Matheus C. Fernandes
fernandes@g.harvard.edu
CS181-S17

Assignment #3
Due: 5:00pm March 24, 2016

Collaborators:
Bolei Deng
Callin Switzer
Joe Sedlak

Homework 3: Max-Margin and SVM

Introduction

This homework assignment will have you work with max-margin methods and SVM classification. The aim of the assignment is (1) to further develop your geometrical intuition behind margin-based classification and decision boundaries, (2) to explore the properties of kernels and how they provide a different form of feature development from basis functions, and finally (3) to implement a basic Kernel based classifier.

There is a mathematical component and a programming component to this homework. Please submit your PDF and Python files to Canvas, and push all of your work to your GitHub repository. If a question requires you to make any plots, like Problem 3, please include those in the writeup.

Problem 1 (Fitting an SVM by hand, 7pts)

For this problem you will solve an SVM without the help of a computer, relying instead on principled rules and properties of these classifiers.

Consider a dataset with the following 7 data points each with $x \in \mathbb{R}$:

$$\{(x_i, y_i)\}_i = \{(-3, +1), (-2, +1), (-1, -1), (0, -1), (1, -1), (2, +1), (3, +1)\}$$

Consider mapping these points to 2 dimensions using the feature vector $\boldsymbol{\phi}(x) = (x, x^2)$. The hard margin classifier training problem is:

$$\begin{aligned} &\min_{\mathbf{w}, w_0} \|\mathbf{w}\|_2^2 \\ \text{s.t.} \quad &y_i(\mathbf{w}^\top \boldsymbol{\phi}(x_i) + w_0) \geq 1, \ \forall i \in \{1, \ldots, n\} \end{aligned} \tag{1}$$

The exercise has been broken down into a series of questions, each providing a part of the solution. Make sure to follow the logical structure of the exercise when composing your answer and to justify each step.

1. Plot the training data in $\mathbb{R}^2$ and draw the decision boundary of the max margin classifer.
2. What is the value of the margin achieved by the optimal decision boundary?
3. What is a vector that is orthogonal to the decision boundary?
4. Considering discriminant $h(\boldsymbol{\phi}(x); \mathbf{w}, w_0) = \mathbf{w}^\top \boldsymbol{\phi}(x) + w_0$, give an expression for *all possible* $(\mathbf{w}, w_0)$ that define the optimal decision boundary. Justify your answer.
5. Consider now the training problem (1). Using your answers so far, what particular solution to $\mathbf{w}$ will be optimal for this optimization problem?
6. Now solve for the corresponding value of w_0, using your general expression from part (4.) for the optimal decision boundary. Write down the discriminant function $h(\boldsymbol{\phi}(x); \mathbf{w}, w_0)$.
7. What are the support vectors of the classifier? Confirm that the solution in part (6.) makes the constraints in (1) binding for support vectors.

Solution

Problem 1

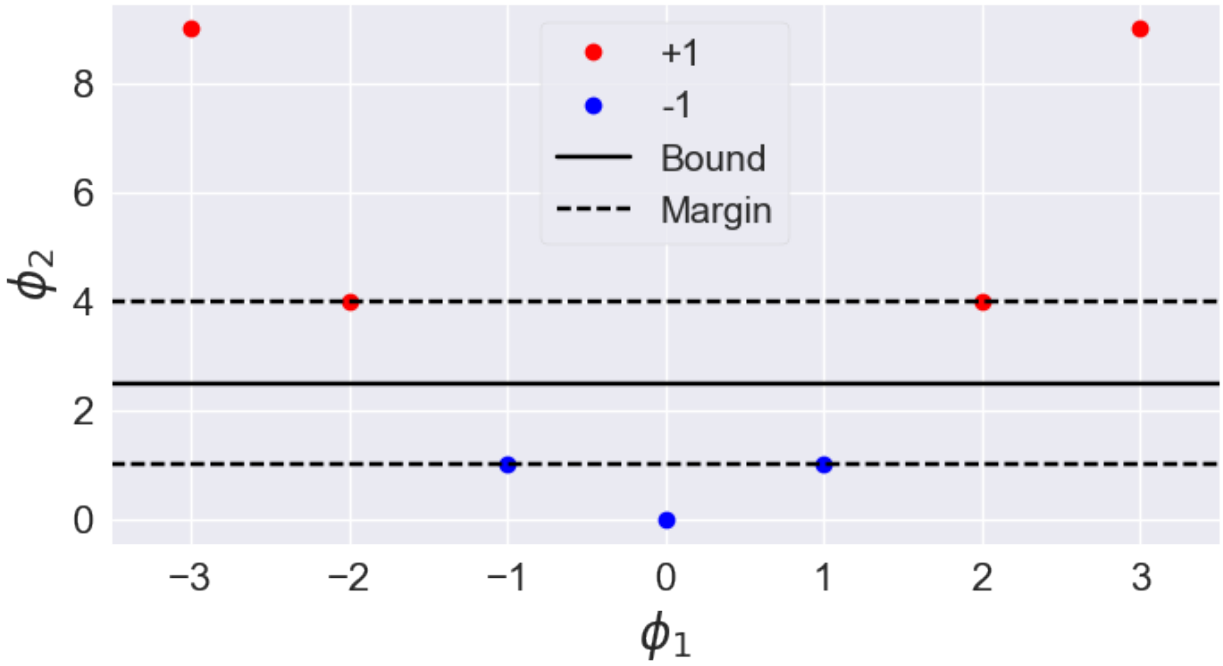

Figure 1: Training data in $\mathbb{R}^2$ with decision boundary and margin.

