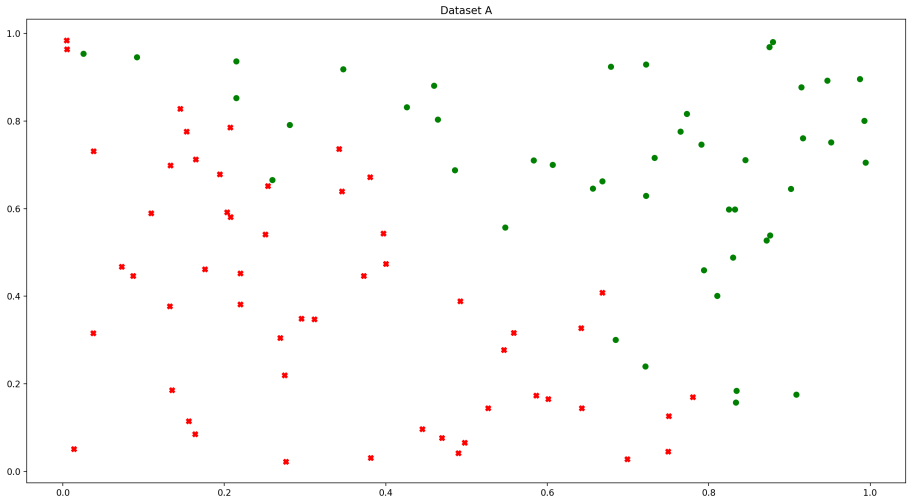
CS 229 Autumn 2018 Problem Set #2

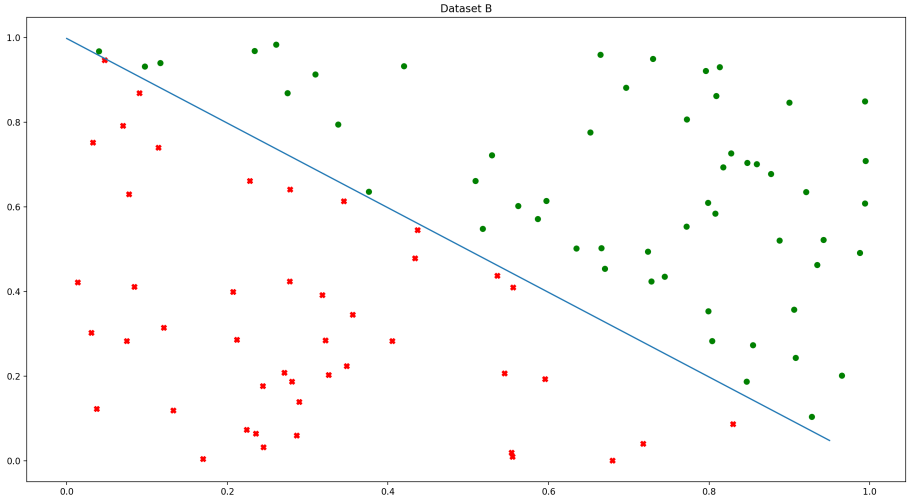
Ruining Li
University of Oxford

Problem 1

(a) Training the logistic regression model converges in about 30000 iterations on dataset A but fails to converge on dataset B in a reasonable amount of time.

(b) We plot the data for the 2 datasets respectively:





Note that dataset B is linearly separable while dataset A isn't. When fitting a logistic regression model to a linearly separable dataset, suppose when optimizing the loss function we find a parameter θ which correctly classifies all training examples, then any parameter $\alpha\theta$ (where the scalar $\alpha > 1$) achieves a smaller loss, as shown below:

$$\begin{aligned} J(\theta) &= -\frac{1}{m}\sum_{i=1}^{m}\log(h_\theta(y^{(i)}x^{(i)})) \\ &> -\frac{1}{m}\sum_{i=1}^{m}\log(h_{\alpha\theta}(y^{(i)}x^{(i)})) = J(\alpha\theta) \end{aligned}$$

where the loss function is different from the one in Problem 1 of Problem Set 1 because here $y^{(i)} \in \{1, -1\}$, rather than $\{0, 1\}$. Note that the inequality comes from the facts that logarithm and sigmoid functions are monotonically increasing and $\theta^T y^{(i)} x^{(i)} > 0$ because of the correct classification of $x^{(i)}$.

