EECS 189 Introduction to Machine Learning
Fall 2020

HW11

This homework is due **Tuesday, November 17 at 11:59 p.m.**

1 Getting Started

Read through this page carefully. You may typeset your homework in latex or submit neatly handwritten/scanned solutions. Please start each question on a new page. Deliverables:

1. Submit a PDF of your writeup, **with an appendix for your code**, to the appropriate assignment on Gradescope. If there are graphs, include those graphs in the correct sections. Do not simply reference your appendix.

2. If there is code, submit all code needed to reproduce your results.
3. If there is a test set, submit your test set evaluation results.

After you've submitted your homework, watch out for the self-grade form.

(a) Who else did you work with on this homework? In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

(b) Please copy the following statement and sign next to it. We just want to make it *extra* clear so that no one inadvertently cheats.

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions nor have I looked at any online solutions to any of these problems. I have credited all external sources in this write up.

2 SVM with custom margins

In the lecture, we covered the soft-margin SVM. The objective to be optimized over the training set $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$ is

$$\min_{\mathbf{w},b,\xi_i} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n} \xi_i \tag{1}$$

$$s.t. \quad y_i(\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 - \xi_i \quad \forall i \tag{2}$$

$$\xi_i \geq 0 \quad \forall i \tag{3}$$

In this problem, we are interested in a modified version of the soft-margin SVM where we have a custom margin for each of the n data points. In the standard soft-margin SVM, we pay a penalty of ξ_i for each of the data point. In practice, we might not want to treat each training point equally, since with prior knowledge, we might know that some data points are more important than the others. (This is related to weighted least squares, where we give each data point an importance or where we have a prior belief that some points are more noisy than others.)

We formally define the following optimization problem:

$$\min_{\mathbf{w},b,\{\xi_i\}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\phi_i\xi_i \tag{4}$$

$$s.t. \quad y_i(\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 - \xi_i \quad \forall i \tag{5}$$

$$\xi_i \geq 0 \quad \forall i \tag{6}$$

Note that the only difference is that we have a custom weighting factor $\phi_i > 0$ for each of the slack variables ξ_i in the objective function. These ϕ_i are some constants given by the prior knowledge, and thus they can be treated as known constants in the optimization problem. Intuitively, this formulation weights each of the violations (ξ_i) differently according to the prior knowledge (ϕ_i).

(a) For the standard soft-margin SVM, we have shown that the constrained optimization problem is equal to the following unconstrained optimization problem, i.e. regularized empirical risk minimization problem with hinge loss:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i - b)) \tag{7}$$

What's the corresponding unconstrained optimization problem for the SVM with custom margins?

(b) The dual of the standard soft-margin SVM is:

$$\max_{\alpha} \alpha^\top \mathbf{1} - \frac{1}{2}\alpha^\top \mathbf{Q}\alpha \tag{8}$$

$$s.t. \sum_{i=1}^{n}\alpha_i y_i = 0 \tag{9}$$

$$0 \leq \alpha_i \leq C \quad i = 1, \cdots, n \tag{10}$$

where $\mathbf{Q} = (\operatorname{diag}\mathbf{y})\mathbf{X}\mathbf{X}^T(\operatorname{diag}\mathbf{y})$

What's the dual form of the SVM with custom margins? Show the derivation steps in detail.

(c) **From the dual formulation above, how would you kernelize the SVM with custom margins? What role does the ϕ_i play in the kernelized version?**

3 Kernel Logistic Regression

We have seen in lecture how to kernelize ridge regression, and by going to the dual formulation, how to kernalize soft-margin SVMs as well. Here, we will consider how to kernelize logistic regression and compare its performance to kernelized SVMs using a real-world dataset.

(a) Imagine we have n different d-dimensional data points in an $n \times d$ matrix $\mathbf{X}$, with associated labels in an n-dimensional vector $\mathbf{y}$. Let this be a binary classification problem, so each label $y_i \in \{0, 1\}$. Remember, this means that each training data point is associated with a row in the matrix $\mathbf{X}$ and the vector $\mathbf{y}$.

