6.867: Homework 2

This homework will be structured in the same way as homework 1. Details of the submission and grading process are repeated at the end. ***Unlike homework 1, for this homework, you must submit your own individual write-up with your own work, figures, and solutions to receive credit.***

- We *strongly* encourage you to work on the problems in groups of two students, though you may work individually if you wish.
- However, all write-ups must be done individually, comprised of your own work, figures, and solutions.
- If partners submit identical PDFs, both will receive no credit.
- Submit your paper via the CMT site by 11:59 PM on Tuesday, October 24.
- After submission on CMT, you will be assigned 3 papers to review.
- Reviews must be entered on the CMT site by 11:59PM on Tuesday, October 31.
- All reviews will be made visible shortly thereafter, and students will have a two day period in which to enter rebuttals of their reviews into CMT if they wish.

Introduction

In this assignment, we will explore several related approaches to learning classifiers. There are lots of different choices one can make, which result in different detailed algorithms. We won't try them all! But it's useful to at least lay out the space of choices that are relevant to the algorithms we ask you to experiment with.

- **Overall setting**: Given an example x with a vectored feature representation and a class label y, we can learn a *conditional probability model* of the form $P(y \mid x)$ or leap directly to a prediction rule of the form $\hat{y} = f(x)$, i.e. our estimate of the true mapping of example to class label, $x \rightarrow y$.
- **Loss function**: One way to obtain a prediction rule is to minimize empirical risk with a regularization penalty. Remember that for finite training examples, the empirical risk is the average loss over all training examples. However, we evaluate the sum because scalars do not affect the minimization problem. In this homework, we will consider two types of loss functions, the logistic loss and hinge loss. When $y \in \{+1, -1\}$, then we can write $\mathcal{L}(w, w_0)$, the log-likelihood of the data for the learned parameters of our linear model, as the negative of the empirical risk with the logistic loss (the positive term on the inside of the summation) :

$$\mathcal{L}(w, w_0) = -\sum_i \log\left(1 + \exp\left(-y_i(w^T x_i + w_0)\right)\right) \quad .$$

Alternatively, the hinge loss has the form

$$\text{HingeLoss}(w, w_0) = \sum_i \max\left(0, 1 - y_i(w^T x_i + w_0)\right) \quad .$$

- **Kernelization**: Is the objective "kernelized"? That is, can we write it strictly in terms of dot products between feature vectors? If so, we can easily try interesting feature spaces as long as they have an appropriate kernel function.
- **Regularizer**: Are we using L1 or L2 regularization? Or some other form of regularization? Is it applied to the weight vector or to the α vector, in the case of kernelized algorithms?
- **Optimizer**: Are we optimizing our metric using a relatively simple gradient descent method, or a more sophisticated solver such as a quadratic programming (QP) solver? (When thinking about this question, it's important to consider whether your objective is convex or not.)

We will be using two types of data sets:

- Synthetic (artificially generated) 2 dimensional (2D) data sets, numbered 1 through 4, designed to illustrate various points about classification. Each of the four synthetic datasets is divided into train, validation, and test sets.
- MNIST: images of handwritten digits. We will experiment with distinguishing between sets of digits.

The sections in this assignment explore different parts of the algorithm space.

1. Section 1 considers logistic regression with different types of regularization; although you could solve these problems with your GD code, or variations on it, you are welcome to use existing implementations on this section.
2. Section 2 considers standard support-vector machine classification, with hinge loss, but expressed in the dual as a quadratic program. It has L2 regularization on the weights, but ends up with sparsity in the α terms because of the form of hinge loss. We also explore non-linear kernels in this context.
3. Section 3 keeps the objective function from SVMs but seeks efficiency in large data sets by using a variation on stochastic gradient descent.
4. Section 4 applies some of these methods to the MNIST digit classification problem.

1 Logistic Regression (LR)

Implementation*: You are welcome to use off-the shelf packages for parts 2 and 3 of this problem, but part 1 should be written from scratch (numerical and vectorized computation libraries like numpy in Python are fine to use).*

Python*: Use Sklearn's* `Logistic Regression`*, which can do both L_1 and L_2 regularization. Note carefully how the regularization works in the implementation you use: in particular, Sklearn uses $C = 1/\lambda$. Make sure that you understand how w_0 (bias/intercept) is treated, as well: see* `intercept_scaling` *in Sklearn. When you report weights, you should include w_0.*

Matlab*: Use L1 regularized Logistic Regression (*`https://stanford.edu/~boyd/l1_logreg/`*) for Matlab. We have provided you a script* `l1LogReg.m` *to help you with that. For L2 regularized Logistic Regression use the* `fminunc` *function.*

We provide skeleton code that you might find useful in `hw2_resources/lr_test.py/m`*. Note that this code is set to assume two input features when reading and plotting the data.*

1. Use a gradient descent method to optimize the logistic regression (LR) objective, with L_2 regularization on the weight vector. For L_2 regularization, the objective function (error of LR) should be of the form

$$E_{LR}(w, w_0) = -\mathcal{L}(w, w_0) + \lambda \|w\|_2^2$$

   where $\|w\|_2 = \sqrt{w_1^2 + \ldots + w_n^2}$.