Therefore, the loss function has no global minimum (i.e., we can reduce the loss function to an arbitrarily small positive number). The gradient of the loss function, which is given by

$$-\frac{1}{m}\sum_{i=0}^{m}\left\{1 - h_\theta(y^{(i)}x^{(i)})\right\} y^{(i)}x^{(i)}$$

does indeed become closer to 0 over time (as $h_\theta(y^{(i)}x^{(i)})$ gets closer to 1), but the decrease is too slow for the gradient descent optimization to converge in a reasonable amount of time. If we make the condition for convergence looser (e.g., `np.linalg.norm( prev_theta - theta) < 1e-4`), training the model converges in about 1680000 iterations on dataset B.

(c) Note that the provided training algorithm, even without any modification, will eventually converge (based on the code that comes along with this problem). But it doesn't converge in a **reasonable** amount of time.

(i) It will not lead to the provided training algorithm converging on datasets such as B (in a reasonable number of iterations).

Using a larger learning rate will only make things worse, as the gradient needs to be even smaller to satisfy the convergence condition. In order to satisfy the convergence condition with a large gradient, we need to choose a learning rate that is significantly smaller than the current one, in which case gradient descent makes progress much slower.

(ii) Theoretically decreasing the learning rate over time can help accelerate convergence, as a smaller learning rate allows gradients farther away from 0 to satisfy the convergence condition. In addition, gradient descent still makes decent progress initially.

However, in practice it is very hard to find a specific way to decrease the learning rate so that the resulting training algorithm converges in a reasonable amount of time on datasets such as B.

(iii) It will not lead to the provided training algorithm converging on datasets such as B (in a reasonable number of iterations).

Linear scaling of the input features will not change the linear separability of the dataset. Therefore, the same problem persists.

(iv) By far this is the modification that is most likely to lead to the provided training algorithm converging on datasets such as B (in a reasonable number of iterations).

Adding a regularization term $\|\theta\|_2^2$ to the loss function would maintain the convexity of the loss function. In addition, the regularized loss function now has a global minimum, which gradient descent approaches iteratively. Note whether or not the optimization eventually converges also depends on a well-tuned learning rate.

(v) Whether or not it would lead to the provided training algorithm converging on datasets such as B depends on whether or not the datasets (with noise) are linearly separable. If they are linearly non-separable, the training algorithm will converge.

(d) SVMs are not vulnerable to datasets like B. Recall that the objective of SVMs is:

$$\begin{aligned} \min_{w} \quad & \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m \\ & \xi_i \geq 0, \quad i = 1, \ldots, m \end{aligned}$$

This constrains our freedom to arbitrarily scale w and b.

Problem 2

(a) $h_\theta(x^{(i)}) \in (0, 1)$ for all i. Therefore, the property holds true if and only if

$$\sum_{i=1}^{m} h_\theta(x^{(i)}) = \sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\}$$

Note that after training the logistic regression model produces θ which satisfies:

$$\sum_{i=1}^{m}(y^{(i)} - h_\theta(x^{(i)}))x^{(i)} = 0.$$

We complete the proof by stating that

$$\text{L.H.S} = \sum_{i=1}^{m} h_\theta(x^{(i)})x_0^{(i)} = \sum_{i=1}^{m} y^{(i)}x_0^{(i)} = \text{R.H.S}$$

where we used the fact that $x_0^{(i)} = 1$ for all i.

(b) The wording of this part is ambiguous, as we don't know whether **perfect accuracy** refers to an accuracy of 100% or the best accuracy that any binary classification model can achieve. Under either assumption, neither direction is necessarily true.

(c) After training the logistic regression model with L_2 regularization, the model parameter θ satisfies:

$$\sum_{i=1}^{m}(y^{(i)} - h_\theta(x^{(i)}))x^{(i)} = \lambda\theta.$$

Therefore, the model is only well-calibrated if $\theta_0 = 0$.

Problem 3

(a) By the Bayes' theorem,

$$p(\theta|x, y) \propto p(y|x, \theta)p(\theta|x)$$

. If we assume that $p(\theta) = p(\theta|x)$, then $\theta_{\text{MAP}} = \text{argmax}_\theta\, p(y|x, \theta)p(\theta)$ follows naturally.

(b) With a zero-mean Gaussian prior over θ, we have

$$p(y|x, \theta)p(\theta) = p(y|x, \theta)\frac{1}{(2\pi\eta^2)^{n/2}}\exp\left\{-\frac{\theta^T\theta}{2\eta^2}\right\}$$

where $\theta \in \mathbb{R}^n$.

Maximizing the above equation is equivalent to minimizing the negative log of it, i.e., we want to minimize

$$-\log p(y|x, \theta) + \frac{\|\theta\|_2^2}{2\eta^2} + \text{const}$$

which is equivalent to applying L_2 regularization with MLE estimation with the value $\lambda = \frac{1}{2\eta^2}$.

