# DSC 140A - Homework 04

Due: Wednesday, May 1

**Instructions:** Write your solutions to the following problems either by typing them or handwriting them on another piece of paper. Show your work or provide justification unless otherwise noted. If you write code to solve a problem, include the code by copy/pasting or as a screenshot. You may use `numpy`, `matplotlib` (or another plotting library), and any standard library module, but no other third-party libraries unless specified. Submit homeworks via Gradescope by 11:59 PM.

**Problem 1.**

In lecture, we saw that the Soft-SVM optimization problem can be framed as the problem of finding the parameter vector $\vec{w}$ that minimizes the *regularized* empirical risk with respect to the hinge loss:

$$R_{\text{svm}}(\vec{w}) = \|\vec{w}\|^2 + C\sum_{i=1}^{n} \max\left(0, 1 - y_i \vec{w} \cdot \text{Aug}(\vec{x}^{(i)})\right).$$

It can be shown that a subgradient of the empirical risk is given by

$$\text{subgrad}\, R_{\text{svm}}(\vec{w}) = 2\vec{w} + C\sum_{i=1}^{n} \begin{cases} -y_i \,\text{Aug}(\vec{x}^{(i)}) & \text{if } y_i \vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) < 1, \\ 0 & \text{otherwise} \end{cases}$$

The file below contains data suitable for a binary classification problem.

`https://f000.backblazeb2.com/file/jeldridge-data/003-two_clusters/data.csv`

The file contains three columns: $x_1$, $x_2$, and $y$. The first two columns are the features, and the third column is the label.

Using subgradient descent, train a Soft-SVM model on this data using your choice of **either**:

- $C = 10$. This will earn full credit, as long as subgradient appears to converge to (close to) the optimal solution.
- $C = 1000$. This will earn one point of extra credit, as long as subgradient appears to converge to (close to) the optimal solution. While still convex, this optimization problem is much harder because the contours of the objective function are much more elongated. You'll have to fight with the learning rate and the stopping criterion to get it to converge.

You should turn in the following:

- The final parameter vector $\vec{w}$.
- A plot of the data, showing each class as a different color, as well as the learned decision boundary and the lines where $H = 1$ and $H = -1$, respectively.

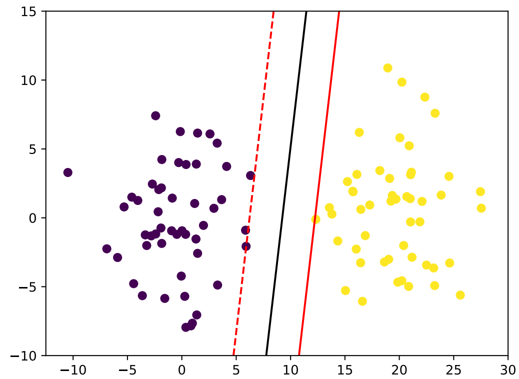
*Tips*: you may find it difficult to get the subgradient descent algorithm to converge if you use a stringent stopping criterion. Instead, you may want to use a smaller threshold and/or a fixed number of iterations and increase it if the plotted decision boundary does not look correct. Remember, you can plot the decision boundary and the lines where $H = 1$ and $H = -1$ by either solving for $x_2$ in terms of $x_1$ or by using matplotlib's `contour` function.

**Solution:**

First, we'll try with $C = 10$:

```python
import numpy as np
import matplotlib.pyplot as plt

data = np.loadtxt("data.csv", delimiter=",")

X = data[:, :-1]
X = np.column_stack((np.ones(X.shape[0]), X))
y = data[:, -1]

C = 10


def svm_loss(w, x, y):
    return np.maximum(0, 1 - y * x @ w)


def svm_risk(w):
    return np.mean([svm_loss(w, x, y) for (x, y) in zip(X, y)])


def subgradient_of_loss(w, x, y):
    h = x @ w
    if y * h <= 1:
        return -y * x
    else:
        return np.zeros_like(w)


def subgradient_of_risk(w):
    return (
        C * np.sum([subgradient_of_loss(w, x, y) for (x, y) in zip(X, y)], axis=0)
        + 2 * w
    )


def gradient_descent(gradient, z_0, initial_learning_rate, stop_threshold):
    z = z_0
    t = 1
    while True:
        learning_rate = initial_learning_rate / np.sqrt(t)
        t += 1
        z_new = z - learning_rate * gradient(z)
        print(t, z_new, np.linalg.norm(z_new))
        if np.linalg.norm(z_new - z) < stop_threshold:
            break
        z = z_new
    return z_new


w_opt = gradient_descent(subgradient_of_risk, np.array([-0.1, 0.01, 0.01]), 0.01, 1e-3)
print(w_opt)
```