Problem 2

The value of the margin achieved by the optimal decision boundary is

$$\frac{y_i(\mathbf{w}^\top \boldsymbol{\phi}(x_i) + w_0)}{||\mathbf{w}||} = \boxed{1.5}$$

for an $\mathbf{x}_i$ laying on the margin. Furthermore, for the above result we clearly see that the margin width is 3.

Problem 3

The vector orthogonal to the decision boundary is given by $\mathbf{w}$ and in this case is of arbitrary magnitude such that:

$$\boxed{\mathbf{w} = \lambda[0, 1]^\top, \forall \lambda \in \mathbb{R}}$$

Problem 4

Considering the discriminant function and choosing the location to be in the decision boundary we can solve the discriminant function such that:

$$\mathbf{w}^\top \boldsymbol{\phi}(x) + w_0 = 0$$

We can expand the inner product to obtain

$$w_1 x + w_2 x^2 + w_0 = 0$$

where from inspection we see that the margin does not depend on x, thus we can set $w_1 \to 0$. We also know that the margin occurs at $x^2 = 2.5$ and thus obtain

$$w_2 * 2.5 + w_0 = 0$$

$$\boxed{w_2 = -\frac{w_0}{2.5}}$$

$$\boxed{w_1 = 0}$$

Note that there are other ways to achieve different equations between the $\mathbf{w}, w_0$, and can be achieved by solving the equation of the margin at the select support vectors knowing their respective values. However, there is only one optimum value for these parameters, which occurs when all of these equations intersect in within their respective spaces.

Problem 5 & 6

We know that there are 4 support vectors that can help us define all of the parameters we need. We also know that the margin does not depend on $\phi_1 = x$, meaning $w_1 = 0$ is always satisfied from $\mathbf{w} = [w_1, w_2]$. Now, we must find a relationship between w_1 and w_0. To do that, we resort to the margin equation:

$$\frac{y_i(\mathbf{w}^\top \boldsymbol{\phi}(x_i) + w_0)}{||\mathbf{w}||} = 1.5.$$

Furthermore, we know that for a point a on the margin, the numerator becomes 1, thus yielding the equation:

$$\frac{1}{||\mathbf{w}||} = \frac{1}{\sqrt{0^2 + w_2^2}} = 1.5 \Rightarrow w_2 = \frac{2}{3}.$$

Now, we also know that for a support vector, the discriminant $h(\boldsymbol{\phi}(x); \mathbf{w}, w_0) = \pm 1$, depending on whether the support vector is located above or below the margin. Therefore, assuming the SV along the right top margin, we obtain the equation constraint

$$-2w_1 + 4w_2 + w_0 = 1$$

where, from graphical inspection, we know that $w_1 = 0$. Therefore, we have a relationship in that $w_0 = 1 - 4w_2$. So for a $\boxed{w_2 = \frac{2}{3}}$ we have that $\boxed{w_0 = -\frac{5}{3}}$

Problem 7

The support vectors of the classifier are the following $\{(x_i, y_i)\}_i$:

$$\{(x, y)\}_{SV} = \{(-2, +1), (-1, -1), (1, -1), (2, +1)\}$$

Taking these points and plugging them into

$$y_i(\mathbf{w}^\top \boldsymbol{\phi}(x_i) + w_0) = y_i h(\boldsymbol{\phi}(x_i); \mathbf{w}, w_0)$$

where

$$\mathbf{w} = \left[0, \frac{2}{3}\right]^\top$$

$$w_0 = -\frac{5}{3}$$

$$\boldsymbol{\phi}(\mathbf{x}) = \left[x, x^2\right]^\top$$

we get

$$y_i h(\boldsymbol{\phi}(-2); \mathbf{w}, w_0) = (1) * (1) = 1 \geq 1$$

$$y_i h(\boldsymbol{\phi}(-1); \mathbf{w}, w_0) = (-1) * (-1) = 1 \geq 1$$

$$y_i h(\boldsymbol{\phi}(1); \mathbf{w}, w_0) = (-1) * (-1) = 1 \geq 1$$

$$y_i h(\boldsymbol{\phi}(2); \mathbf{w}, w_0) = (1) * (1) = 1 \geq 1$$

Thus perfectly satisfying the above constraint posed by the problem, as all of the other points will have a greater value since they are even farther away from the decision boundary.

■

Problem 2 (Composing Kernel Functions, 10pts)

A key benefit of SVM training is the ability to use kernel functions $K(\mathbf{x}, \mathbf{x}')$ as opposed to explicit basis functions $\phi(\mathbf{x})$. Kernels make it possible to implicitly express large or even infinite dimensional basis features. We do this by computing $\phi(\mathbf{x})^\top \phi(\mathbf{x}')$ directly, without ever computing $\phi(\mathbf{x})$.

When training SVMs, we begin by computing the kernel matrix $\mathbf{K}$, over our training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. The kernel matrix, defined as $K_{i,i'} = K(\mathbf{x}_i, \mathbf{x}_{i'})$, expresses the kernel function applied between all pairs of training points.

In class, we saw Mercer's theorem, which tells us that any function K that yields a positive semi-definite kernel matrix forms a valid kernel, i.e. corresponds to a matrix of dot-products under *some* basis ϕ. Therefore instead of using an explicit basis, we can build kernel functions directly that fulfill this property.

A particularly nice benefit of this theorem is that it allows us to build more expressive kernels by composition. In this problem, you are tasked with using Mercer's theorem and the definition of a kernel matrix to prove that the following compositions are valid kernels, assuming $K^{(1)}$ and $K^{(2)}$ are valid kernels. Recall that a positive semi-definite matrix $\mathbf{K}$ requires $\mathbf{z}^\top \mathbf{K} \mathbf{z} \geq 0, \ \forall \ \mathbf{z} \in \mathbb{R}^n$.