(c) With $\epsilon \sim \mathcal{N}(0, \sigma^2)$, we have

$$p(\vec{y}|X, \theta) = \prod_{i=1}^{m}\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left\{-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right\}.$$

Therefore,

$$-\log\{p(\vec{y}|X, \theta)p(\theta)\} = -\log p(\vec{y}|X, \theta) - \log p(\theta) = \left\{\sum_{i=1}^{m}\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right\} + \frac{\|\theta\|_2^2}{2\eta^2} + \text{const}$$

which we want to minimize.

Differentiate the above equation w.r.t θ and set the gradient to 0, we obtain

$$\frac{X^T(X\theta_{\text{MAP}} - \vec{y})}{\sigma^2} + \frac{\theta_{\text{MAP}}}{\eta^2} = 0$$

which gives the closed form expression for θ_{MAP} as

$$\theta_{\text{MAP}} = (X^T X + \frac{\sigma^2}{\eta^2} I)^{-1} X^T \vec{y}.$$

(d) With a Laplace prior,

$$p(\theta) = \frac{1}{2b} \exp\left(-\frac{\|\theta\|_1}{b}\right)$$

and

$$-\log\{p(\vec{y}|X,\theta)p(\theta)\} = \left\{\sum_{i=1}^{m} \frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right\} + \frac{\|\theta\|_1}{b} + \text{const}$$

which we want to minimize.

The solution is therefore the same as the solution of linear regression with L_1 regularization with $\gamma = \frac{2\sigma^2}{b}$.

Problem 4

Suppose $K_1(x,z) = \mu(x)^T\mu(z)$ and $K_2(x,z) = \eta(x)^T\eta(z)$ for some feature maps $\mu : \mathbb{R}^n \to \mathbb{R}^s$ and $\eta : \mathbb{R}^n \to \mathbb{R}^t$.

(a) K is a kernel. Given n vectors $x_1, x_2, \ldots, x_n$, the corresponding kernel matrix of K is the sum of the corresponding kernel matrix of K_1 and K_2. The sum of positive semi-definite matrices is also positive semi-definite, which indicates that K is a valid kernel.

(b) K is not necessarily a kernel. Let $K_1(x,z) = 0$ for all x and z (which is a valid kernel as the (kernel) matrix with all 0 entries is indeed positive semi-definite). Then, $K(x,z) = -K_2(x,z)$. Given n vectors, the corresponding kernel matrix of K is the negative of the the corresponding kernel matrix of K_2. Negating a positive semi-definite matrix produces a negative semi-definite matrix, which indicates that K is not a valid kernel.

(c) K is a kernel. Similar to part (a), the kernel matrix of K is a times the kernel matrix of K_1. Multiplying a positive semi-definite matrix by a positive number produces another positive semi-definite matrix, which indicates that K is a valid kernel.

(d) K is not a kernel. The kernel matrix of K is $(-a)$ times the kernel matrix of K_1. Multiplying a positive semi-definite matrix by a negative number produces a negative semi-definite matrix, which indicates that K is not a valid kernel.

(e) K is a kernel.

$$\begin{aligned} K(x,z) &= \left(\sum_i \mu_i(x)\mu_i(z)\right)\left(\sum_j \eta_j(x)\eta_j(z)\right) \\ &= \sum_i \sum_j \{\mu_i(x)\eta_j(x)\}\{\mu_i(z)\eta_j(z)\} \end{aligned}$$

which indicates that K is a valid kernel with feature map

$$\phi(x) = [\mu_1(x)\eta_1(x), \mu_1(x)\eta_2(x), \ldots, \mu_1(x)\eta_t(x), \ldots, \mu_s(x)\eta_t(x)]^T$$

(f) K is a kernel. The result simply comes from $K(x,z) = \phi(x)^T\phi(z)$ where $\phi(x) = [f(x)]$.

(g) K is a kernel. The kernel matrix of K corresponding to n given vectors $x_1, x_2, \ldots, x_n$ is just the kernel matrix of K_3 corresponding to n vectors $\phi(x_1), \phi(x_2), \ldots, \phi(x_n)$, which, because K_3 is a kernel, is positive semi-definite. It follows that K is a valid kernel.

(h) K is a kernel. The result follows from part (a), (c) and (e), as p has positive coefficients.

Problem 5

(a) As $\theta^{(0)}$ is initialized to $\vec{0}$, and the update rule is

$$\theta^{(i+1)} := \theta^{(i)} + \alpha(y^{(i+1)} - h_\theta(x^{(i+1)}))\phi(x^{(i+1)}),$$

$\theta^{(i)}$ is a linear combination of $\phi(x^{(1)}), \ldots, \phi(x^{(i)})$. Suppose there are m training examples in total, then for any i,

$$\theta^{(i)} = \sum_{j=1}^{m} \beta_j \phi(x^{(j)})$$

for some β_j's. Note for $\theta^{(i)}$, all β_j's where $j > i$ are 0.