Notice that we've set the stopping threshold to be rather small, but we still converge to what we expect:



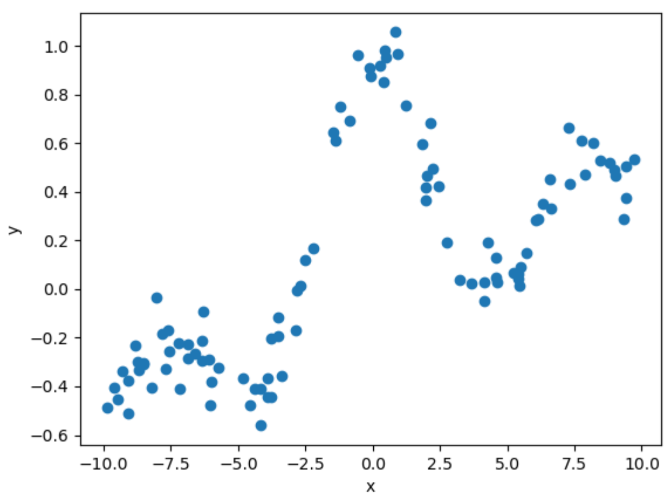

Using $C = 1{,}000$ is considerably harder, but we can still get it to converge. The problem is that the contours of the objective function are much more elongated, meaning that the learning rate and stopping criterion are much more sensitive. One approach is use a check of $\|\vec{w}\|$ as a stopping criterion, as we know that the optimal solution will have small $\|\vec{w}\|$. Some trial and error shows that stopping when $\|\vec{w}\| < 3.5$ works well.

**Problem 2.**

The data set linked below contains data for performing non-linear regression. The first column is $x$ (the independent variable), and the second column is $y$ (the dependent variable).

`https://f000.backblazeb2.com/file/jeldridge-data/010-nonlinear_regression/data.csv`

Plotting the data shows that there is a non-linear relationship between $x$ and $y$:




**a)** Consider the function:

$$H(x) = w_0 + w_1\phi_1(x) + w_2\phi_2(x) + w_3\phi_3(x) + w_4\phi_4(x) + w_5\phi_5(x),$$

where each $\phi_i(x)$ is a Gaussian basis function:

$$\phi_i(x) = \exp\left(-\frac{(x - \mu_i)^2}{\sigma^2}\right).$$

Assume that the basis functions are equally spaced, with $\mu_1 = -10, \mu_2 = -5, \mu_3 = 0, \mu_4 = 5, \mu_5 = 10$. Also assume that all of the basis functions have the same width parameter, $\sigma$.

Write a Python function `plot_h(w_0, w_1, w_2, w_3, w_4, w_5, sigma)` that takes in $w_0, \ldots, w_5$ and $\sigma$ and plots the function $H(x)$ on top of the data.

Using this function, guess values for $w_0, \ldots, w_5$ and $\sigma$ that make the plot of $H(x)$ look like it fits the data well. Provide your code, the values you guessed, and the plot of $H(x)$ that results from your guesses.

*Note:* your function doesn't need to fit the data perfectly, but should be reasonably close. There will necessarily be places where $H$ does not fit the data well.

