Homework 9

Due: Thursday, April 04, 2024 at 12:00pm **(Noon)**

Written Assignment

Problem 1: Kernels (25 points)

In lecture, we described a kernel $K(\mathbf{u}, \mathbf{v}) = \langle \psi(\mathbf{u}), \psi(\mathbf{v}) \rangle$ as a function that allows us to compute inner products in high dimensions efficiently. However, some functions $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ are not valid kernels for any mapping $\psi : \mathbb{R}^d \to \mathbb{R}^e$. 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* G for a kernel function K and a dataset $\mathbf{u}_1, ..., \mathbf{u}_n \in \mathbb{R}^d$ to be:

$$G := \begin{bmatrix} K(\mathbf{u}_1, \mathbf{u}_1) & K(\mathbf{u}_1, \mathbf{u}_2) & \dots & K(\mathbf{u}_1, \mathbf{u}_n) \\ K(\mathbf{u}_2, \mathbf{u}_1) & K(\mathbf{u}_2, \mathbf{u}_2) & \dots & K(\mathbf{u}_2, \mathbf{u}_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(\mathbf{u}_n, \mathbf{u}_1) & K(\mathbf{u}_n, \mathbf{u}_2) & \dots & K(\mathbf{u}_n, \mathbf{u}_n) \end{bmatrix}$$

K is a valid kernel if and only if G is symmetric and positive semidefinite (PSD) for all possible datasets. In Homework 1, you showed that a matrix M is PSD if and only if $\mathbf{a}^T G \mathbf{a} \geq 0$ for all $\mathbf{a} \in \mathbb{R}^n$. You may find this fact useful in the following, but it is not required.

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

a. Prove that the linear kernel $K(\mathbf{u}, \mathbf{v}) = \langle \mathbf{u}, \mathbf{v} \rangle$ is a valid kernel.

b. Show that, assuming K_1 is a valid kernel and $b > 0$, $K(\mathbf{u}, \mathbf{v}) = bK_1(\mathbf{u}, \mathbf{v})$ is a valid kernel by showing that the Gram matrix of K is PSD.

c. Prove that if K_1 is a valid kernel and $c \geq 0$, then $K(\mathbf{u}, \mathbf{v}) = K_1(\mathbf{u}, \mathbf{v}) + c$ is a valid kernel.

   Hint: Let $\psi_1(\mathbf{u})$ be the mapping for K_1. How could you use $\psi_1(\mathbf{u})$ to construct another mapping, $\psi(\mathbf{u})$, for which K is a valid kernel?

d. Using the fact that, assuming K_1 and K_2 are valid kernels, $K_1(\mathbf{u}, \mathbf{v}) \cdot K_2(\mathbf{u}, \mathbf{v})$ is a valid kernel and parts (a)-(c), show that the polynomial kernel $K(\mathbf{u}, \mathbf{v}) = (b\langle \mathbf{u}, \mathbf{v} \rangle + c)^d$ is a valid kernel.

Problem 2: Deriving a Kernel Function (5 points)

For the programming portion, our goal will be to classify data points $\mathbf{x} \in \mathbb{R}^2$. Unfortunately, the training set we have may not always be linearly separable in its original feature space $\mathbb{R}^2$, so we will use the following embedding function, $\psi : \mathbb{R}^2 \to \mathbb{R}^3$.

$$\psi(\mathbf{x}) = (x_1, x_2, x_1^2 + x_2^2)$$

However, to avoid embedding every point manually, we also would like to have a kernel representation of this embedding.

Prove that the following equality holds.

$$K(\mathbf{x}', \mathbf{x}) = \langle \mathbf{x}', \mathbf{x} \rangle + \|\mathbf{x}'\|_2^2 \|\mathbf{x}\|_2^2 = \langle \psi(\mathbf{x}'), \psi(\mathbf{x}) \rangle$$

This implies that $K : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ is a kernel representation of ψ.