   Using your code to perform gradient descent, train your model parameters (w, w_0) using the training data from `data1_train.csv` with $\lambda = 0$. What happens to the weight vector as a function of the number of iteration of gradient descent? What happens when $\lambda = 1$? Explain.

2. Let's now compare the effects of L_1 and L_2 regularization on LR. Note that for L_1 regularization, the objective function should be of the form

$$E_{LR}(w, w_0) = -\mathcal{L}(w, w_0) + \lambda \|w\|_1$$

   where $\|w\|_1 = \sum_{i=1}^{n} |w_i|$. See the implementation notes.

   Evaluate the effect of the choice of regularizer (L_1 vs L_2) and the value of λ on (a) the weights, (b) the decision boundary and (c) the test-set classification error rate for each of the training data sets.

3. Use the training and validation sets to pick the best regularizer and value of λ for each data set: `data1`, `data2`, `data3`, `data4`. Report the performance on the test sets.

2 Support Vector Machine (SVM)

1. Implement the dual form of linear SVMs with slack variables. Do not use the built-in SVM implementation in Matlab or sklearn. Instead, write a program that takes data as input, converts it to the appropriate objective function and constraints, and then calls a quadratic programming package to solve it. See the file `optimizers.txt` for installation and usage for matlab/python.

   Show in your report the constraints and objective that you generate for the 2D problem with positive examples (2, 2), (2, 3) and negative examples (0, -1), (-3, -2). Which examples are support vectors?

   Note: When using quadratic program solvers, be careful when computing support vectors based on the solution. Optimizers do not always drive variables to 0 despite them being 0 in the optimal solution. Thus you may need to perform thresholding on α values to extract the correct support vectors. We suggest using a threshold of around 10^{-4}, but please check your α values to ensure you are correctly counting the number of support vectors!

2. Test your implementation on the 2D datasets. Set C=1 and report/explain your decision boundary and classification error rate on the training and validation sets. We provide some skeleton code in `svm_test.py/m` to get you started.

3. The dual form SVM is useful for several reasons, including an ability to handle kernel functions that are hard to express as feature functions in the primal form. Extend your dual form SVM code to operate with kernels. Do the implementation as generally as possible, so that it either takes the kernel function or kernel matrix as a hyper-parameter input to your model fitting function and uses the kernel directly in the optimization. For this implementation, explore the effects of choosing values of $C \in \{0.01, 0.1, 1, 10, 100\}$ on (a) linear kernels and (b) Gaussian RBF kernels as the bandwidth is also varied. Report your results and answer the following questions:
   (a) What happens to the geometric margin $1/\|\mathbf{w}\|$ as C increases? Will this always happen as we increase C?
   (b) What happens to the number of support vectors as C increases?
   (c) The value of C will typically change the resulting classifier and therefore may also affect the accuracy on test examples. Why would maximizing the geometric margin $1/\|\mathbf{w}\|$ on the training set not be an appropriate criterion for selecting C? Is there an alternative criterion that we could use for this purpose?

3 Soft-Margin Support Vector Machine with Pegasos

SVMs are convenient, especially because of the ease of kernelizing them and the sparse representation in terms of data points when we represent our decision boundary solely in terms of the support vectors. But it can be very inefficient to solve large SVMs. Here, we consider solving the following formulation of soft-margin SVM using the Pegasos algorithm.

$$\min_w \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{i=1}^{N} \max\{0, 1 - y_i(w^T x_i)\}$$

The soft-margin SVM is sometimes referred to as C-SVM where $C = \frac{1}{N\lambda}$. The Pegasos algorithm is a combination of good ideas: hinge loss, L2 regularization, stochastic (sub)-gradient descent (because the gradient does not exist everywhere, the descent has to be handled a bit specially), and a decaying step size (initialized at $\frac{1}{\lambda}$ and decaying with $\frac{1}{t}$). Compared to other SVM algorithms, Pegasos[1] is appealing due to its performance in theory and in practice, and its ease of implementation.