1. $K(\mathbf{x}, \mathbf{x}') = c\, K^{(1)}(\mathbf{x}, \mathbf{x}') \quad$ for $c > 0$
2. $K(\mathbf{x}, \mathbf{x}') = K^{(1)}(\mathbf{x}, \mathbf{x}') + K^{(2)}(\mathbf{x}, \mathbf{x}')$
3. $K(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})\, K^{(1)}(\mathbf{x}, \mathbf{x}')\, f(\mathbf{x}') \quad$ where f is any function from $\mathbb{R}^m$ to $\mathbb{R}$
4. $K(\mathbf{x}, \mathbf{x}') = K^{(1)}(\mathbf{x}, \mathbf{x}')\, K^{(2)}(\mathbf{x}, \mathbf{x}')$

   [Hint: Use the property that for any $\phi(\mathbf{x})$, $K(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}')$ forms a positive semi-definite kernel matrix.]
5. (a) The $\exp$ function can be written as,

$$\exp(x) = \lim_{i \to \infty} \left(1 + x + \cdots + \frac{x^i}{i!}\right).$$

   Use this to show that $\exp(xx')$ (here $x, x' \in \mathbb{R}$)) can be written as $\phi(x)^\top \phi(x')$ for some basis function $\phi(x)$. Derive this basis function, and explain why this would be hard to use as a basis in standard logistic regression.

   (b) Using the previous identities, show that $K(\mathbf{x}, \mathbf{x}') = \exp(K^{(1)}(\mathbf{x}, \mathbf{x}'))$ is a valid kernel.
6. Finally use this analysis and previous identities to prove the validity of the Gaussian kernel:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(\frac{-||\mathbf{x} - \mathbf{x}'||_2^2}{2\sigma^2}\right)$$

Solution

Problem 1

Multiplying it out by vectors on both sides such that:

$$\mathbf{z}^\top c K(\mathbf{x}, \mathbf{x}')^{(1)} \mathbf{z} = c \mathbf{z}^\top K(\mathbf{x}, \mathbf{x}')^{(1)} \mathbf{z} \geq 0.$$

we can factor c out of the matrix-vector multiplication as it is a constant and does not affect the operation.

■

Problem 2

If each individual $\mathbf{K}$ we know it satisfies that it is a positive semi-definite matrix. Thus we can write it such that:

$$K(\mathbf{x}, \mathbf{x}') \rightarrow \mathbf{z}^\top K^{(1)}(\mathbf{x}, \mathbf{x}')\mathbf{z} + \mathbf{z}^\top K^{(2)}(\mathbf{x}, \mathbf{x}')\mathbf{z}$$

which we can factor out the $\mathbf{z}$'s to give

$$\mathbf{z}^\top (K^{(1)}(\mathbf{x}, \mathbf{x}') + K^{(2)}(\mathbf{x}, \mathbf{x}'))\mathbf{z} = \mathbf{z}^\top (K(\mathbf{x}, \mathbf{x}'))\mathbf{z} \leq 0$$

■

Problem 3

For this problem we can decompose $K^{(1)}(\mathbf{x}, \mathbf{x}')$ into

$$K^{(1)}(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}^{(1)}(\mathbf{x})\boldsymbol{\phi}^{(1)}(\mathbf{x}')$$

with that we can re-write the equation as

$$f(\mathbf{x})\boldsymbol{\phi}^{(1)}(\mathbf{x})\boldsymbol{\phi}^{(1)}(\mathbf{x}')f(\mathbf{x}')$$

where we can combine both sides such that

$$[f(\mathbf{x})\boldsymbol{\phi}^{(1)}(\mathbf{x})][f(\mathbf{x}')\boldsymbol{\phi}^{(1)}(\mathbf{x}')]$$

and where we can define a new basis function

$$\boldsymbol{\phi}^{(2)}(\mathbf{x}) = [f(\mathbf{x})\boldsymbol{\phi}^{(1)}(\mathbf{x})].$$

Thus,

$$K(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}^{(2)}(\mathbf{x})\boldsymbol{\phi}^{(2)}(\mathbf{x}')$$

■

Problem 4

To prove this portion we must begin by considering each K by itself and convert it into its respective basis functions such that:

$$K(\mathbf{x}, \mathbf{x}') = K^{(1)}(\mathbf{x}, \mathbf{x}')K^{(2)}(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}^{(1)}(\mathbf{x})^\top \boldsymbol{\phi}^{(1)}(\mathbf{x}')\boldsymbol{\phi}^{(2)}(\mathbf{x})^\top \boldsymbol{\phi}^{(2)}(\mathbf{x}')$$

multiplying this out we obtain the following sum:

$$K(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{n}\sum_{j=1}^{m} \phi_i^{(1)}(\mathbf{x})\phi_i^{(1)}(\mathbf{x}')\phi_j^{(2)}(\mathbf{x})\phi_j^{(2)}(\mathbf{x}')$$

thus we can separate this into two multiplications such that we have two separate vectors multiplying each other and adding

$$K(\mathbf{x},\mathbf{x}') = \underbrace{\begin{bmatrix}\phi_1^{(1)}(\mathbf{x})\phi_1^{(2)}(\mathbf{x}) & \phi_1^{(1)}(\mathbf{x})\phi_2^{(2)}(\mathbf{x}) & \dots & \phi_1^{(1)}(\mathbf{x})\phi_m^{(2)}(\mathbf{x}) & \phi_2^{(1)}(\mathbf{x})\phi_1^{(2)}(\mathbf{x}) & \dots & \phi_n^{(1)}(\mathbf{x})\phi_m^{(2)}(\mathbf{x})\end{bmatrix}}_{n\times m}\begin{bmatrix}\phi_1^{(1)}(\mathbf{x}')\phi_1^{(2)}(\mathbf{x}')\\ \phi_1^{(1)}(\mathbf{x}')\phi_2^{(2)}(\mathbf{x}')\\ \dots\\ \phi_1^{(1)}(\mathbf{x}')\phi_m^{(2)}(\mathbf{x}')\\ \phi_2^{(1)}(\mathbf{x}')\phi_1^{(2)}(\mathbf{x}')\\ \dots\\ \phi_n^{(1)}(\mathbf{x}')\phi_m^{(2)}(\mathbf{x}')\end{bmatrix}$$