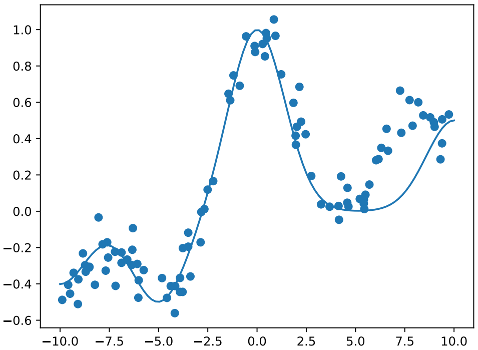
**Solution:**

```
import numpy as np
import matplotlib.pyplot as plt


data = np.loadtxt("data.csv", delimiter=",")
x, y = data.T


def gaussian(x, mu, sigma=2):
    return np.exp(-(((x - mu) / sigma) ** 2))


def h(x, w_0, w_1, w_2, w_3, w_4, w_5, sigma):
    return (
        w_0
        + w_1 * gaussian(x, -10, sigma)
        + w_2 * gaussian(x, -5, sigma)
        + w_3 * gaussian(x, 0, sigma)
        + w_4 * gaussian(x, 5, sigma)
        + w_5 * gaussian(x, 10, sigma)
    )


def plot_h(w_0, w_1, w_2, w_3, w_4, w_5, sigma=1):
    x = np.linspace(-10, 10, 100)
    y = h(x, w_0, w_1, w_2, w_3, w_4, w_5, sigma)
    plt.plot(x, y)
```

Using $w_0 = 0$, $w_1 = -0.4$, $w_2 = -0.5$, $w_3 = 1$, $w_4 = 0$, $w_5 = 0.5$, and $\sigma = 2$, and running:

```
plot_h(0, -0.4, -0.5, 1, 0, 0.5, sigma=2)
```

We get the following plot:




This seems to fit the data reasonably well. Notice that $H$ does not fit the data perfectly around $x = 7.5$, but this is to be expected since there isn't a basis function nearby. We can only precisely control the plot near the basis functions.

**b)** Write a function `phi(x)` which takes in a scalar $x$ in the input space and maps it to a vector in the feature space using the Gaussian basis functions above. The function should return a vector of length 5, where the $i$th feature is given by $\phi_i(x)$. Your function should use $\sigma = 2$ for the Gaussian width, but the same locations as in the previous part.

For this part, provide 1) your code, and 2) the output of `phi(3)`.

**Solution:**

```
def phi_x(x):
    return np.array(
        [
            1,
            gaussian(x, -10, 2),
            gaussian(x, -5, 2),
            gaussian(x, 0, 2),
            gaussian(x, 5, 2),
            gaussian(x, 10, 2),
        ]
    )
```

Running `phi(3)` gives:

```
[1.00000e+00 4.47773e-19 1.12535e-07 1.05399e-01 3.67879e-01 4.78511e-06]
```

**c)** Fit a linear regression model in feature space by minimizing mean squared error, again using a Gaussian width parameter of $\sigma = 2$. What is the learned $\vec{w}$? Show your work.

**Solution:**

```
# map data to feature space
Phi = np.array([phi_x(xi) for xi in x])

# solve for w
w = np.linalg.solve(Phi.T @ Phi, Phi.T @ y)
```

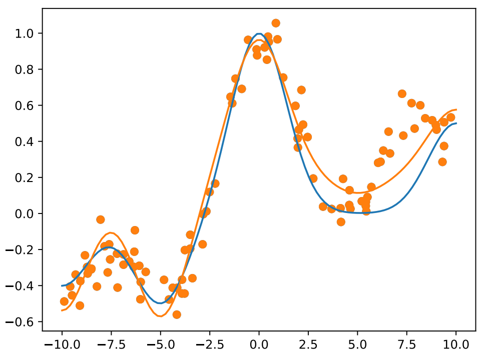
The learned $\vec{w}$ is:

```
[ 0.21682776 -0.75298839 -0.78825379 0.74848684 -0.10506351 0.35905595]
```

**d)** Using the learned $\vec{w}$, plot the prediction function $H(x)$ on top of the data.

*Hint:* Can you reuse the `plot_h` function from part (a)?

**Solution:** Plug in the learned $\vec{w}$ into the `plot_h` function, we get the following plot:




The orange line is the function fit by minimizing mean squared error, whereas the blue line is the function we guessed in part (a).