In this problem, we will implement the Pegasos learning algorithm, both as a linear classifier and using a non-linear kernel. We will use the 2D data sets for this problem. We provide skeleton code in `pegasos_linear_test.py/m` and `pegasos_gaussian_test.py/m`.

1. The following pseudo-code is a slight variation on the Pegasos learning algorithm, with a fixed iteration count and non-random presentation of the training data. Implement it, and then add a bias term (w_0) to the hypothesis, but take care not to penalize the magnitude of w_0. Your function should output classifier weights for a linear decision boundary.

[1] If you'd like to learn more about Pegasos, lecture slides and the original publication are available on Stellar in the "Materials > Supplementary Material" folder.

Input: data (x_i, y_i), regularization constant λ, max_epochs
Initialize: $t \leftarrow 0$, $w_{t=0} \leftarrow 0$
while $epoch < max\_epochs$ **do**
  for $i = 1, ..., N$ **do**
    $t \leftarrow t + 1$
    $\eta_t \leftarrow 1/(t\lambda)$
    if $y_i(w_t^T x_i) < 1$ **then**
      $w_{t+1} \leftarrow (1 - \eta_t\lambda)w_t + \eta_t y_i x_i$
    else
      $w_{t+1} \leftarrow (1 - \eta_t\lambda)w_t$
    end if
  end for
end while

2. Test various values of the regularization constant, $\lambda = 2^1, \ldots, 2^{-10}$. Observe the the margin (distance between the decision boundary and margin boundary) as a function of λ. Does this match your understanding of the objective function?

3. We can also solve the following kernelized Soft-SVM problem with a few extensions to the above algorithm. Rather than maintaining the w vector in the dimensionality of the data, we maintain α coefficients for each training instance. This vector may be sparse, depending on our hyperparameter choices, with non-zero entries for support vectors, as we saw in the previous problem.

$$\min_w \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{i=1}^{N} \max\{0, 1 - y_i(w^T\phi(x_i))\}$$

Implement a kernelized version of the Pegasos algorithm. It should take in a Gram matrix, where entry i, j is $K(x^{(i)}, x^{(j)}) = \phi(x^{(i)}) \cdot \phi(x^{(j)})$, and should should output the support vector values, α, or a function that makes a prediction for a new input. In this version, you do not need to add a bias term.

NOTE: The α variables here cannot be interpreted in the same way as the dual variables we have seen in Bishop and in class. Here α_i can be negative and may not satisfy the other constraints. However, like in the dual optimization problem, the non-zero α terms are the only examples contributing to the solution and can be thought of as support vectors. Also, note that the discriminant in the if statement does not include y_j in the sum over the training examples - this is a consequence of the unfortunately confusing notation. The discriminant for an example x_i is given by $\sum_j \alpha_j K(x_j, x_i)$

Input: data (x_i, y_i), regularization constant λ, kernel K, max_epochs
Initialize: $t \leftarrow 0$, $\alpha_{t=0} \leftarrow 0$
while $epoch < max\_epochs$ **do**
  for $i = 1, ..., N$ **do**
    $t \leftarrow t + 1$
    $\eta_t \leftarrow 1/(t\lambda)$
    if $y_i(\sum_j \alpha_j K(x_j, x_i)) < 1$ **then**

$\alpha_i \leftarrow (1 - \eta_t\lambda)\alpha_i + \eta_t y_i$
else
$\alpha_i \leftarrow (1 - \eta_t\lambda)\alpha_i$
end if
end for
end while

Given this formulation, how should you make a prediction for a new input x? Does it have the same sparsity properties as the dual SVM solution?

4. Classify the same data using a Gaussian kernel, $K(x, x') = \gamma\|x - x'\|^2$, and test various values of the $\gamma = 2^2, \ldots 2^{-2}$, to vary the bandwidth of the kernel. Use a fixed $\lambda = .02$.

   How does the decision boundary and the number of support vectors change depending on γ? How do your results compare to those obtained with the SVM in the previous section?

4 Handwritten Digit Recognition with MNIST

In this question, we will apply our classifiers to automatically classify images of hand-written digits from the famous MNIST dataset. Though MNIST is often used for multi-class classification, we will look at binary classification of different subsets of digits.

The data is provided as separate files each containing images of the individual digits (0-9). You can find these files in `data/mnist_digit_*.csv`. Each file contains contains approximately 5000 images of one of the digits. Each 28 x 28 image in the dataset is represented as a vector of length 784, specifying the gray-scale value of each pixel. In the provided data, the first 28 columns represent the first row of the image, and so on.