where n is the number of features in $K^{(1)}$ and m is the number of features in $K^{(2)}$. Here we can define a new basis function

$$\phi^{(3)}(\mathbf{x}) = \begin{bmatrix}\phi_1^{(1)}(\mathbf{x})\phi_1^{(2)}(\mathbf{x}) & \phi_1^{(1)}(\mathbf{x})\phi_2^{(2)}(\mathbf{x}) & \dots & \phi_1^{(1)}(\mathbf{x})\phi_m^{(2)}(\mathbf{x}) & \phi_2^{(1)}(\mathbf{x})\phi_1^{(2)}(\mathbf{x}) & \dots & \phi_n^{(1)}(\mathbf{x})\phi_m^{(2)}(\mathbf{x})\end{bmatrix}^\top$$

such that

$$K(\mathbf{x},\mathbf{x}') = \phi^{(3)}(\mathbf{x})^\top\phi^{(3)}(\mathbf{x}')$$

and from the hint, if we know that K takes this form, we know it to be a valid kernel.

■

Problem 5

Part (a)

For this problem we know the series expansion for the exponential function above. However, if we need to separate the series expansion into a multiplication of expansions, we obtain that:

$$\exp(\mathbf{x}\mathbf{x}') = \begin{bmatrix}1 & x & \frac{x^2}{\sqrt{2!}} & \dots & \frac{x^i}{\sqrt{i!}}\end{bmatrix}\begin{bmatrix}1\\ x'\\ \frac{x'^2}{\sqrt{2!}}\\ \dots\\ \frac{x'^i}{\sqrt{i!}}\end{bmatrix}$$

such that

$$\phi(x) = \sum_{i=0}^{\infty}\frac{x^i}{\sqrt{i!}}$$

and

$$\exp(\mathbf{x}\mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^\top\boldsymbol{\phi}(\mathbf{x}').$$

Where from hint of part 4, we know that this yields a valid positive semi-definite kernel matrix.

■

Part (b)

We know that if

$$\mathbf{K}(\mathbf{x},\mathbf{x}')^{(1)} = \boldsymbol{\phi}(\mathbf{x})^{(1)\top}\boldsymbol{\phi}(\mathbf{x}')^{(1)}$$

then,

$$\exp(\boldsymbol{\phi}(\mathbf{x})^{(1)\top}\boldsymbol{\phi}(\mathbf{x}')^{(1)}) = \begin{bmatrix} 1 & \boldsymbol{\phi}(\mathbf{x})^{(1)\top} & \frac{\boldsymbol{\phi}(\mathbf{x})^{(1)\top 2}}{\sqrt{2!}} & \dots & \frac{\boldsymbol{\phi}(\mathbf{x})^{(1)\top i}}{\sqrt{i!}} \end{bmatrix} \begin{bmatrix} 1 \\ \boldsymbol{\phi}(\mathbf{x}')^{(1)\top} \\ \frac{\boldsymbol{\phi}(\mathbf{x}')^{(1)\top 2}}{\sqrt{2!}} \\ \dots \\ \frac{\boldsymbol{\phi}(\mathbf{x}')^{(1)\top i}}{\sqrt{i!}} \end{bmatrix}$$

which we can then use such that

$$\boldsymbol{\phi}(x)^{(2)} = \sum_{i=0}^{\infty} \frac{x^i}{\sqrt{i!}}.$$

Thus,

$$\exp(\boldsymbol{\phi}(\mathbf{x})^{(1)\top}\boldsymbol{\phi}(\mathbf{x}')^{(1)}) = \boldsymbol{\phi}(\boldsymbol{\phi}(\mathbf{x})^{(1)})^{(2)\top}\boldsymbol{\phi}(\boldsymbol{\phi}(\mathbf{x}')^{(1)})^{(2)}$$

Where from hint of part 4, we know that this yields a valid positive semi-definite kernel matrix.

■

Problem 6

From the identity:

$$\begin{aligned} ||x-y||^2 &= \langle x-y, x-y\rangle \\ &= \|x\|^2 - \langle x, y\rangle - \langle y, x\rangle + \|y\|^2 \\ &\leq \|x\|^2 - 2|\langle x, y\rangle| + \|y\|^2 \\ &\leq \|x\|^2 - 2\|x\|\|y\| + \|y\|^2 \\ &= ||x||^2 + ||y||^2 - 2xy \end{aligned}$$

Then we can take:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(\frac{-||\mathbf{x}-\mathbf{x}'||_2^2}{2\sigma^2}\right)$$

and decompose it into

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(\frac{-||\mathbf{x}-\mathbf{x}'||_2^2}{2\sigma^2}\right) = \exp\left(-\frac{||\mathbf{x}||}{2\sigma^2}\right)\exp\left(-\frac{||\mathbf{x}'||}{2\sigma^2}\right)\exp\left(\frac{\mathbf{x}\mathbf{x}'}{\sigma^2}\right)$$

which we know from previous identities that

$$\exp(\mathbf{x}\mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^\top\boldsymbol{\phi}(\mathbf{x}')$$

and that $\exp$ is a valid kernel. Thus also from problem 4 we know that multiplying valid kernels yields a total valid kernel for the above expression.