Programming Assignment (30 points)

Background

Recall that the soft SVM optimization objective is given by

$$\operatorname*{argmin}_{\mathbf{w},\xi} \lambda\|\mathbf{w}\|^2 + \frac{1}{m}\sum_{i=1}^{m}\xi_i \tag{1}$$

$$\text{s.t. } \forall i \in [m],\ y_i\langle \mathbf{w}, \mathbf{x}_i\rangle \geq 1-\xi_i,\ \xi_i \geq 0,$$

where $\mathbf{x}_i$ is a training example and $\mathbf{w}$ is a vector normal to the hyperplane. In lecture, we saw that the optimal solution, $\mathbf{w}^*$ can be expressed as a sum of coefficients and training examples:

$$\mathbf{w}^* = \sum_{i=1}^{m}\alpha_i\mathbf{x}_i \tag{2}$$

The classifier $h(x)$ can be expressed as

$$h(\mathbf{x}) = \operatorname{sign}(\langle \mathbf{w}, \mathbf{x}\rangle) = \operatorname{sign}\left(\sum_{i=1}^{m}\alpha_i\langle \mathbf{x}_i, \mathbf{x}\rangle\right). \tag{3}$$

We also saw that, in cases where the data is not linearly separable, it may be favorable to embed the data into a higher dimensional feature space. For an embedding function $\psi(\mathbf{x})$, equation (2) and (3) become

$$\mathbf{w}^* = \sum_{i=1}^{m}\alpha_i\psi(\mathbf{x}_i), \qquad h(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{m}\alpha_i\langle \psi(\mathbf{x}_i), \psi(\mathbf{x})\rangle\right). \tag{4}$$

Assuming there exists a valid kernel function, K, for $\psi(x)$, by definition, equation (5) becomes

$$h(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{m}\alpha_i K(\mathbf{x}_i, \mathbf{x})\right). \tag{5}$$

So, the corresponding hyperplane is given by

$$H(\mathbf{x}) = \sum_{i=1}^{m}\alpha_i K(\mathbf{x}_i, \mathbf{x}) = 0. \tag{6}$$

Introduction

You can find and download the stencil code here: `HW9 on Github`.

All four parts of the coding portion of this assignment will be related to producing visualizations which you will be asked to reflect upon. We will be using `sklearn`'s implementation of a support vector machine classifier. You can find documentation related to svm.SVC here. You can find documentation about overall implementation details of `sklearn`'s SVM library here.

`sklearn`'s SVM implementation and treatment of the optimization object use an explicit intercept value. Thus equation (6) and (7) would be implemented as

$$h(\mathbf{x}) = \text{sign}(H(\mathbf{x})) = \text{sign}\left(\sum_{i \in SV} \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b\right) \tag{7}$$

$$H(\mathbf{x}) = \sum_{i \in SV} \alpha_i K(\mathbf{x}_i, \mathbf{x}) + b \tag{8}$$

where b is some scalar representing the intecrcept or bias, and SV is the set of indices of support vectors. Both of these are discovered by the optimization algorithm used by `sklearn` after calling `svm.fit()`.

You will need to modify the following files:

- `decision_surface_visualization.py`: Most of the implementation logic for getting the data to plot in part 1, 2.
- `support_vector_visualization.py`: All of the plotting and data generation for part 4 is done here.
- `main.py`: Here is where you'll call the functions you filled out in `decision_surface_visualization.py` and `support_vector_visualization.py`

You will *not* need to modify `utils.py`; however, we strongly recommend you take a look at it to understand some of the plotting logic and what the parameters do.

Before you start

Since this assignment relies on external libraries, some of the features used in the stencil require certain versions or above of these libraries. Please ensure that the installed libaries in the version of Python you're using is at least of versions:

- sklearn: 1.0.2
- numpy: 1.22.2
- matplotlib: 3.5.1

Finally we highly encourage you to read the SVC documentation and become familiar with the methods and attributes available as you will be using some of them in this assignment. (Don't worry about implementation details, just be aware they exist)

Part 1

First, you'll be visualizing the decision boundary from $H(\mathbf{x})$ in both the higher dimensional feature space and the original feature space.

Visualizing The Embedding Space

As mentioned in problem 2 of the written portion, we'll use a simple embedding, $\phi((x_1, x_2)) \rightarrow (x_1, x_2, x_1^2+x_2^2)$, called the *polar embedding*. The data provided is not linearly separable in the original space it's given in; however, it is linearly separable in this embedding space. To do so, we manually calculate the higher dimensional feature space coordinates. We then train an SVM using a linear kernel using those manually calculated coordinates.

This has the same effect as training on the original data and using a Kernel representation We will visually confirm this in Part 2.