For the questions below, you will first need to construct training, validation and test sets. You will do that by collecting images for some subset of the digits, that will be labeled as +1, and some other subset that will be labeled as -1. Take the first 200 points for each digit and use them for training, use the next 150 points for validation and the final 150 points as a test set. If you are classifying (2,4) vs (3,5), then your training set will have 800 samples, your validation and training sets will have 600 samples each. Note that you will construct a vector of ones of length 400 and a vector of -1s, also of length 400, and these will be the labels for your training set, and so on. `Be careful with the sizes of these data sets, you probably don't want to use all the samples in the data files.`

After constructing these data sets, you should also construct corresponding sets that are "normalized", so that each input feature (pixel) is in the range [-1, 1]. Because each feature in this dataset has minimum value 0 and maximum value 255, you can do this normalization by transforming the whole X matrix (element-wise) to $2X/255 - 1$.

Try (at least) the following pairwise classification tasks: 1 vs 7, 3 vs 5, 4 vs 9, (0, 2, 4, 6, 8) vs (1, 3, 5, 7, 9).

1. Show the classification accuracy of your logistic regression from problem 1 and the linear SVM classifier from problem 2 on the MNIST data. Compare and contrast the performance of each classifier (both training and testing). Does normalization of the data matter? Look at some of the images of mis-classified digits, do they make sense?

2. Explore the classification accuracy of a Gaussian RBF SVM classifier on this data. Pick values of C and γ (bandwidth) using the validation data and report performance on the test sets. Does normalization of the data matter? Can you improve on the linear classifiers?

3. Now, let's compare Pegasos (in Problem 3) to the QP implementation (in Problem 2). Are the accuracy values comparable? Consider running time of both approaches as the number of training points increase (200, 300, 400, 500). This comparison will depend on the Pegasos parameters (λ and number of epochs). Summarize and explain your findings for a few settings of λ and number of epochs.

Procedure, grading, etc.

You will find a zip file with some useful code and data in the Stellar Materials page. You can do these assignments in any computational system that you are used to. We recommend Matlab or the Pylab/Numpy/Scipy/Matplotlib cluster of packages and we'll try to provide help for those two systems. If you use anything else, you're on your own...

You will be turning in a single, readable "paper" (a single PDF file) with your solutions. In the Stellar Materials page we have provided some sample solutions from a previous year. The content in these examples is not directly relevant, but note that there are coherent explanations and nice illustrations of results in figures and tables. You should write your paper as if it is going to be read by someone who has taken a class in machine learning but has not read this assignment handout.

We will be emulating the process of submitting papers for publication to a conference. We will be using an actual conference review system (CMT) to have these papers peer reviewed (by the other students). This means that your answers have to be readable and understandable to your peers and, where possible, interesting. Note that when explanations are called for, you will need to convince the reviewers that you understand what you're talking about. The course staff will serve as the Program Committee for the conference and make all final decisions.

Grading rubric

Your paper must be anonymous (no identifying information should appear in the PDF file). If it is not, it will automatically receive a 20% deduction, and will be graded by a grumpy staff member.

The paper must be no more than 6 pages long in a font no smaller than 10 point. It should include whatever tables, graphs, plots, etc., are necessary to demonstrate your work and conclusions. *It should not include code.*

Each of the **four** parts of the assignment will be graded on a scale from 0 to 5 (where 0 is failing and 5 is an A) on two aspects:

- **Content:** Did the solution answer the questions posed? Were the answers correct? Were the experiments well-designed or examples well chosen?
- **Clarity:** Were the results written up clearly? Were the plots labeled appropriately and described well? Did the plots support the points in the paper? Did the discussion in the paper illuminate the plots?

As a reviewer, you will be asked to provide a score for each section, and at at least two paragraphs of feedback, per review, explaining things that were done well and things that could have been improved upon.

Your overall score for this assignment will be:

- **80%**: The average of all 8 scores on your assignment given by all three reviewers.
- **20%**: A score for the quality of your reviews. This will be full credit, by default. But we will skim reviews and examine some carefully and may reduce this grade for review commentary that is sloppy or wrong.

The course staff will spot-check submissions and reviews, paying careful attention to cases where there were rebuttals. The staff will act as the program committee and determine a final score. Our overall goals in this process are:

- To motivate you to work seriously on the problems and learn something about the machine learning material in the process
- To engage you in thinking critically and learning from other students' solutions to the problems

We will arrange to give full credit to anyone who submits a serious and careful solution to the problems and who gives evidence of having read carefully the solutions they were assigned and who writes thoughtful reviews of them.

The questions are the points that your paper should cover in order to receive full credit. Your presentation should roughly follow the order of these questions so that your reviewers can see what you're doing.