■

Problem 3 (Scaling up your SVM solver, 10pts (+opportunity for extra credit))

For this problem you will build a simple SVM classifier for a binary classification problem. We have provided you two files for experimentation: training *data.csv* and validation *val.csv*.

- First read the paper at http://www.jmlr.org/papers/volume6/bordes05a/bordes05a.pdf and implement the Kernel Perceptron algorithm and the Budget Kernel Perceptron algorithm. Aim to make the optimization as fast as possible. Implement this algorithm in *problem3.py*.

  [Hint: For this problem, efficiency will be an issue. Instead of directly implementing this algorithm using numpy matrices, you should utilize Python dictionaries to represent sparse matrices. This will be necessary to have the algorithm run in a reasonable amount of time.]

- Next experiment with the hyperparameters for each of these models. Try seeing if you can identify some patterns by changing β, N (the maximum number of support vectors), or the number of random training samples taken during the Randomized Search procedure (Section 4.3). Note the training time, training and validation accuracy, and number of support vectors for various setups.

- Lastly, compare the classification to the naive SVM imported from scikit-learn by reporting accuracy on the provided validation data. *For extra credit, implement the SMO algorithm and implement the LASVM process and do the same as above.*[a]

We are intentionally leaving this problem open-ended to allow for experimentation, and so we will be looking for your thought process and not a particular graph. Visualizations should be generated using the provided code. You can use the trivial $K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$ kernel for this problem, though you are welcome to experiment with more interesting kernels too.

In addition, provide answers the following reading questions **in one or two sentences for each**.

1. In one short sentence, state the main purpose of the paper.
2. Describe each of the parameters in Eq. 1 in the paper
3. State, informally, one guarantee about the Kernel perceptron algorithm described in the paper.
4. What is the main way the budget kernel perceptron algorithm tries to improve on the perceptron algorithm?
5. (*if you did the extra credit*) In simple words, what is the theoretical guarantee of LASVM algorithm? How does it compare to its practical performance?

[a] Extra credit only makes a difference to your grade at the end of the semester if you are on a grade boundary.

Solution

(1)

The purpose of this paper is to survey different ways to improve fast kernel classifiers and introduce various ways to allocate attention to the training data without compromising the fact that we are taking into consideration all of the examples.

(2)

The w' is a weight vector parameter consisting of weights that tells us how important each feature of the vector $\phi(x)$ is. The b parameter is known as the bias parameter and allows the data to be corrected in space by a constant denoted by this parameter. The $\phi(x)$ are the basis functions and dictate the functional form of x for the discriminate function. $\hat{y}(x)$ is the discriminant function end result.

(3)

When a solution exists, the perceptron algorithm is guaranteed to converge after a finite number of iterations, or equivalently after inserting a finite number of support vectors. Meaning, the there is a mathematical proof that shows we can achieve the exact convergence of a solution without requiring infinite iterations of the code, but instead only a finite number.

(4)

The budget kernel perceptron algorithm tries to improve on the perceptron algorithm by improving it's performance on noisy datasets – this is due to the fact that the number of support vectors increases very rapidly and potentially causes over fitting and poor convergence. This avoids such problem by educatedly removing support vectors from the list that have a large bound given by the argmax term in 4b as it happily ignores training examples that are very close to being misclassified.

(5)

The LASVM is a faster yet more substantial way that is guaranteed convergence to the normal SVM after a single sequential pass over the training example. Thus, the ultimate guarantee of this method is convergence of online interations. In terms of performance, this method will outperform the SVM's if implemented correctly while guaranteeing convergence. For me, when implementing SMO, my code takes way longer due to inefficiencies of nested for loops as well as other redundant loops. For my code it can take almost 10 times as long as the regular SVM code implemented.

Hyperparameters Experimentation

By playing with the hyperparameters, I noticed that changes in the number of support vectors, training time, and accuracy occur. When tweaking the β parameter I noticed that as it is increased, we see that the accuracy of the budget kernel algorithm is decreased. Furthermore, the time it takes for it to converge is also increased as now we have to check more parameters through the for loop. This analysis was conducted holding N=100 and number of samples =1000 fixed. Another thing noticed was that the number of support vectors is also increased as β is increased. However, because N=100, the maximum number of support vectors is always lower or equal to 100 for any value of β.

Another parameter that was tweaked was N, the number of maximum support vector for any given iteration. Here, as expected, the lower N, the lower the accuracy as well as the time it takes to train. As N is increased, the accuracy is also increased up to 99.71% for N=100. Here we also assume remain constant at β=20. Another thing noticed, is that when N is small, the variance in the accuracy varies largely between different runs. This is expected as we are sampling random selected points and each run we are sampling different random points generated trough the indexes. Also, as we increase N, the code takes longer to run.

The third and last parameter to play with is the number of random training samples. For this, as expected, as we increase the number of samples we see a large increase in the time it takes the code to run. Also, for very small numbers of samples the accuracy is small and has a large variance between different runs. However, as the number of random training samples is increased, we see that the average accuracy is increased and the variance between simulations is also decreased. Of course, if we sample the subspace to the most number of points computationally possible, we would be able to achieve perfect accuracy – however, due to computational limitation, we need to balance the time it takes to obtain a reasonable result with low variance. When this is compared to the SKLearn, we see that we cannot obtain an accuracy and consistency to such level without compromising the computational time. In general, however, in some instances, we are able to achieve better accuracy than achieved by SKLearn when using extremely large number of samples. Though, this yielded an order of magnitude larger time than for SKLearn and a very marginal improvement from their result.

