CSCI 6971/4971: Homework 2

Assigned Monday February 11 2019. Due at beginning of class Thursday February 21 2019.

Remember to typeset your submission, and label it with your name. Coding should be done in Python+NumPy+Sklearn. Attach a printout of your code. Labels all plots appropriately and structure and comment your code neatly and appropriately.

Hand in a physical submission. Do not send me electronic copies of your writeup or code unless I ask for them.

1. Consider the support vector machine optimization problem: minimize the objective function

$$f(\mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^{n} \big(1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)\big)_+ + \frac{\lambda}{2} \|\mathbf{w}\|_2^2,$$

where $\mathbf{x}_i \in \mathbb{R}^d$ are feature vectors and $y_i \in \{-1, 1\}$.

   - Give a simple argument that f is a convex function on $\mathbb{R}^{d+1}$ (consider $(\mathbf{w}, b)$ to be a single vector in $\mathbb{R}^{d+1}$).
   - What is the subdifferential of f, considered as a function of the vector $(\mathbf{w}, b)$? Explain your answer.
   - Implement a stochastic subgradient solver `subgradmethod(X, y, subgradloss, subgradreg, regparam, w1, T, a, m)` that takes as input:
     * `X`, an array of size $n \times d$, each row of which is a feature vector,
     * `y`, an array with n rows, each row of which is the corresponding target,
     * `subgradloss(x, y, w)`, a function that computes a subgradient of the loss on a single training example at the current parameter vector,
     * `subgradreg(w)`, a function that computes a subgradient of the regularizer at the current parameter vector,
     * `regparam`, the regularization parameter λ,
     * `w1`, the initial guess for the parameter vector $(\mathbf{w}, b)$,
     * `T`, the number of iterations,
     * `a`, a parameter governing the step size as $\alpha_t = (1 + at)^{-1}$,
     * `m`, the minibatch size,

     and returns the sequence $\boldsymbol{\omega}_1 = (\mathbf{w}_1, b_1), \ldots, \boldsymbol{\omega}_T = (\mathbf{w}_T, b_T)$ of model parameters. Sample m training pairs without replacement to form each minibatch.
   - Load the `a9a` training and testing data sets from `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html` using `sklearn.datasets.load_svmlight_file`. This is the UCI Adult data set, and the task is to predict whether a person's income is above $50K$/yr, based on census data. Remove the last column from the training data set, so both data sets have 122 features. You may, if you want, preprocess the features in order to increase the accuracy of the model; my baseline against which you will be measured uses only the original unpreprocessed 122 features.
   - Take $\lambda = 1/n_{\text{train}}$ and, by looking at the convergence curves for different values of a on a subset of the training data, choose a parameter a that you think will perform well for the subgradient method on the entire training data set.
   - Run the subgradient method using this value of a to 'convergence', as determined visually by the asymptoting out of $f(\boldsymbol{\omega}_t)$.
   - Similarly choose an a appropriate for sgd with minibatch size of one and run the method to convergence.

- Plot, on the same graph, the accuracy (percentage of correctly predicted labels) of:
  - The sequence of models returned by the subgradient descent method, on the training set.
  - The sequence of models obtained by averaging the last half of the models returned by the subgradient method up to iterate t, on the training set. That is, for each $t \geq 2$, plot the accuracy of the model with parameter $\frac{1}{t-\lfloor t/2 \rfloor+1} \sum_{i=\lfloor t/2 \rfloor}^{t} \theta_i$. This is a more robust alternative to the model averaging we analyzed in class.
  - The sequence of models returned by the subgradient descent method, on the test set.
  - The sequence of models obtained by averaging the last half of the models returned by the subgradient method up to iterate t, on the test set.

  To facilitate comparison between the rates of convergence, let the x-axis be the number of datapoints used (counting multiple passes) to generate each model — e.g. the first model returned from the subgradient method touched n_{train} datapoints, the second touched $2n_{\text{train}}$ and so on. Label your data series and axes! You may find it useful to omit the first few points to more clearly compare the behaviors
- Make a similar plot, but this time plot the performance of the sgd method on the training and test sets.
- Report the highest test accuracy for each of four methods (gradient and subgradient descent with the last model returned, and with the averaged models), along with the number of datapoints used to obtain those accuracies.
- What observations do you have about the performance of sgd and the subgradient method, and the impact of averaging?[1]

2. Consider the problem of completing a low-rank matrix. Let $\Omega \subseteq [m] \times [n]$ be a collection of indices for which we know the corresponding entries in an otherwise unknown matrix $\mathbf{X}_\star \in \mathbb{R}^{m \times n}$, and let $\mathcal{A} : \mathbb{R}^{m \times n} \to \mathbb{R}^{|\Omega|}$ map a matrix to a vector of its entries with indices in Ω. Let $\mathbf{y} = \mathcal{A}(\mathbf{X}_\star)$ denote the known entries of $\mathbf{X}_\star$. We want to recover $\mathbf{X}_\star$ as (a member of)

$$\operatorname{argmin}_{\operatorname{rank}(\mathbf{M})=\operatorname{rank}(\mathbf{X}_\star)} \frac{1}{2}\|\mathcal{A}(\mathbf{M}) - \mathbf{y}\|_2^2, \tag{1}$$

but the problem is that we don't know $\operatorname{rank}(\mathbf{X}_\star)$, *and* this constraint set is not convex.

