriately. This means that the total number of mistaken points grows smaller.

(5 points) Question: Find β_t that minimizes ϵ_t every iteration and show that the corresponding $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$. Combine this with the previous parts to get the final bound.
Hint: The errors ϵ_t can be written in terms of the weights $\mathbf{p}_t(m)$. Use this to write Z_t in terms of ϵ_t.

4 Soft SVM (35 points)

In this question, we will study SVMs. As we saw in the previous question, AdaBoost is an ensemble approach; SVMs, in contrast, are a max-margin approach.

Here we will look at a special kind of SVM, called an online SVM. Online approaches are especially useful when it is infeasible to look at all the data at once. This could be because you have a lot of data, or perhaps because you receive the data points in an ongoing stream.

The usual objective for an SVM is:

$$\min_{\mathbf{w}} \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{M}\sum_{m=1}^{M} \max\{0, 1 - y_m \mathbf{w}^\top \mathbf{x}_m\}$$

4.1 Understanding hyperplanes

The SVM objective includes the term $\mathbf{w}^\top \mathbf{x}_m$ as well $||\mathbf{w}||^2$. These terms are related to hyper-planes. A hyper-plane in n dimension space can be described by the equation $\mathbf{w}^T x + b = 0$, where $\mathbf{w}, \mathbf{x} \in \mathbb{R}^n$.
(5 points) Question: Derive the distance from the origin to this the hyperplane.
Hint: Write out the distance in a form of constrained optimization and use the Lagrange multiplier method.

4.2 Subgradients

As we saw in class, the SVM objective is non-differentiable. An approach like gradient descent cannot be applied to this objective, because gradients do not exist everywhere. Instead, we can use subgradients! So, we need to learn about sub-gradients.

A subgradient of a function f at a point $x \in \mathbb{R}^n$ is any $g \in \mathbb{R}^n$ such that:

$$f(y) \geq f(x) + g^\top (y - x) \;\; \forall \;\; y \in \mathbb{R}^n$$

In other words, subgradients specify the lines which lower-bound the function at a given point.

4.2.1 Existence of Subgradients

Subgradients are useful for optimizing convex objective functions because they always exist for every point in the domain, even if they are not unique.

The set of all subgradients of a function at x is called the differential set or the subdifferential, denoted by $\partial f(x)$. If the function is differentiable at x, then the subgradient is unique and is the same as the gradient $\nabla f(x)$. But what about subgradients of non-convex functions?

(5 points) Question: Consider a non-convex function f. Do subgradients always exist for all points in the domain of f? Prove this if true or give a counter example if false. If the answer is false, is it possible for a non-convex function to have a subgradient at some point, even if not all points?

4.2.2 Finding subgradients

For gradient descent, you need to have access to the gradient of the function at the points of evaluation. Similarly, subgradient methods require you to have access to the subgradients of the function. So let us look at some simple examples of subgradients.

(5 points) Question: Find the differential sets of the following functions:

1. $f(x) = \max\{f_1(x), f_2(x)\}$, where f_1 and f_2 are convex functions
2. $f(x) = ||x||_2 = \sqrt{\sum_i x_i^2}$

   Hint: Here, f isn't differentiable at $x = 0$. Treat this case separately using the definition of a subgradient.

4.3 Implementing the Soft SVM

(20 points) Code: Implement the Soft SVM discussed in class, and train for classification on the following dataset: http://www.cs.cmu.edu/ 16831-f14/homework/Lab2-online_learning/data.zip.

The data consists of 3D pointclouds from Oakland, and some features of these pointclouds[1]. You may want to visualize the pointclouds to see what you

[1] See https://www.cs.cmu.edu/ vmr/datasets/oakland_3d/cvpr09/doc/)

are working with, but train your SVM on the features. Do not distribute this data without permission. Features are provided courtesy of Dan Munoz.

The five classes in the dataset are:

- 1004: Veg
- 1100: Wire
- 1103: Pole
- 1200: Ground
- 1400: Facade

Since there are 5 classes, you need to implement a one-vs-all classifier for each class. In other words, for each class, treat the target class as positive, and all the remaining classes as negative. Then, assemble these into a multi-class classifier: feed each input to all classifiers, collect all responses, and the argmax of these should tell you the input's class. (Since each classifier votes for its own class, the classifier with the strongest response should correspond to the correct class.)

The file `svm.py` has code for reading this data and splitting it into train and test sets, along with template functions for `train_svm`, `eval_svm`, and `eval_multi_svm`.

After completing the implementation, answer the following questions:

1. What was the algorithm's accuracy on each class? (Paste the output of your code.)
2. What were your hyperparameter choices? Explain these choices.
3. How long does the algorithm take (in terms of number of points, classes, iterations, feature dimensions, etc.) to train and predict?

Include your code in your submission.

5 What to Submit

Your submission should consist of:

1. a zip file named `<AndrewId>.zip` consisting of a folder `code` containing all the code and data (if any) files you were asked to write and generate. Submit this to `Homework 2 - Programming` on Gradescope.
2. a `pdf` named `<AndrewId>.pdf` with the answers to the theory questions and the results, explanations and images asked for in the coding questions. **It is compulsory to type-set your answers; images of handwritten documents will not be accepted**. Submit this to `Homework 2 - Theory` on Gradescope. For easier grading, please use the Gradescope feature for tagging the pages and position for each question. 5 style points will be deducted from this homework if this is not followed.

6 Academic Integrity

You are encouraged to work together BUT you must do your own work (code and write up). If you work with someone, you must include their name in your write up and inside any code that has been discussed. If we find highly identical write-ups or code without proper accreditation of collaborators, we will take action according to university policies. If you use someone's material, you must give proper credit by including a citation where necessary