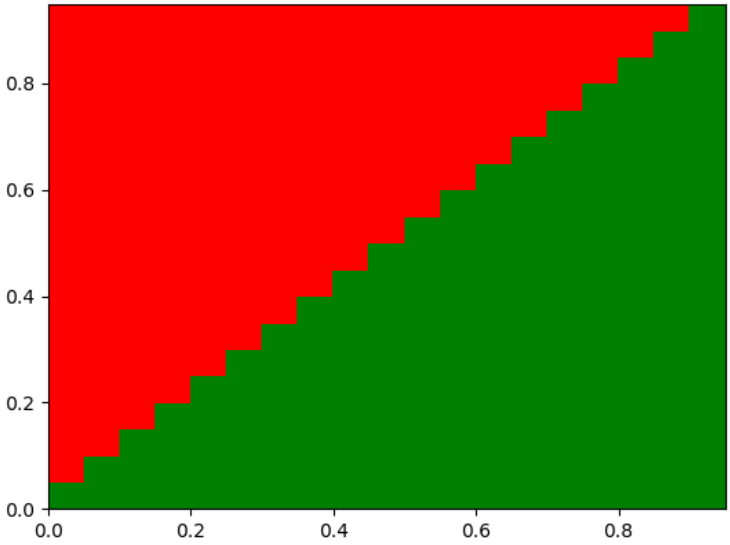

Figure 2: Kernel Optimal result with 100 % accuracy

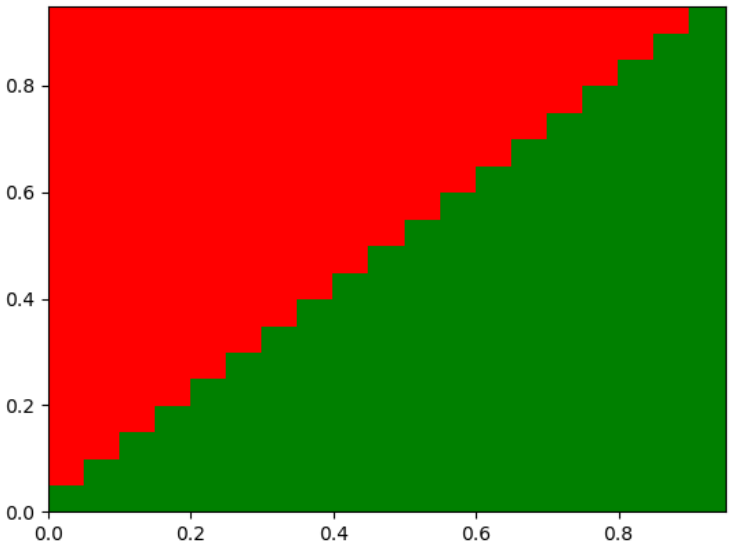

Figure 3: Budget Kernel Optimal result with 100 % accuracy

Codes for Problem 3

```
# CS 181, Harvard University
# Spring 2016
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as c
from Perceptron import Perceptron
import random
import time
from sklearn import svm

# Implement this class
class KernelPerceptron(Perceptron):
    def __init__(self, numsamples):
        self.numsamples = numsamples

    def __Kernel1(self,X,XP):
        return np.dot(X,XP)

    def __Kernel(self,X,XP):
        return self.__Kernel1(X,XP)

    def __CheckAccuracy(self,X,Y):
        v=self.predict(X)
        print 'Percent correct:',100-100*sum(abs(v-Y))/(2*len(Y))

    def __ImportValidationSet(self):
        data = np.loadtxt("data.csv", delimiter=',')
        X = data[:, :2]
        Y = data[:, 2]
        return X,Y
```

```
31
32      def __SciKitLearn(self,XTest,YTest,X,Y):
33          clf = svm.SVC()
34          bt=time.time()
35          clf.fit(X,Y)
36          print 'SciKit Learn Training Time:', time.time() - bt
37          v=clf.predict(XTest)
38          print 'SciKit Learn Percent correct:',100-100*sum(abs(v-YTest))/(2*len(YTest))
39
40
41      # Implement this!
42      def fit(self, X, Y):
43          print 'Fitting Kernel Perceptron'
44          print '_____'
45          print '_____'
46
47          bt=time.time()
48          numSamples=self.numsamples
49          numOfPoints=len(X)
50
51          numOfTrain=int(2./3.*numOfPoints)
52          XTest=X[numOfTrain:]
53          YTest=Y[numOfTrain:]
54
55          X=X[:numOfTrain]
56          Y=Y[:numOfTrain]
57
58          self.X=X
59          self.Y=Y
60
61          S=[]; alpha={x:0 for x in xrange(numOfTrain)}
62
63          for num in xrange(numSamples):
64
65              t=random.randint(0,numOfTrain-1)
66              Xt=X[t];Yt=Y[t]
67              YHatT=0
68
69              for i in S:
70                  YHatT+=alpha[i]*self.__Kernel(Xt,X[i])
71
72              if Yt*YHatT <=0:
73                  S.append(t)
74                  alpha[t]=Yt
75
76          w=[0,0]
77
78          for i in xrange(numOfTrain):
79              for j in xrange(len(X[i])):
80                  w[j]+=alpha[i]*X[i,j]
81
82          self.w=w
83          print 'Fitting Kernel Perceptron Done!'
84          print 'Training Time:',time.time()-bt
85          print 'Number of Support Vectors:',len(S)
86
87          self.__CheckAccuracy(XTest,YTest)
88          self.__SciKitLearn(XTest,YTest,X,Y)
89
90          print '_____'
91          print 'Using Validation Set'
92          [XValidation,YValidation]=self.__ImportValidationSet()
93          self.__CheckAccuracy(XValidation,YValidation)
94          self.__SciKitLearn(XValidation,YValidation,X,Y)
95          print '_____'
```

```
96
97
98
99      # Implement this!
100     def predict(self, X):
101         v=np.sign(np.dot(np.transpose(self.w), np.transpose(X)))
102         v[v==0]=1
103         return v
104
105
106 # Implement this class
107 class BudgetKernelPerceptron(Perceptron):
108     def __init__(self, beta, N, numsamples):
109         self.beta = beta
110         self.N = N
111         self.numsamples = numsamples
112
113     def __Kernel1(self,X,XP):
114         return np.dot(X,XP)
115
116     def __Kernel(self,X,XP):
117         return self.__Kernel1(X,XP)
118
119     def __CheckAccuracy(self,X,Y):
120         v=self.predict(X)
121         print 'Percent correct:',100-100*sum(abs(v-Y))/(2*len(Y))
122
123
124     def __SciKitLearn(self, XTest, YTest, X, Y):
125         clf = svm.SVC()