Recall that logistic regression associates a point $\mathbf{x}$ with a real number from 0 to 1 by computing:

$$f(\mathbf{x}) = \frac{1}{1 + \exp\{-\mathbf{w}^T\mathbf{x}\}},$$

This number can be interpreted as the estimated probability for the point $\mathbf{x}$ having a true label of $+1$. Since this number is $\frac{1}{2}$ when $\mathbf{w}^T\mathbf{x} = 0$, the sign of $\mathbf{w}^T\mathbf{x}$ is what predicts the label of the test point $\mathbf{x}$.

As you've seen in earlier homeworks, the loss function is defined to be

$$L(\mathbf{w}) = \sum_i -y_i \ln(f(\mathbf{x}_i)) - (1 - y_i)\ln(1 - f(\mathbf{x}_i)), \tag{11}$$

where the label of the ith point $\mathbf{x}_i$ is y_i.

Write down the gradient-descent update step for logistic regression, with step size γ. Assume that we are working with the raw features $\mathbf{X}$ for now, with no kernelization.

For convenience, define the logistic function $s(\cdot)$ to be

$$s(x) = \frac{1}{1 + e^{-x}}. \tag{12}$$

(b) You should have found that the update $\mathbf{w}^{(t+1)} - \mathbf{w}^{(t)}$ is a linear combination of the observations $\{\mathbf{x}_i\}$. This suggests that gradient descent for logistic regression might be compatible with the kernel trick. After all, this is the same thing that we saw when we were doing least-squares iteratively by gradient descent, and that was certainly something that we could kernelize.

When we argued for the kernelization of least-squares, we did so by means of the augmented-features view of ridge regression. That had the pedagogic advantage of helping you internalize the importance of norm-minimization and made for an argument by which you could naturally discover the kernelization for yourself. Here, we will pursue a more direct path that has an inductive feeling to it.

Imagine that we start with some weight vector $\mathbf{w}^{(t)} = \mathbf{X}^T\mathbf{a}^{(t)}$ that is a linear combination of the $\{\mathbf{x}_i\}$. (Notice that if we start with the zero vector as our base case, this is true for the base case.) **Show that after one gradient step, $\mathbf{w}^{(t+1)}$ will remain a linear combination of the $\{\mathbf{x}_i\}$, and so is expressible as $\mathbf{X}^T\mathbf{a}^{(t+1)}$ for some "dual weight vector" $\mathbf{a}^{(t+1)}$. Then write down the gradient-descent update step for the dual weights $\mathbf{a}^{(t)}$ directly without referring to $\mathbf{w}^{(t)}$ at all.** In other words, tell us how $\mathbf{a}^{(t+1)}$ is obtained from the data, the step size, and $\mathbf{a}^{(t)}$.

(c) You should see from the previous part that the gradient-descent update step for $a^{(t)}$ can be written to depend solely on $\mathbf{X}\mathbf{X}^T$, not on the individual $\{\mathbf{x}_i\}$ in any other way. Since $\mathbf{X}\mathbf{X}^T$ is just the Gram matrix of pairwise inner-products of training point inputs, this suggests that we can use the kernel trick to quickly compute gradient steps for $\mathbf{a}^{(t)}$ so long as we can compute the inner products of any pair of featurized observations.

Now suppose that you just have access to a similarity kernel function $k(\mathbf{x}, \mathbf{z})$ that can be understood in terms of an implicit featurization $\boldsymbol{\phi}(\cdot)$ so that $k(\mathbf{x}, \mathbf{z}) = \boldsymbol{\phi}(\mathbf{x})^T\boldsymbol{\phi}(\mathbf{z})$. **Describe how you would compute gradient-descent updates for the dual weights $\mathbf{a}^{(t)}$ as well as how you would use the final weights together with the training data to classify a test point $\mathbf{x}$.**

Note: You do not have access to the implicit featurization $\boldsymbol{\phi}(\cdot)$. You have to use the similarity kernel $k(\cdot, \cdot)$ in your final answer.