TODO: First, implement the polar embedding function and it's kernel representation. Functions you'll need to modify:

- `polar_embedding()` in `decision_boundary_visualization.py`
- `polar_kernel()` in `decision_boundary_visualization.py`

To plot our hyperplane, imagine a standard xyz Cartesian coordinate space, where x, y, and z are each their own axis. Let our new feature values from the embedding, $x_1^2 + x_2^2$, be the z-axis, x_1 be the x-axis and x_2 be the y-axis. We would like to find what $x_1^2 + x_2^2$ values on our hyperplane correspond to each pair x_1 and x_2. We can do this with some simple elementary algebra

$$H(\mathbf{x}) = w_1x_1 + w_2x_2 + w_3(x_1^2 + x_2^2) + b = 0 \tag{9}$$

$$w_1x_1 + w_2x_2 + b = -w_3(x_1^2 + x_2^2) \tag{10}$$

$$-\frac{w_1}{w_3}x_1 - \frac{w_2}{w_3}x_2 - \frac{b}{w_3} = x_1^2 + x_2^2 \tag{11}$$

Once we have fit our SVM, `sklearn` provides us with $\mathbf{w}^*$ and b^*, so we have everything we need to implement (12). Now, generate an image of

- The decision boundary in the point's original space, along with the original points.
- The decision boundary in the higher dimensional feature space, along with the embeddings of the points.

This will be done by implementing (12), and calculating the new feature values, $x_1^2 + x_2^2$, over a grid of points (x_1, x_2).

TODO: The functions you'll need to modify for this part (listed in the recommended order of implementation) are

1. `generate_new_feature_values()` in `decision_boundary_visualization.py`
2. `visualize_decision_boundary_embedded_space()` in `main.py`

Part 2

Now, we'll visualize $H(\mathbf{x})$ in its own space. This time, instead of letting the "z"-axis represent the feature $x_1^2 + x_2^2$, we will let it represent values of $H(\mathbf{x})$. We would also like to visually confirm that there's no difference in the solution found (when all the samples are linearly separable in embedding space) when training a linear SVM in embedding space vs training an SVM using the kernel trick which corresponds to the same embedding space.

This process should feel similar to part (1). Implement equation (9), then calculate the value of $H(\mathbf{x})$ over a grid.

TODO: The functions you'll need to modify for this part (listed in the recommended order of implementation) are

1. The rest of the TODO's in `decision_boundary_visualization.py`
2. `visualize_H()` in `main.py`

Part 3

In this part you will be observing the effect of the regularization λ on the decision surface found by the SVM. Without changing anything else in our optimization objective we can express (1) as

$$\underset{\mathbf{w},\xi}{\operatorname{argmin}} \|\mathbf{w}\|^2 + \frac{1}{\lambda}\sum_{i=1}^{m} \xi_i \tag{12}$$

The constant coefficients will differ from representation to representation, but regardless the effect of λ as a regularization parameter stays the same. Rather than λ, `sklearn` uses C to denote the regularization parameter with the following formulation:

$$\underset{\mathbf{w},\xi}{\operatorname{argmin}} \|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \xi_i \tag{13}$$

Bear in mind that C is inversely proportional to λ.

To observe this we will plot the decision surface in $\mathbb{R}^2$, using an SVM with the RBF kerel, or radial basis function.

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\gamma\|\mathbf{u} - \mathbf{v}\|^2\right)$$

where γ is a tunable parameter. Note that if we express γ as $\gamma = \frac{1}{2\sigma^2}$, we get the Gaussian kernel similar to how it's presented in lecture. The RBF kernel is an extremely popular choice as it provides a lot of representational power, in fact, it provides an infinite dimensional embedding. See this resource from the University of Wisconsin - Madison for more info. For this assignment we will use the default value of γ used by `sklearn`.

At this point, all the code you need outside of `visualize_slack_penalty()` should be completed from previous parts. Your job here is to just fill in the rest of the function and observe the difference between each plot.

TODO: Fill out `visualize_slack_penalty()` in `main.py`

Part 4

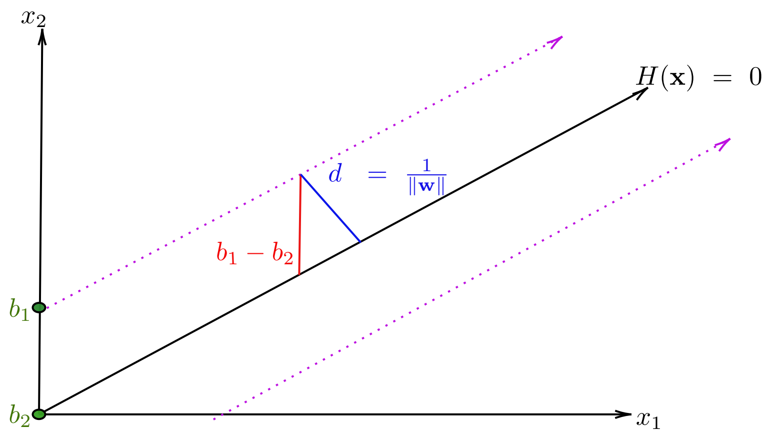
In this part we will pay attention to the support vectors themselves. Using some linearly separable data in $\mathbb{R}^2$, and a large C value to mimic a hard margin SVM, we'll visualize which points get picked as support vectors, and verify that they lie exactly on the margins.