126         bt = time.time()
127         clf.fit(X, Y)
128         print 'SciKit Learn Training Time:', time.time() - bt
129         v = clf.predict(XTest)
130         print 'SciKit Learn Percent correct:', 100 - 100 * sum(abs(v - YTest)) / (2 * len(YTest))
131
132     def __ImportValidationSet(self):
133         data = np.loadtxt("data.csv", delimiter=',')
134         X = data[:, :2]
135         Y = data[:, 2]
136         return X,Y
137
138
139     # Implement this!
140     def fit(self, X, Y):
141         print ''
142         print ''
143         print 'Fitting Budged Kernel Perceptron'
144         print '-----------------------------------------'
145         print '-----------------------------------------'
146         bt=time.time()
147         numSamples=self.numsamples
148         beta=self.beta
149         N=self.N
150
151         numOfPoints=len(X)
152
153         numOfTrain=int(2./3.*numOfPoints)
154         XTest=X[numOfTrain:]
155         YTest=Y[numOfTrain:]
156
157         X=X[:numOfTrain]
158         Y=Y[:numOfTrain]
159
160         self.X=X
```

```
        self.Y=Y


        S=[]; alpha={x:0 for x in xrange(numOfTrain)}

        for num in xrange(numSamples):

            t=random.randint(0,numOfTrain-1)
            Xt=X[t];Yt=Y[t]
            YHatT=0

            for i in S:
                YHatT+=alpha[i]*self.__Kernel(Xt,X[i])

            if Yt*YHatT<=beta:
                S.append(t)
                alpha[t]=Yt

                if len(S)>N:
                    YinS=[]

                    for i in S:
                        YHatTi=0
                        for j in S:
                            YHatTi+=alpha[j]*self.__Kernel(X[i],X[j])

                        YinS.append(Y[i]*(YHatTi-alpha[i]*self.__Kernel(X[i],X[i])))

                    ind=np.argmax(YinS)

                    del S[ind]
                    alpha[ind]=0

        w=[0,0]

        for i in xrange(numOfTrain):
            for j in xrange(len(X[i])):
                w[j] += alpha[i] * X[i, j]

        self.w = w
        print 'Fitting Budged Kernel Perceptron Done!'
        print 'Training Time:',time.time()-bt
        print 'Number of Support Vectors:',len(S)

        self.__CheckAccuracy(XTest,YTest)
        self.__SciKitLearn(XTest,YTest,X,Y)

        print '-----------------------------------------'
        print 'Using Validation Set'
        [XValidation,YValidation]=self.__ImportValidationSet()
        self.__CheckAccuracy(XValidation,YValidation)
        self.__SciKitLearn(XValidation,YValidation,X,Y)
        print '-----------------------------------------'


    # Implement this!
    def predict(self, X):
        v=np.sign(np.dot(np.transpose(self.w), np.transpose(X)))
        v[v==0]=1
        return v

# Do not change these three lines.
data = np.loadtxt("data.csv", delimiter=',')
X = data[:, :2]
Y = data[:, 2]
```

```
226
227 # These are the parameters for the models. Please play with these and note your observations
        about speed and successful hyperplane formation.
228 beta = 0.5
229 N = 100
230 numsamples = 5000
231
232 kernel_file_name = 'k.png'
233 budget_kernel_file_name = 'bk.png'
234
235 # Don't change things below this in your final version. Note that you can use the parameters
        above to generate multiple graphs if you want to include them in your writeup.
236 k = KernelPerceptron(numsamples)
237 k.fit(X,Y)
238 k.visualize(kernel_file_name, width=0, show_charts=False, save_fig=True, include_points=False)
239
240 bk = BudgetKernelPerceptron(beta, N, numsamples)
241 bk.fit(X, Y)
242 bk.visualize(budget_kernel_file_name, width=0, show_charts=False, save_fig=True, include_points=
        False)
```