(d) You have now derived kernel logistic regression! Next, we will study how it relates to the kernel SVM, which we will do numerically. **Complete all the parts in the associated Jupyter notebook.**

4 Adversarial Examples

In this problem, suppose that we are learning parameters $\boldsymbol{\theta}$ by trying to minimize the following robust optimization problem on training data:

$$\min_{\boldsymbol{\theta}} \left(\frac{1}{n} \sum_{i=1}^{n} \max_{\boldsymbol{\delta}_i \in \mathcal{S}} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i + \boldsymbol{\delta}_i), y_i) \right). \tag{13}$$

where $\mathbf{x}_i \in \mathbb{R}^d$ is the i-th input and $y_i \in \{-1, +1\}$ is the binary label of the i-th data. The loss $\ell(\cdot, \cdot)$ could be square loss or logistic loss. This looks familiar, except that there is this $\boldsymbol{\delta}$ hanging around together with the inner maximization. Why?

In general, we might be concerned about how robust we are if an adversary can perturb our inputs. For example, consider the logistic loss with such a perturbation

$$\ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i + \boldsymbol{\delta}_i), y_i) = \ln\Big(1 + \exp(-y_i\langle \boldsymbol{\theta}, \mathbf{x}_i + \boldsymbol{\delta}_i\rangle)\Big)$$

where $\boldsymbol{\delta}_i \in \mathcal{S}$ is the perturbation on $\mathbf{x}_i$.

What we ideally want is for the adversary to not be able to cause us errors at test time by introducing such perturbations. One approach to trying to do so is to at least try to guarantee robustness to such perturbations on the training set. Our goal here becomes finding model parameters $\boldsymbol{\theta}$ such that they minimize the loss function on a *worst case perturbation* of the training dataset. That is what (13) is trying to achieve.

To have the problem be meaningful, we must constrain the adversary in some manner otherwise the problem is hopeless. This is where the contraint set $\mathcal{S}$ comes in. If we set this to $\mathcal{S} = \{\mathbf{0}\}$, then the problem (13) reduces to the standard training problem. To add some robustness, we could set the perturbation set to be $\mathcal{S} = \{\boldsymbol{\delta} : \|\boldsymbol{\delta}\| \leq \epsilon\}$, where ϵ is small compared with the size of typical training data points. (Note that such an approach to adversarial robustness is somewhat naive in practice, as has been recently pointed out in a paper `arxiv.org/abs/2006.12655` by Laidlaw, Singla, and Feizi. This is because even large perturbations to inputs might be imperceptable to humans. But it is a start, and you have to start somewhere.)

Recall that the motivation of training models on worst case perturbations is we want our model to provide "robust" prediction. In other words, for test data $\mathbf{x}_{\text{test}}$, the learned model f_θ should predict consistently over the whole perturbation set around $\mathbf{x}_{\text{test}}$, i.e.,

$$f_\theta(\mathbf{x}_{\text{test}}) \approx f_\theta(\mathbf{x}_{\text{test}} + \boldsymbol{\delta}_{\text{test}}), \quad \forall \boldsymbol{\delta}_{\text{test}} \in \mathcal{S}.$$

In practice, there exist data points, called adversarial examples, which in the context of images or videos are visually indistinguishable from normal examples but can fool/mislead machine-learning algorithms. This has been found to be quite a ubiquitous phenomenon and impacts audio data as well as textual data. Because it impacts all the kinds of data that humans have direct perceptual intuitions about, this phenomenon is believed to also impact machine learning on data sources that humans have no intuition about.

This problem is designed to help introduce you to this important phenomenon.

(a) One way of constraining the perturbation is to bound it in each entry. This is referred to as the infinity-norm so $\|\boldsymbol{\delta}\|_\infty \le \epsilon$ means that $|\delta[j]| \le \epsilon$ for all j.