Recall that in equation (9), for all x_i which are not support vectors, $\alpha_i = 0$, which means that non support vector points do not contribute to the decision boundary at all. We show this by first training with all points, then training with only support vectors

We would also like to see how the support vectors themselves directly effect the margin. To do this, we iterate over a number of times, and each iteration we remove support vectors from the data set.

Plotting the Margins

Plotting the decision boundary is a similar process as part (1), but with one less feature. Let the "y"-axis be x_2 and get x_2 in terms of x_1, w_1, w_2, b. Then calculate x_2 for a range of values of x_1. Plotting the margins is a bit more involved. One approach is to use the points of the decision boundary we've already calculated then just add or subtract the "vertical" distance between the decision boundary and the margins in $\mathbb{R}^2$.



Luckily, as discussed in lecture, the margins are parallel to the decision boundary. The distance between parallel lines at $x_1 = 0$ is well known to be

$$d = \frac{b_1 - b_2}{\sqrt{1+s^2}} \tag{14}$$

Where s is the slope both lines share, and b_1, b_2 are the intercepts, $b_2 - b_1$ is the vertical distance. We also know the distance between the decision boundary and a point $\mathbf{a}$ on the margin is $\frac{|\langle \mathbf{w}, \mathbf{a} \rangle|}{\|\mathbf{w}\|}$, since the margins are the points where $\langle \mathbf{w}, \mathbf{x} \rangle = \pm 1$, the distance between the decision boundary and either margin is $\frac{1}{\|\mathbf{w}\|}$.

$$\frac{1}{\|\mathbf{w}\|} = \frac{b_1 - b_2}{\sqrt{1+s^2}}$$

$$\text{vertical distance} = b_1 - b_2 = \frac{\sqrt{1+s^2}}{\|\mathbf{w}\|} \tag{15}$$

TODO: Functions you'll need to modify for this part

1. All of `support_vector_visualization.py`
2. `visualize_support_vectors()` in `main.py`

Project Report (40 points)

1. Add all the plots generated as part of this assignment. **This is how we will be primarily grading correctness of your code.** For the 3d plots, just take a screen shot from any angle.

2. In part 1, we saw that the embedding, $\phi((x_1, x_2)) \to (x_1, x_2, x_1^2 + x_2^2)$, was able to make our non linearly separable data in $\mathbb{R}^2$ perfectly linearly separable in the higher dimensional feature space. Give a non rigorous explanation using the embedding function and the nature of the data regarding why the polar embedding was able to make *this particular dataset* perfectly linearly separable. Would this embedding be able to make any data set linearly separable? Feel free to use the plots generated in part 1 to support your answer

3. In part 2, we visually confirmed the solution found by the SVM by manually embedding and training a linear SVM vs training a SVM using a corresponding kernel were the same. In lecture we mathematically proved that they are the same. If this is the case, then why would we use a kernel function rather than manually embedding? Give two reasons why this is the case.

4. In part 3, what trend do you see regarding the complexity of the decision boundaries as C increases? When would we want to use low C values or high C values?

5. In part 4, does the plot of the SVM trained on the entire dataset vs the plot of the SVM trained on only the support vectors match your expectations? Explain why you had those expectations. It's fine if the plots contradicted your expectations.

6. In part 4, comment on the margins as we remove support vectors. Explain why this is the case using the optimization problem presented in lecture, and the definition of support vectors.

Grading Breakdown

Note that the coding portion score is based on your submission including your graphs. Otherwise you will at best get partial credit from manually grading your code. The grading breakdown for this assignment is as follows:

Written Questions	30%
Coding Portion	30%
Report	40%
Total	100%

Handing in

You will hand in both the written assignment and the coding portion on gradescope, separately.

1. Your written assignment should be uploaded to gradescope under "Homework 9".

2. Submit your hw9 github repo containing all your source code and your project report named **report.pdf** on gradescope under "Homework 9 Code". **report.pdf** should live in the root directory of your code folder; the autograder will check for the existence of this file and inform you if it is not found. For questions, please consult the download/submission guide.

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.