Listing 1: Problem 3 Code

```
1  # CS 181, Harvard University
2  # Spring 2016
3  import numpy as np
4  import matplotlib.pyplot as plt
5  import matplotlib.colors as c
6  from Perceptron import Perceptron
7  import random
8  import time
9  from sklearn import svm
10
11 # Implement this class
12 class SMO(Perceptron):
13     def __init__(self, numsamples):
14         self.numsamples = numsamples
15
16     def __Kernel1(self,X,XP):
17         return np.dot(X,XP)
18
19     def __Kernel(self,X,XP):
20         return self.__Kernel1(X,XP)
21
22     def __CheckAccuracy(self,X,Y):
23         v=self.predict(X)
24         print 'Percent correct:',100-100*sum(abs(v-Y))/(2*len(Y))
25
26     def __ImportValidationSet(self):
27         data = np.loadtxt("data.csv", delimiter=',')
28         X = data[:, :2]
29         Y = data[:, 2]
30         return X,Y
31
32     def __ComputeGradient(self,X,Y,alpha,k):
33         yHat=0
34         for i in xrange(len(alpha)):
35             yHat+=alpha[i]*self.__Kernel(X[k],X[i])
36         g=Y[k]-yHat
37         return g
38
39     def __SciKitLearn(self,XTest,YTest,X,Y):
40         clf = svm.SVC()
41         bt=time.time()
```

```
        clf.fit(X,Y)
        print 'SciKit Learn Training Time:', time.time() - bt
        v=clf.predict(XTest)
        print 'SciKit Learn Percent correct:',100-100*sum(abs(v-YTest))/(2*len(YTest))


    # Implement this!
    def fit(self, X, Y):
        print 'Fitting SMO'
        print '---------------------------------------'
        print '---------------------------------------'

        Tau=0.1
        C=5

        breakFlag=True

        bt=time.time()
        numSamples=self.numsamples
        numOfPoints=len(X)

        numOfTrain=int(1./3.*numOfPoints)
        XTest=X[numOfTrain:]
        YTest=Y[numOfTrain:]

        X=X[:numOfTrain]
        Y=Y[:numOfTrain]

        self.X=X
        self.Y=Y

        alpha={x:0 for x in xrange(numOfTrain)}
        A=np.zeros((len(Y),1));B=np.zeros((len(Y),1))
        for i in range(len(Y)):
            A[i]=np.min([0,Y[i]*C])
            B[i]=np.max([0,Y[i]*C])

        g=np.zeros((len(X),1))
        for k in range(len(X)):
            g[k]=self.__ComputeGradient(X, Y, alpha,k)

        while breakFlag:
            breakFlagInside=True
            for i in xrange(len(X)):
                for j in xrange(len(X)):
                    if alpha[i]<B[i] and alpha[j]>A[j] and g[i]-g[j]>Tau:
                        minVec=[(g[i]-g[j])/(self.__Kernel(X[i],X[i])+self.__Kernel(X[j],X[j])+2*
    self.__Kernel(X[i],X[j]))]
                        minVec.append(B[i]-alpha[i])
                        minVec.append(alpha[i]-A[i])
                        lam=min(minVec)
                        alpha[i]=alpha[i]+lam
                        alpha[j]=alpha[j]-lam
                        for s in range(len(X)):
                            g[s] += -lam*(self.__Kernel(X[i],X[s])-self.__Kernel(X[j],X[s]))
                        breakFlagInside=False
            if breakFlagInside:
                breakFlag=False


        for i in xrange(numOfTrain):
            for j in xrange(len(X[i])):
                w[j]+=alpha[i]*X[i,j]

        self.w=w
```

```
        print 'Fitting Kernel Perceptron Done!'
        print 'Training Time:',time.time()-bt
        print 'Number of Support Vectors:',len(S)

        self.__CheckAccuracy(XTest,YTest)
        self.__SciKitLearn(XTest,YTest,X,Y)

        print '----------------------------------------'
        print 'Using Validation Set'
        [XValidation,YValidation]=self.__ImportValidationSet()
        self.__CheckAccuracy(XValidation,YValidation)
        self.__SciKitLearn(XValidation,YValidation,X,Y)
        print '----------------------------------------'


    # Implement this!
    def predict(self, X):
        v=np.sign(np.dot(np.transpose(self.w), np.transpose(X)))
        v[v==0]=1
        return v



# Do not change these three lines.
data = np.loadtxt("data.csv", delimiter=',')
X = data[:, :2]
Y = data[:, 2]

# These are the parameters for the models. Please play with these and note your observations about speed and successful hyperplane formation.

kernel_file_name = 'SMO.png'

# Don't change things below this in your final version. Note that you can use the parameters above to generate multiple graphs if you want to include them in your writeup.
k = SMO(numsamples)
k.fit(X,Y)
k.visualize(kernel_file_name, width=0, show_charts=False, save_fig=True, include_points=False)
```

Listing 2: Problem 3 Extra Credit Code

```
# CS 181, Harvard University
# Spring 2016
# Author: Ankit Gupta
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as c

# This is the base Perceptron class. Do not modify this file.
# You will inherit this class in your implementations of KernelPerceptron and BudgetKernelPerceptron.
class Perceptron(object):
  def __init__(self, numsamples):
    self.numsamples = numsamples

  def fit(self, X, Y):
    self.X = X
    self.Y = Y
    assert(X.shape[0] == Y.shape[0])

  def predict(self, X):

    # This is a temporary predict function so that the distribution code compiles
    # You should delete this and write your own
```

```
    return (X[:, 0] > .5)

  # Do not modify this method!
  def visualize(self, output_file, width=3, show_charts=False, save_fig=True, include_points=True
    ):
    X = self.X

    # Create a grid of points
    x_min, x_max = min(X[:, 0] - width), max(X[:, 0] + width)
    y_min, y_max = min(X[:, 1] - width), max(X[:, 1] + width)
    xx,yy = np.meshgrid(np.arange(x_min, x_max, .05), np.arange(y_min,
        y_max, .05))

    # Flatten the grid so the values match spec for self.predict
    xx_flat = xx.flatten()
    yy_flat = yy.flatten()
    X_topredict = np.vstack((xx_flat,yy_flat)).T

    # Get the class predictions
    Y_hat = self.predict(X_topredict)
    Y_hat = Y_hat.reshape((xx.shape[0], xx.shape[1]))

    cMap = c.ListedColormap(['r','b','g'])

    # Visualize them.
    plt.figure()
    plt.pcolormesh(xx,yy,Y_hat, cmap=cMap)
    if include_points:
      plt.scatter(X[:, 0], X[:, 1], c=self.Y, cmap=cMap, facecolor='0.5')
    if save_fig:
      plt.savefig(output_file)
    if show_charts:
      plt.show()
```

Listing 3: Perceptron Code

Calibration [1pt]

Approximately how long did this homework take you to complete?

10 hours