Homework 9

Due: Thursday, March 25, 2021 at 12:00pm **(Noon)**

Written Assignment

Problem 1: Kernels (35 points)

In lecture, we described a kernel $K(x, y) = \langle \psi(x), \psi(y) \rangle$ as a function that allows us to compute inner products in high dimensions efficiently. However, some functions $K : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ are not valid kernels for any mapping $\psi : \mathbb{R}^d \mapsto \mathbb{R}^e$. While we can sometimes easily find the specific ψ that proves K is a valid kernel, it is often easier to prove that K is a valid kernel without explicitly finding ψ.

To determine if there exists some mapping ψ for which K is a valid kernel, we define the **Gram matrix** for a kernel function K and a dataset $x_1, ..., x_n \in \mathbb{R}^d$ to be:

$$G = \begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \dots & K(x_1, x_n) \\ K(x_2, x_1) & K(x_2, x_2) & \dots & K(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(x_n, x_1) & K(x_n, x_2) & \dots & K(x_n, x_n) \end{bmatrix}$$

K is a valid kernel if and only if G is a symmetric and **positive semidefinite** matrix for all possible datasets. Furthermore, G is a positive semidefinite matrix if and only if $a^T G a \geq 0$ for all $a \in \mathbb{R}^n$.

In this problem, you will prove that the polynomial kernel $K(x, y) = (b\langle x, y \rangle + c)^d$ is a valid kernel function when $b > 0$, $c \geq 0$, and $d \in \mathbb{Z}^+$.

a. Prove that the linear kernel $K(x, y) = \langle x, y \rangle$ is a valid kernel.

b. Prove that if K_1 is a valid kernel and $b > 0$, then $K(x, y) = bK_1(x, y)$ is a valid kernel, by showing that the Gram matrix of K is positive semidefinite.

c. Prove that if K_1 is a valid kernel and $c \geq 0$, then $K(x, y) = K_1(x, y) + c$ is a valid kernel. *Hint: Let $\psi_1(x)$ be the mapping for K_1. Think about how you could modify $\psi_1(x)$ to make another mapping, $\psi(x)$, for which K is a valid kernel.*

d. It is also true that if K_1 and K_2 are valid kernels, then $K(x, y) = K_1(x, y) * K_2(x, y)$ is a valid kernel. Use this fact and parts (a)-(c) to prove that the polynomial kernel $K(x, y) = (b\langle x, y \rangle + c)^d$ is a valid kernel.

Programming Assignment (60 points)

Introduction

In this assignment, you will be implementing an SVM to solve binary classification problems. While some modern solvers use gradient-based methods to train the SVM, you will be using a Python quadratic programming library, `quadprog` as an optimizer.

Stencil Code & Data

You can find the stencil code for this assignment on the course website. We have provided the following three files:

- `main.py` is the entry point of your program, which will read in the data, run the classifiers and print the results.
- `model.py` contains the `SVM` model that you will be implementing.
- `qp.py` contains a `quadprog` wrapper function, `solve_QP`, which can efficiently solve quadratic programs.

You should *not* modify any code in `main.py` or `qp.py`. All the functions you need to fill in reside in `model.py`, marked by `TODO`s. To run the program, run `python main.py <path_to_dataset>` in a terminal.

Fake Datasets

We've provided two fake datasets, `fake-data1.csv` and `fake-data2.csv`, for you to train your SVM classifier with. Both datasets contains only two dimensional data so that you can easily plot the data if you like. Our main.py converts the labels into $\{-1, 1\}$. The first fake dataset is linearly separable while the second is not. You should use the fake datasets for debugging purposes.

Spambase Dataset

You will also be testing your SVM classifier on a real world dataset, the Spambase dataset. You can find more details on the dataset **here**. We will only be using a subset of the full dataset.

You can find the stencil code for this assignment in the course directory. On a department machine, you can copy the files to your working directory by running the command

```
cp -r /course/cs1420/pub/hw09/* <DEST DIRECTORY>
```

where `<DEST DIRECTORY>` is the directory where you would like to copy the stencil code and data. If you are working locally, you will need to use the `scp` command to copy the files to your computer over `ssh`:

```
scp -r <login>@ssh.cs.brown.edu:/course/cs1420/pub/hw09/* <DEST DIRECTORY>
```

The Assignment

Quadratic Program Review

Quadratic programs are a specific type of optimization problem. Each quadratic program consists of an objective function and a set of constraints. The objective function is the term that is being optimized, and the constraints are inequalities that limit the possible values our decision variables (the variables we are solving for) can take.

