Introduction to Machine Learning (67577)

Exercise 6
PCA, convex optimization and SGD

June 2020

Submission guidelines

- The assignment is to be submitted via Moodle only.
- Files should be zipped to **ex_6_First_Last.zip** (First is your first name, Last is your last name. In English, of course).
- The .pdf file size should not exceed 10Mb.
- Each question is enumerated, use these numbers when answering your questions.
- You must submit your code in python files

PCA

[Relevant material - Tirgul 12 and Lec 10]

Theoretical part

1. Let X be a random variable with some distribution over $\mathbb{R}^d$ so that its mean is $(0,\ldots,0) \in \mathbb{R}^d$ and its covariance matrix is $\Sigma \in \mathbb{R}^{d\times d}$. Show that for any $v \in \mathbb{R}^d$, where $\|v\|_2 = 1$, the variance of $\langle v, X\rangle$ is not larger then variance of the PCA embedding of X into 1-dimension (Assume that the PCA uses the actual Σ).

Coding part

2. Now we will examine a scenario in which we had n datapoints in a k-dimensional linear subspace of $\mathbb{R}^d$, where $k < d$ (e.g. in the original dataset only the first 10 coordinates can have a non zero value). As in real life, the datapoints got corrupted. We will examine a scenario in which the data was corrupted by additive Gaussian noise.

   Before handling the noisy data, one would like to get rid of directions of the data that are a result of pure noise (e.g. coordinates $11, 12, \ldots, d$). Let's see how we can do that with PCA:

* **Important:** We would like you to implement the original version of the PCA as shown in subsection (2.2) of recitation 12 - where one needs to take the mean of the noisy data before doing any computation.

- Generate a data matrix $X \in \mathbb{R}^{n \times d}$ where $n = 100$, $d = 100$, with rank 10 so that its singular values will be $\{\sqrt{20}, \sqrt{18}, \ldots, \sqrt{2}\}$.
  * You may use an SVD on some random matrix.
  * In order to make things easier for you- construct X so that the mean of each column will be 0 (meaning that the empirical mean of the data will be 0). You may use this code:

  ```
  import numpy as np
  A= np.random.randn(n,d)
  A_mean_mat= np.tile(np.mean(A,0).reshape((1,-1)),(n,1))
  A -= A_mean_mat
  U,_,V = np.linalg.svd(A,full_matrices= True)
  D= np.sqrt(2*np.arange(10,0,-1))
  X= np.dot(np.dot(U[:,:10], np.diag(D)), V[:10,:])
  ```

- Sample a random matrix $Z \in \mathbb{R}^{n \times d}$, where all of the matrix entries are sampled iid from the distribution $\mathcal{N}(0, \sigma^2)$.
- Generate a noisy data matrix $Y = X + Z$
- Compute the eigenvalues of noisy data's (Y) covariance matrix (Treat each row as a sample). Denote its eigenvalues by $\lambda_1, \ldots, \lambda_{100}$, where $\lambda_1 \geqslant \lambda_2 \geqslant \ldots \geqslant \lambda_{100}$.
- Plot (and add to your submission pdf) a bar graph of the eigenvalues in descending order - the x values will be the indices of the eigenvalues, and the y values will be the eigenvalues themselves, e.g. $(1, \lambda_1)$, $(2, \lambda_2), \ldots$ (you may use matplotlib.pyplot.bar)

(a) Repeat this question for $\sigma \in \{0, 0.05, 0.1, 0.15, 0.2, 0.4, 0.6\}$:

(b) Write in your own words the phenomenon that you have observed in (a).

(c) We will now explore the problem of choosing the embedding dimension

   i. Given the eigenvalues of the covariance matrix- try to think of a method to distinguish between an eigenvalue that is a product of pure noise and an eigenvalue that is a product of the original data.
   The only assumption that you have on your data is that originally it had a low unknown rank structure, and then it was corrupted with some noise with some unknown level.

   Describe your method, even if it is a little vague, and apply it to three scenarios from question (a).

   ii. Calculate the inner product between the 10 leading left singular vectors of X with the 10 leading eigenvectors (the eigenvectors with the highest eigenvalues) of the covariance matrix of Y. Explain what you saw and how would you change your method based on that.

Convex optimization and SGD

Theoretical part

[Relevant material - Tirgul 13]

3. (a) **Prove:** Let $f_i : V \to \mathbb{R}$ for $i = 1, \dots, m$ be functions and let $\gamma_1, \dots, \gamma_m$ be positive scalars. Consider the function $g : V \to \mathbb{R}$ given by

$$g(u) = \sum_{i=1}^{m} \gamma_i f_i(u).$$

If $f_1, \dots, f_m$ are convex, then g is also convex.

(b) Give a counterexample for the following claim: Given two functions $f, g : \mathbb{R} \to \mathbb{R}$, define a new function $h : \mathbb{R} \to \mathbb{R}$ by $h = f \circ g$. If f and g are convex than h is convex as well.

(c) **Prove:** A function $f : C \to \mathbb{R}$ defined over a convex set C is convex iff its *epigraph* is a convex set, where $\mathrm{epi(f)} = \{(u, t) : f(u) \leqslant t\}$.

(d) Let $f_i : V \to \mathbb{R}$, $i \in I$. Let $f : V \to \mathbb{R}$ given by $f(u) = \sup_{i \in I} f_i(u)$. If f_i are convex for every $i \in I$, then f is also convex. (Hint: what can you say about *epi*(f)?)

4. (a) Given $x \in \mathbb{R}^d$ and $y \in \{\pm 1\}$. Show that the hinge loss is convex in (w, b). Meaning, define

$$f(w, b) = l_{x,y}^{hinge}(w, b) = \max\left(0, 1 - y \cdot (\langle w, x \rangle + b)\right)$$

show that f is convex in (w, b).

