Statistical Techniques in Robotics 16-831 - The Robotics Institute

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

Deadline: Oct 15, 2020 11:59 PM EST (submit via Canvas)

1 Introduction

In this assignment, we will deal with online supervised learning techniques, particularly for classification.

The theoretical goal of this project is to familiarize you with the mistake and regret bounds behind some of the foundational classification techniques like the Perceptron and AdaBoost. We will also briefly touch some of the basics of convex optimization in subgradients.

The programming portion of the project is designed to give you some practice with implementing some of these popular techniques on real datasets. You are required to use Python for this homework.

1.1 Instructions

Submit this homework on Canvas. Remember, you can only use a maximum of 2 late days (out of 5 for the semester) for any assignment. Detailed instructions on what to submit are given in section 5.

2 Perceptron (40 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 $\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{u}_t$.

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 Non-Separable Setting

In class, we assume that the data is linear separable: 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 margin: the larger γ is, the more confident the prediction is. However, the linear separability assumption is fairly strong and in practice we most likely will encounter situations where the data is not linear separable.

In this problem, we are going to eliminate this assumption. Surprisingly, without any further modifications to the Perceptron we saw class, we can achieve a non-trial 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. A common approach 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 pefectly seperate 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 Inequality 2.

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 $\|\mathbf{w}^*\|_2 \leq D$. Hence, to upper bound $\Phi(\mathbf{w}_{T+1})$, we only need to upper bound $\|\mathbf{w}_{T+1}\|_2$.

(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.

(5 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: $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}, \forall a \geq 0, b \geq 0$

2.1.5 Conclusion

As you might have noticed, we are comparing the mistake bound M, which is the cumulative zero-one loss, with the cumulative hinge loss of $\mathbf{w}^*$. Namely, we are using different loss functions on the two sides of the inequality. Comparing the mistake bound M to the mistake bound of $\mathbf{w}^*$ in general is hard without any assumptions, as the zero-one loss is non-convex. To connect it to the linear separable setting, note that when $\mathbf{w}^*$ linear 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 and if margin is larger, then we are more confident 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 $\{1, 2, ..., k\}\setminus\{y_t\}$ stands for the set of all labels excluding the label y_t.

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

2.2.2 Connecting Hinge Loss and Update Rule

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, intuitively you can understand the update rule as doing Gradient Descent with a constant learning rate 1.

2.2.3 A Formal Proof (Bonus)

The intuition that mentioned above is not quite correct, as simply using the gradient descent analysis will not give us the mistake bound we want, and this analysis generally requires a decay of the learning rate. But the Perceptron uses a constant learning rate 1.

(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: Implement the multi-class Perceptron. We have given you a pre-processed version of the MNIST dataset with HoG features. Test your algorithm on this dataset.

The dataset consists of 70000 samples. Generate a plot, whose x-axis is t ranging from $t = 1$ to $T = 70000$, and y-axis is M_t/t, where M_t is the number of mistakes you have made up till and including step t.

Please see the `multi_perceptron.py` template for more details. You task is to fill in the missing details in the two functions: `multi_perceptron_predict` and `multi_perceptron_update`. All you need for this is numpy. Do not use any other existing machine learning libraries such as scikit-learn. The template provides code for reading the dataset and maintaining prediction statistics already.

Submit your completed `multi_perceptron.py` and the plot.

3 AdaBoost (25 points)

Here is a re-cap of the AdaBoost algorithm introduced in class:

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: $\mathbf{p}_1(m) = \frac{1}{M}$ 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: $\mathbf{p}_{t+1}(m) = \frac{\mathbf{p}_t(m)}{Z_t} \times \begin{cases} \exp(-\beta_t) \text{ if } y_m = h_t(\mathbf{x}_m) \\ \exp(\beta_t) \text{ if } y_m \neq h_t(\mathbf{x}_m) \end{cases}$
    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 β_t is a weighting factor that depends on ϵ_t. We want to show that the error ϵ on running the final classifier h_F on the dataset $\mathcal{D}$ is bounded:

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

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 its weight must be large. We can make this explicit.