For the linear model with logistic loss, **prove the following**:

$$\frac{1}{n}\sum_{i=1}^{n} \max_{\|\boldsymbol{\delta}_i\|_\infty \le \epsilon} \ln\left(1 + \exp(-y_i\langle\boldsymbol{\theta}, \mathbf{x}_i + \boldsymbol{\delta}_i\rangle)\right) = \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + \exp(-y_i\langle\boldsymbol{\theta}, \mathbf{x}_i\rangle + \epsilon\|\boldsymbol{\theta}\|_1)\right), \qquad (14)$$

and along the way **prove**

$$\boldsymbol{\delta}_i^\star = \arg\max_{\|\boldsymbol{\delta}_i\|_\infty \le \epsilon} \log\left(1 + \exp(-y_i\langle\boldsymbol{\theta}, \mathbf{x}_i + \boldsymbol{\delta}_i\rangle)\right) = -\epsilon \cdot y_i \cdot \text{sign}(\boldsymbol{\theta}), \qquad (15)$$

where $y_i \in \{+1, -1\}$. This is all a consequence of the fact that $\|\cdot\|_1$ is the dual norm of $\|\cdot\|_\infty$.

You should be able to see that if the original constraint were on the traditional Euclidean norm of $\boldsymbol{\delta}$, the role of the one-norm would be played by the traditional 2-norm itself — it is its own dual norm. (Recall from optimization that if $\|\mathbf{x}\|$ is a norm, then its dual norm is $\|\mathbf{x}\|_*$, which is defined as $\|\mathbf{x}\|_* = \max_{\|\mathbf{z}\|\le 1} \mathbf{z}^\top\mathbf{x}$.)

(b) In this part, we consider a simple 2d example. Without explicitly minimizing the worst case loss in (13), we explore the decision boundaries of different models and whether the boundaries could avoid intersecting with the perturbation sets around training data points. **Run Part (I) in Jupyter Notebook and describe the decision boundaries of (i). Neural networks; (ii). Linear models using random Fourier features; (iii). Kernel ridge regression. Also, describe how the boundary changes when you**

  (a) Increase the number of Fourier features in (ii).

  (b) Vary the γ parameter for the rbf kernel from 10^{-3} to 10^{6}.

(c) Now consider that we are trying to learn a binary linear classifier using logistic loss on a subset of the MNIST dataset. We want to learn a classifier distinguishing digit '1' and digit '3'. For this task, we define the perturbation set as

$$\mathcal{S} = \{\delta : \|\delta\|_\infty \leq 0.1\}.$$

Run Part (II) in Jupyter Notebook and finish the code for constructing the adversarial perturbations by using the results you proved in Part (a). Then report the accuracy of the learned linear model and the learned kernel ridge regression on adversarially perturbed test dataset (adversarially perturbations are computed only using the information of the linear model).

5 Your Own Question

Write your own question, and provide a thorough solution.

Writing your own problems is a very important way to really learn the material. The famous "Bloom's Taxonomy" that lists the levels of learning is: Remember, Understand, Apply, Analyze, Evaluate, and Create. Using what you know to create is the top-level. We rarely ask you any HW questions about the lowest level of straight-up remembering, expecting you to be able to do that yourself. (e.g. make yourself flashcards) But we don't want the same to be true about the highest level.

As a practical matter, having some practice at trying to create problems helps you study for exams much better than simply counting on solving existing practice problems. This is because thinking about how to create an interesting problem forces you to really look at the material from the perspective of those who are going to create the exams.

Besides, this is fun. If you want to make a boring problem, go ahead. That is your prerogative. But it is more fun to really engage with the material, discover something interesting, and then come up with a problem that walks others down a journey that lets them share your discovery. You don't have to achieve this every week. But unless you try every week, it probably won't happen ever.

Contributors:

- Anant Sahai
- Chawin Sitawarin
- Kailas Vodrahalli
- Peter Wang
- Philipp Moritz
- Rahul Arya
- Yang Gao
- Yaodong Yu