The objective function of a quadratic program has the form:

$$\text{Minimize } \frac{1}{2}x^{\mathrm{T}}Qx + c^{\mathrm{T}}x$$

The constraints have the form:

$$Ax \leq b$$

In these equations, Q, c, A, and b are constants that define the problem, and x is the variable that we would like to solve for.

Example

Let's see how formulating the matrices (Q and A) and vectors (c and b) would work in an example. Let $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ be a two-dimensional vector. Let's say we have the following:

$$\begin{aligned} \text{minimize } & 3x_1^2 + 2x_1x_2 + x_2^2 - 6x_1 + 4x_2 \\ \text{subject to: } & 3x_1 - x_2 \geq 5 \\ & 2x_2 \leq 6 \\ & x_2 - x_1 \leq -1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

We can reformat the objective function by defining $c = \begin{pmatrix} -6 \\ 4 \end{pmatrix}$ giving us $-6x_1 + 4x_2 = c^T x$. We can also pull out a $\frac{1}{2}$ from the quadratic terms, leaving us with:

$$\frac{1}{2}(6x_1^2 + 4x_1x_2 + 2x_2^2) + c^T x$$

We can define $Q = \begin{pmatrix} 6 & z \\ 4 - z & 2 \end{pmatrix}$ with $z \in \mathbb{R}$ (z can be any real number) giving us $6x_1^2 + 4x_1x_2 + 2x_2^2 = x^T Qx$. Thus our objective function is now in the form we wish:

$$\frac{1}{2}x^T Qx + c^T x$$

In order to format the inequality constraints, we must first format all constraints to be less than or equal to constraints. Therefore, for any constraint of the form $ax_1 + bx_2 \geq y$, we must flip the inequality by multiplying both sides by -1. The example before would yield $-ax_1 - bx_2 \leq -y$. Thus we perform this on all greater than or equal to constraints, giving us:

$$\begin{aligned} -3x_1 + x_2 &\leq -5 \\ 2x_2 &\leq 6 \\ -x_1 + x_2 &\leq -1 \\ -x_1 &\leq 0 \\ -x_2 &\leq 0 \end{aligned}$$

Each row of A will correspond to the coefficients of x of a constraint while each element of b is the constraint itself. For example, the first row of A could be $\begin{pmatrix} -3 & 1 \end{pmatrix}$ and the first element of b would be -5. We can have $A = \begin{pmatrix} -3 & 1 \\ 0 & 2 \\ -1 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix}$ and $b = \begin{pmatrix} -5 \\ 6 \\ -1 \\ 0 \\ 0 \end{pmatrix}$. Thus our original problem is now of the form required:

$$Ax \leq b$$

Thus our problem has been converted to the form we wanted, with Q, c, A, and b defined so that we achieve that form.

Quadprog

We will be using the Python library `quadprog` to solve quadratic programs. You do not need to call any of the methods provided by `quadprog` since we have written a wrapper function `solve_QP`, located in the `qp.py` file. Given Q, c, A, and b, `solve_QP` will return the values of x that obtain the optimal solution. The example quadratic program from above is also included in `qp.py` and you can try running the file to find the solution.

Before running the code you will need to make sure that `quadprog` is installed on your machine. If you are on a department machine, we have installed `quadprog` in our virtualenv. If you are running locally, you should run `pip3 install quadprog`.

SVMs as Quadratic Programs

In lecture, we showed that we can formulate the soft SVM learning objective with kernels as

$$\min_{\alpha \in \mathbb{R}^m} \lambda \alpha^T G \alpha + \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y_i (G\alpha)_i\}$$

where G is the Gram matrix and λ is a regularization hyperparameter. In order to use `quadprog`, we will equivalently express this objective as

$$\min_{[\alpha, \xi] \in \mathbb{R}^{2m}} \lambda \alpha^T G \alpha + \frac{1}{m} \sum_{i=1}^{m} \xi_i$$

$$\text{such that } \ \forall i \in [m] \quad \xi_i \geq 0 \ \text{ and } \ \xi_i \geq 1 - y_i (G\alpha)_i$$

In doing so, we have added a new variable, ξ, into our equations. However, to make our quadratic program fit the form required by `quadprog`, we need to have just one variable vector. Therefore, we can define our variable vector, x, to be the α vector concatenated with the ξ vector. Recall that `quadprog` expects our quadratic program to fit the following form.

$$\text{Minimize } \frac{1}{2} x^{\mathrm{T}} Q x + c^{\mathrm{T}} x$$

$$\text{such that } Ax \leq b$$

Tips:

- We know that x should be α concatenated with ξ (this should be length $2m$)
- You can turn greater-than inequalities into less-than inequalities by multiplying them by -1, $-Ax \leq -b$
- Remember to set the appropriate values of Q, c, A and b to zero so that the input to `quadprog` is equivalent to the soft SVM objective.