(b) Find a g such that $g \in \partial l_{x,y}^{hinge}(w, b)$.
(Hint: you can find useful theorem about $\partial \max_{i \in [m]} f_i(x)$ in the Tirgul)

(c) Given $f_1, \dots, f_m : \mathbb{R}^d \to \mathbb{R}$ convex functions, and $\xi_k \in \partial f_k(x)$ for all k. Define $f : \mathbb{R}^d \to \mathbb{R}$ by $f(x) = \sum_{i=1}^{m} f_i(x)$. Show that $\sum_k \xi_k \in \partial \sum_k f_k(x)$.

(d) Consider the dataset $\{(x_i, y_i)\}_{i=1}^{m} \subset \mathbb{R}^d \times \{\pm 1\}$ and $\lambda \geqslant 0$. Define the function $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}$ by:

$$f(w, b) = \frac{1}{m} \sum_{i=1}^{m} l_{x_i, y_i}^{hinge}(w, b) + \frac{\lambda}{2} \|w\|^2$$

Show that f is convex and find a member of the sub-gradient of f.

Practical part - SVM on MNIST

[Relevant material - Tirgul 13 and 4]

In this part we will solve a classification problem on a handwritten digits dataset (the "MNIST" dataset- The dataset contains a 28 by 28 images of handwritten digits). Since the computer represents images by arrays of pixels (integers in the range [0,255]), we will describe each image as a vector. The dataset contains all 10 digits, but we will focus on the case of binary classification between only a pair of digits. We will do so by finding the separating hyper-plane with maximum margin, namely - SVM.

- Download the file load_data.py from the course website.
- Your results should contain your code, and all the figures you are asked to plot (in the PDF of the answers to the theoretical part). Additionally, please include a script ex6_runme.py that runs your code, generates the figures, and prints the parameters and errors you are asked to report.

Assumptions:

- When we write GD, we actually mean sub gradient descent.
- The GD (gradient descent) and SGD (stochastic gradient descent) algorithms will use the loss function defined in the theoretical part. While in SGD we define m as the batch size, in GD it will defined as the training set size (Meaning, the batch is the whole training set).
- At each iteration the SGD algorithm draws a batch of samples from the training set without repetitions (repetitions could occur between different batchs). In order to implement it examine the function- np.random.choice.
- We would like to test our hypothesis using the 0-1 loss, but in the training process we are going to use the hinge loss (or the likelihood) as a surrogate loss (**Train on hinge loss, test on 0-1 loss**).

5. Create a file: "SVM.py" that will contain the functions: gd, sgd, and test_error.

   (a) The declaration of the GD function should be: "*gd(data, label, iters, eta, w)*", where
   - *data*- a $n \times d$ numpy array, where n is the amount of samples, and d is the dimension of each sample
   - *labels*- a $n \times 1$ numpy array with the labels of each sample
   - *iters* - an integer that will define the amount of iterations.
   - *eta* - a positive number that will define the learning rate.
   - w- a $(d+1) \times 1$ numpy array, where d is the dimension of each sample- the parameters of the classifier (the last element is the bias).

   The function will output a $(d+1) \times 1$ numpy array, contain the output of the GD descent algorithm over 'iters' iterations. Use the gradient that we found in the SVM section of the theoretical part.

   (b) The declaration of the sgd function should be: "*sgd(data, label, iters, eta, w, batch)*", where in addition to the definitions from the previous bullet, we add:
   - *batch*- The amount of samples that the algorithm would draw and use at each iteration.

   The output of the function is the same as in gd.

   (c) The declaration of the *test_error* function should be: "*test_error(w, test_data, test_labels)*", where
   - w- a $(d+1) \times 1$ numpy array, where $h_w(x) = sign(\langle w[: -1], x\rangle + w[-1])$.
   - *test_data* - a $n \times d$ numpy array with n samples
   - *test_labels*- a $n \times 1$ numpy array with the labels of the samples.

   The function will output a scalar - with the respective 0-1 loss for the hypothesis.

6. After writing all this code, lets play with it and try to analyze the results.

   (a) We start by classifying the digits 0-1. Load the training data and test data using the function *get_digits()* in *load_data.py*. In order to get the 0/1 digits use: *x_train, y_train, x_test, y_test = get_digits(0, 1)*. This function uses the 'Tensorflow' package. In order to use this package in the school read the instructions in cs-wiki, and watch this.

   Note: (As in Ex. 3) If you have any problem with tensorflow, please do not fight it. Instead, there are many ways to download MNIST data, like this link, or you can download it locally into a csv file, and just read it from there, like this link.
   Run the SGD algorithm with batches of 5, 50, 100 and the GD algorithm over 150 iterations. **Make sure to change the labels to 1/-1: e.g.** $1 \to 1, \quad 0 \to -1$. Which learning rate η did you use?

   (b) **Draw** a graph of the training loss of each algorithm at each iteration and another graph of the test loss. Use the 0-1 loss for both algorithms.

   (c) **Question:** SGD uses mini-batches and tries to approximate GD but with less samples. Analyze the results in the previous item, and explain them. Explain how varying the batch size affects the computational complexity and the accuracy of the subgradient in the learning process.

   (d) Run through a,b again, but with $\eta = 0.1, 10$. What can we learn from the learning rate results in this specific problem?
   For what cases should we consider high learning/ low learning rate? You may consider the gradient of $f(x) = 10^8 \cdot |x|$ and $g(x) = 10^{-8} \cdot |x|$ at different points and its effect in GD when you answer the question.

   (e) Try classifying another pair of digits. Find a pair of MNIST digits that the GD or the SGD doesn't succeed very well on them and suggest an explanation for the bad results.