In analogy to the lasso approach to finding sparse solutions to linear systems, we replace the rank constraint with a regularizer encouraging low-rankedness. Specifically, let $\|\mathbf{M}\|_\star = \sum_{i=1}^{\min\{m,n\}} \sigma_i(\mathbf{M})$ be the nuclear norm of $\mathbf{M}$, the sum of all singular values of $\mathbf{M}$. This is a norm, so is a convex function. Note that it is the ℓ_1 norm of the singular value vector of $\mathbf{M}$, so encourages that vector to be sparse, that is, it encourages $\mathbf{M}$ to be low-ranked. This leads to the convex optimization problem

$$\operatorname{argmin}_{\|\mathbf{M}\|_\star \leq \tau_\star} \frac{1}{2}\|\mathcal{A}(\mathbf{M}) - \mathbf{y}\|_2^2, \tag{2}$$

where we take $\tau_\star = \|\mathbf{X}_\star\|_\star$ and we assume it is known.

To solve (2) we need to know how to project onto nuclear norm balls: if $\mathbf{M} = \mathbf{U}\operatorname{diag}(\boldsymbol{\sigma})\mathbf{V}^T$ is the SVD of $\mathbf{M}$ and C is the nuclear norm ball of radius τ, then $P_C(\mathbf{M}) = \tau\mathbf{U}\operatorname{diag}(P_B(\boldsymbol{\sigma}))\mathbf{V}^T$, where B is the ℓ_1 ball of radius 1.

- Why is the optimization problem in (1) not convex? What does this imply about the prospect of using projected gradient descent to find a minimizer?

[1]You may want to play around with ℓ_1 regularization and minibatch sizes to see how they affect performance as well.

- Download $\mathbf{y} = \mathcal{A}(\mathbf{X}_\star)$, m, n, the indices Ω, the target rank, and $\tau_\star$ from `http://www.cs.rpi.edu/~gittea/teaching/spring2019/files/hw2recoverydata.npz`.
- Write a function `mat2im` that converts a numpy float array into a grayscale Pillow `Image` object by first scaling the minimum entry of the array to 0 and the largest entry to 255, converting the datatype to uint8, then using the `Image.fromarray` function.
- Write a function that implements $\mathcal{A}$, given a matrix and Ω, and another function that, given a vector and Ω, implements the adjoint $\mathcal{A}^\star$ that maps the measurement vector onto a matrix consistent with those measurements that has zeros everywhere else.
- Use `mat2im` to visualize $\mathcal{A}^\star(\mathbf{y})$.
- Implement a projected gradient descent solver for (1)[2]. Take the rank as given in the datafile, $\mathbf{X}_1 = \mathbf{0}$, and find a constant stepsize α and appropriate iteration count T so that the recovered image is as visually satisfactory as you can manage. Use `mat2im` to visualize $\mathbf{X}_T$.
- Report α, T, and $\|\mathcal{A}(\mathbf{X}_T) - \mathbf{y}\|_2^2 / \|\mathbf{y}\|_2^2$.
- Are there any details in the recovered image that you could not visually extrapolate from $\mathcal{A}^\star(\mathcal{A}(\mathbf{y}))$?
- Argue that the objective function in (2) is indeed convex, and find its gradient. You will find the fact that $\langle \mathcal{A}(\mathbf{M}), \mathcal{A}(\mathbf{M}) \rangle = \langle \mathcal{A}^\star(\mathcal{A}(\mathbf{M})), \mathbf{M} \rangle$ to be useful.
- Download `simplex_projection.py` from `https://gist.github.com/daien/1272551`. It contains an implementation of an algorithm for computing projections onto ℓ_1 balls.
- Implement a projected gradient descent solver for (2). Take $\mathbf{X}_1 = \mathbf{0}$ and find a constant stepsize α and appropriate iterate T so that the recovered image is as visually satisfactory as you can manage. Use `mat2im` to visualize $\mathbf{X}_T$.
- Report α, T, and $\|\mathcal{A}(\mathbf{X}_T) - \mathbf{y}\|_2^2 / \|\mathbf{y}\|_2^2$.
- How does the output of the convex formulation compare visually to the output of the nonconvex formulation? Try to explain the cause for any difference in quality.

3. [For CSCI6971 students.] Consider the following iterative algorithm for minimizing a function f on a convex set C: given a current point $\mathbf{x}_t$, linearize the function f around $\mathbf{x_t}$, then minimize this linear approximation over C, using a quadratic regularization that forces us not to stray too far from $\mathbf{x}_t$. That is, given $\mathbf{g} \in \partial f(\mathbf{x}_t)$

$$\mathbf{x}_{t+1} = \text{argmin}_{\mathbf{u} \in C}\, f(\mathbf{x}_t) + \langle \mathbf{g}, \mathbf{u} - \mathbf{x}_t \rangle + \frac{1}{2\alpha_t} \|\mathbf{u} - \mathbf{x}_t\|_2^2.$$

Argue that this algorithm is exactly the projected subgradient method with stepsize given by α_t.

[2]The projection onto the set of rank r matrices is given by the truncated SVD. sklearn has a truncated SVD implementation that is efficient; if you choose to use it, use the Arpack version.