(5 points) Question: Take $f(\mathbf{x}) = \sum_{t=1}^{T} \beta_t h_t(\mathbf{x})$. Show that $p_{T+1}(m) = \frac{1}{M} \frac{\exp(-y_m f(x_m))}{\prod_{t=1}^{T} Z_t}$. 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 as follows: $\epsilon \leq \prod_{t=1}^{T} Z_t$.
Hint: What is the relationship between $\mathbb{1}(\alpha \leq 0)$ and $\exp(-\alpha)$?

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.

3.4 How many iterations should you run AdaBoost?

Weak learners are typically associated with a parameter $\gamma > 0$ which represents their ability to do better than random guessing. More explicitly, a hypothesis h by weak learner $\mathcal{W}$ has an error over some data distribution $\mathcal{P}$ given by:

$$Pr\left[err_{\mathcal{P}}(h) \leq \frac{1}{2} - \gamma\right] \geq 1 - \delta$$

for some $\delta > 0$. This means that, with a large probability $1-\delta$, the hypothesis h does better than random guessing by γ. We can make δ smaller (or equivalently, $1 - \delta$ closer to 1) with more observations. In general, we can make δ as small as we want given enough observations.
(10 points) Question: Assume that in every iteration t, $\mathcal{W}$ produces a hypothesis h_t with error $\epsilon_t = \frac{1}{2} - \gamma_t$ such that $\gamma_t \geq \gamma$; this is the same as assuming $\delta = 0$. Find an upper bound on the number of iterations T to run AdaBoost to get the final error ϵ less than some constant ϵ_0. Your answer should be in terms of γ and ϵ_0.
Hint: Rewrite the bound on ϵ from the previous part in terms of γ_t and make use of the assumption. This inequality might be useful: $1 + x \leq e^x$.

4 Soft SVM (35 points)

We have looked at AdaBoost, an ensemble learning approach which combines the predictions of a sequence of weak learners in order to classify. This weak learner could be a decision stump, a decision tree, an SVM, etc. For this question, we will take a look at an online version of the SVM, which is a max-margin classifier.

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 you only can see the data points one at a time, in the case of streaming data.

The usual objective for an SVM is:

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

As we saw in class, this objective is non-differentiable because of the hinge-loss term. An approach like gradient descent is not feasible to optimize this objective as the gradients might not always exist. Instead, we can use subgradients! So, let us start with a quick primer of subgradients.

4.1 Subgradients

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.

4.1.1 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?

4.1.2 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.

4.2 Implementing the Soft SVM

(25 points) Code: For this part, you will implement the Soft SVM discussed in class. You can find the dataset here: http://www.cs.cmu.edu/ 16831-f14/homework/Lab2-online_learning/data.zip

They are 3D point-clouds of Oakland[1]. You should not distribute the data without permission. Features are provided courtesy of Dan Munoz. The five classes in the dataset are:

- 1004: Veg

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

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

Since there are 5 classes, you would need to implement a multi-class version of the soft SVM. The simplest thing to do here would be to implement a one-vs-rest classifier for each class, where you take the relevant class as positive and all the remaining classes are negative. Then, aggregate the results of the classifiers by choosing the label as the class which is most confidently classified.

There should be two data files. Create a train and test set by aggregating all the data together and then randomly splitting them into two subsets (say 80% for train and 20% for test).

Your write up should include answers and results to the following:

1. Include a confusion matrix of the results.
2. How well did the algorithm perform? Were there any classes which were tough to classify? Why do you think that is?
3. Include an image of the point-clouds of the classified data.
4. Explain your choices of hyper-parameters.
5. How long does the algorithm take (in terms of number of points, classes, iterations, feature dimensions, etc.) to train and predict?

Include the code in your submission.

5 What to Submit

Submit this homework on Canvas. Your submission should consist of a single zip file named `<AndrewId>.zip`. This zip file should contain:

- a folder `code` containing all the code and data (if any) files you were asked to write and generate.
- a `pdf` named `writeup.pdf` with the answers to the theory questions and the results, explanations and images asked for in the coding questions. Read the write up carefully before your submit. You will lose points if you do not include everything. Feel free to add extra images and figures if you think they are useful for better explaining your answers.
- If you attempted the bonus questions, remember to indicate in **bold** in your writeup.