Classification

After solving for the α values, we can classify a given data point x using the following expression:

$$h(x) = \text{sign}\Big(\sum_{i=1}^{m} \alpha_i K(x_i, x)\Big)$$

where the sign function is defined as usual. **Note:** You can assign the output of sign(0) to either -1 or 1.

Kernels

We use the kernel trick to implicitly transform the data to higher dimensions. Two popular kernels for SVMs are the polynomial kernel and the radial basis function kernel. Recall that a kernel function can be used to compute an inner product in high dimensions, so you can replace $\langle \Psi(x_i), \Psi(x_j) \rangle$ with $K(x_i, x_j)$, which takes a different form depending on the kernel used:

- **Linear Kernel** (the same as the dot product):

$$K(x_i, x_j) = x_i^T x_j.$$

- **Polynomial Kernel**:

$$K(x_i, x_j) = (x_i^T x_j + c)^k.$$

- **Radial Basis Function Kernel**:

$$K(x_i, x_j) = \exp\Big(-\gamma ||x_i - x_j||^2\Big).$$

Note that c, k and γ are all *hyperparameters* here. That is, the kernels will behave differently depending on the values you use for these parameters. For example, as d increases, the polynomial kernel can represent more flexible decision boundaries. We provide you with hyperparameter values that perform well, and do not expect you to change them; however, feel free to experiment yourselves.

Project Report

Guiding Questions

- Comment on the testing and training accuracy of your SVM classifier on the Spam dataset. Discuss how kernels (and their hyperparameters) affect the classifier's accuracy. (5 points)
- Read this NY times article about algorithmic bias in law enforcement. What do you think law enforcement organizations can do to limit cases of wrongful accusations caused by biased algorithms? What can the companies that produce these technologies do? Consider some of the solutions hinted at in the article: would increasing the racial diversity in the images fed into the algorithm help? What about banning low-quality images from being used? Think of other potential solutions too. On a more general level, should we even be using such algorithms in law enforcement? (5 points)

- (Extra Credit) Plot the differences in training and testing error as you change the hyperparameters of the polynomial and RBF kernel. (5 points)
- (Extra Credit) Produce a visualization of the decision boundaries on the both of the 2D datasets (`fake-data1.csv`, `fake-data2.csv`). Your plot should contain the training data and decision boundaries. In addition, the label of each training point should be included. This should all be contained within a single figure for each dataset. (5 points)

Grading Breakdown

The following are the testing accuracy targets for this assignment, with random seed and np random seed set to 0, and default hyperparameter values:

	Kernel		
Dataset	Linear	RBF	Polynomial
fake-data1	>65%	>95%	>95%
fake-data2	>43%	>95%	>95%
spambase	>80%	>80%	>80%
spambase_abridged	>80%	>80%	>80%

The grading breakdown for this assignment is as follows:

Written Questions	30%
SVM Classifier	60%
Report	10%
Total	100%

Handing in

Written Assignment

You will turn in your written questions along with your report via Gradescope. If you have questions on how to set up or use Gradescope, ask on Piazza!

Programming Assignment

To hand in the programming component of this assignment, first ensure that your code runs on *Python 3* using our course `virtualenv`. You can activate the `virtualenv` on a department machine by running the following command in a Terminal:

```
source /course/cs1420/cs142_env/bin/activate
```

Once the `virtualenv` is activated, run your program and ensure that there are no errors. We will be using this `virtualenv` to grade all programming assignments in this course so we recommend testing your code on a department machine each time before you hand in. Note that handing in code that does not run may result in a significant loss of credit.

To hand in the coding portion of the assignment, run `cs142_handin hw09` from the directory containing all of your source code.

Anonymous Grading

You need to be graded anonymously, so do not write your name anywhere on your handin.

Obligatory Note on Academic Integrity

Plagiarism — don't do it.

As outlined in the Brown Academic Code, attempting to pass off another's work as your own can result in failing the assignment, failing this course, or even dismissal or expulsion from Brown. More than that, you will be missing out on the goal of your education, which is the cultivation of your own mind, thoughts, and abilities. Please review this course's collaboration policy and if you have any questions, please contact a member of the course staff.