6.867: Homework 1

For the first homework, you will submit a single PDF containing your solutions to the following problems. There are 2 parts to the homework: a theoretical part and a hands-on part. We impose a strict 6-page limit on your writeup, to ensure that you focus on the important parts of your implementation and analysis. We make a few notes about the collaboration policy, submission procedure below, grading policy and other procedural information below.

Collaboration Policy

- We *strongly* encourage you to work on the problems in groups. Group size can be arbitary but we recommend keeping it to no larger than 5. However, all write-ups must be done individually, comprised of your own work, figures, and solutions.
- If partners submit identical PDFs, all will receive no credit.
- You should never use results from other students, the staff (from this year or from previous years), or other *online resources* in preparing your solutions to homework problems. In addition, students should never share their solutions (or staff solutions) with other students, including through public code repositories such as Github.

Submission: CMT & Dates

CMT is an online conference management software that you will use to submit and peer-grade your submissions. Instructions on how to use CMT should have been posted to Piazza. A few notes on deadlines.

- Submit your paper via the CMT site by 11:59 PM on September 27th.
- After submission on CMT, you will be assigned 3 papers to review.
- Reviews must be entered on the CMT site by 11:59PM on October 2nd.
- 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.

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). We will send out details about CMT soon. 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.

Introduction to Theoretical Section

The first part of this assignment is 2 theoretical problems that will allow you to apply your more theoretical, proof-oriented problem solving skills. Note that this is a bit different from homeworks of past years, which were focused entirely on more hands-on implementation. For the theoretical problems, the writeup has no page-limit, and can be included with the write-up for the hands-on section. The theoretical problems are stated as follows:

1 Classification Error of k-Nearest Neighbors Rule with $k = 1$.

This question is about k-nearest neighbors classification with $k = 1$ (abbreviated as 1-NN in the rest of the question). Our goal is to derive a generalization error bound for 1-NN from *first principles*.

The 1-NN algorithm classifies a new example x by finding a training example that is nearest to x. For simplicity, we consider a scalar feature space. Suppose that there are n historical observations $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. For a new example x, let

$$n'(x) \in \operatorname*{argmin}_{1 \le j \le n} |x_j - x|.$$

Then the predicted label is $Y_{\text{1-NN}}(x) = y_{n'(x)}$.

We shall consider the setting described as follows. Let X denote the feature random variable and Y the label. We assume that Y is a binary random variable, *i.e.* $Y \in \{0, 1\}$. The conditional distribution of X given $Y = \theta$ has a density function $f_\theta(\cdot)$ for $\theta = 0, 1$. We shall also assume that

$$\min_{x \in [a,b]} (f_0(x), f_1(x)) \ge \gamma,$$

for some $a < b$ and $\gamma > 0$. Define $\eta_0(x) = \mathbb{P}(Y = 0|X = x)$ and $\eta_1(x) = \mathbb{P}(Y = 1|X = x)$. Let us consider any $x \in (a, b)$ for the following questions.

(a) Recall that the Bayes classifier minimizes the probability of classification error and therefore provides a benchmark for evaluating performance of other algorithms. Let $Y^*(x)$ denote the Bayes classifier and $\delta^*(x)$ the Bayes error (*i.e.* the probability of misclassification). Give the formulae for $Y^*(x)$ and $\delta^*(x)$.

(b) What is the VC-dimension of the 1-NN algorithm? Explain. What does it imply about generalization error?

(c) Under the assumption that training examples $X_1, X_2, \ldots, X_n$ are independent and identically distributed as X, argue that, for any $\beta > 0$,

$$\lim_{n \to \infty} \mathbb{P}\left(\min_{1 \le j \le n} |X_j - x| \ge \beta \right) = 0.$$

(d) For an arbitrary positive integer n, derive the formula for $\mathbb{P}(Y \neq Y_{\text{1-NN}}(x)|X = x)$. Derive the limits of $\eta_0(X_{n'(x)})$ and $\eta_1(X_{n'(x)})$ as $n \to \infty$, under the assumption that $f_0(\cdot)$ and $f_1(\cdot)$ are continuous over the interval (a, b).

(e) Prove that, in the limit of infinite training samples, the 1-NN classification error is at most twice of the Bayes error:

$$\lim_{n\to\infty} \mathbb{P}(Y \neq Y_{\text{1-NN}}(x)|X = x) \leq 2\delta^*(x).$$

2 Logistic Regression

In this question, we will guide you to understand the learning capability as well as different modeling choices of logistic regression. Let us consider two-class classification problems, where each sample $x_i \in \mathbb{R}$ is either labeled as 1 or 0. Recall that logistic regression attempts to find a *linear classifier* (i.e., a hyperplane), and it does so by modeling the conditional probability as:

$$\mathbb{P}(y = 1|x; w, w_0) = \sigma(w^T x + w_0), \tag{1}$$

where $\sigma(\cdot)$ represents the sigmoid function and the notation $< \cdot, \cdot >$ means the inner product between two vectors.

For simplicity, throughout this question, we assume that there is no intercept/offset term w_0 in Eqn. (1). That is, we consider the case where

$$\mathbb{P}(y = 1|x; w) = \sigma(w^T x). \tag{2}$$

1. **VC-Dimension:** Let us first understand the learning capability of logistic regression. In particular, we will work out the VC-dimension of logistic regression.

   Suppose that $x \in \mathbb{R}^d$. We then claim that without the intercept term w_0, the VC-dimension of logistic regression is precisely d. Verify the claim by rigorously prove the following two statements:

   (a) The VC-dimension of logistic regression is $>= d$. (Hint: come up with a set of d points in $\mathbb{R}^d$, and show that logistic regression can always shatter them.)

   (b) The VC-dimension of logistic regression is $< d + 1$. (Hint: let $x_1, \cdots, x_{d+1}$ be a set of $d + 1$ vectors in $\mathbb{R}^d$. Then, there must exist real numbers $a_1, \cdots, a_{d+1}$, not all of them are zero, such that $\sum_{i=1}^{d+1} a_i x_i = 0$.)

2. **Linearly separable Logistic Regression:** Given the conditional probability in Eqn. (2), the parameters of logistic regression is found by maximum likelihood estimation. That is, we maximize:

$$L(w) = \log \mathbb{P}(y|X; w) = \sum_{i=1}^{m} \Big[y_i \log \sigma(w^T x) + (1 - y_i) \log(1 - \sigma(w^T x)) \Big]$$

   (a) Suppose that your training data is linearly separable, what would happen when you try to find the parameters w? You should answer the question as formally as possible, e.g., constructing a simple example and/or show relevant plots to support your claim.

3. **Least Squared "Logistic Regression":** As a curious student, John always wants to innovate new algorithms. In particular, he does not like the approach of maximum likelihood for finding the parameters. Since John has learned linear regression and heard many good things about it in various applications, he decides to minimize the squared loss to find the parameters. He knows that this is not a good idea for multi-class problems ($>= 3$ classes), because then one is incorrectly assuming an implicit order between the labels. However, this is not the case here where only 0/1 classification is considered. The output of sigmoid is always in $[0, 1]$, and essentially, all what one needs is to make the probability of the positive class ($y = 1$) as high as possible (i.e., close to 1) and make the probability of the negative class ($y = 0$) as low as possible (i.e., close to 0). Therefore, John believes the squared loss should also be applicable, and he writes down the following loss function to minimize:

$$L(w) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \sigma(w^T x))^2 \tag{3}$$

   (a) Derive the gradient descent step for minimizing the squared loss in Eqn. (3).

   (b) Now that John has the gradient step for minimizing the squared loss, he is excited to run gradient descent and wants to sell his big idea to the instructors. As a close friend of him, *who studies optimization*, do you think this overall approach of finding parameters w is good or not? (Think from a computational perspective, i.e., what kind of optimization problem is easy/hard.) You should answer the question as formally as possible, e.g., constructing a simple example and/or show relevant plots to support your claim.

Introduction to Hands-On Section

The 2nd part of this assignment deals with hands-on implementation questions. We start with a bit of overview related 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 the hinge loss for SVMs. However, we review logistic loss as relevant background. 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 objective 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.)

The dataset we will evaluate our classifiers on is a set of 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 (available in the data folder).

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

1. Problem 3 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.
2. Problem 4 keeps the objective function from SVMs but seeks efficiency in large data sets by using a variation on stochastic gradient descent.

3 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?

4 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: It can be easily checked that the α variables are always non-negative. 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.

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 $y_i(\sum_j \alpha_j y_j K(x_j, x_i)) < 1$ **then**
      $\alpha_i \leftarrow (1 - \eta_t\lambda)\alpha_i + \eta_t$
    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, $K(x, x') = \exp(-\gamma\|x - x'\|^2)$, 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?