We therefore (implicitly) represent the high-dimensional parameter vector $\theta^{(i)}$ by its corresponding β_j's, as a vector in $\mathbb{R}^m$. The initial value $\theta^{(0)} = 0$ is represented by $\vec{0}$.

To make a prediction on a new input $x^{(i+1)}$, we compute $h_{\theta^{(i)}}(x^{(i+1)})$:

$$h_{\theta^{(i)}}(x^{(i+1)}) = g({\theta^{(i)}}^T\phi(x^{(i+1)})) = \sum_{j=1}^{m} \beta_j \phi(x^{(j)})^T\phi(x^{(i+1)}) = \sum_{j=1}^{m} \beta_j K(x^{(j)}, x^{(i+1)})$$

where the right-hand side can be computed efficiently.

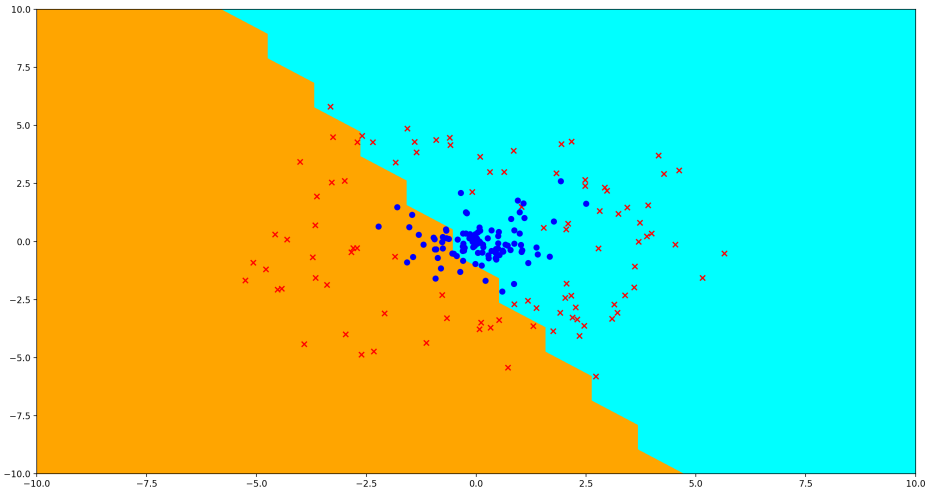
Suppose $\vec{\beta}^{(i)}$ is the vector representing $\theta^{(i)}$, then the update rule becomes

$$\vec{\beta}_{i+1}^{(i+1)} := \vec{\beta}_{i+1}^{(i)} + \alpha(y^{(i+1)} - h_\theta(x^{(i+1)}))$$

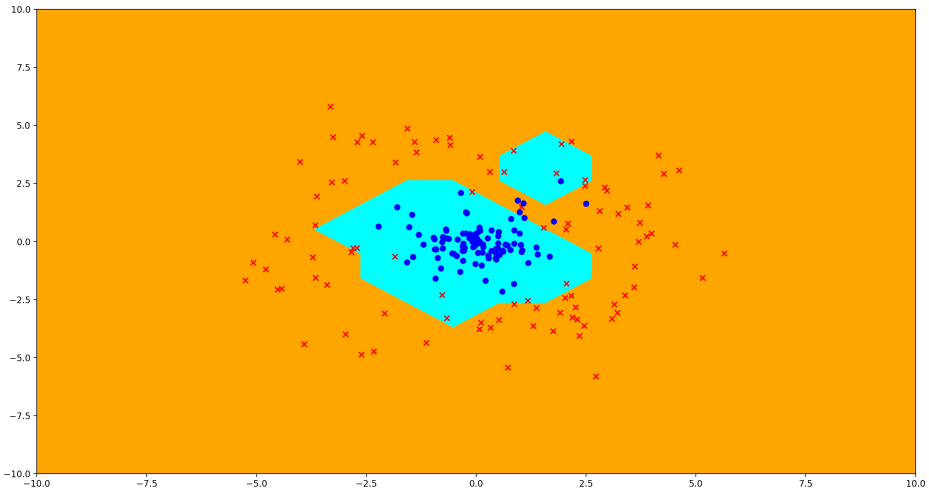
where only one entry in $\vec{\beta}$ vector is updated. Note in the context of this problem, the $\vec{\beta}_{i+1}^{(i)}$ on the right-hand side is always 0.

(b) Please see the code for a detailed implementation.

(c) For the dot product kernel, the decision boundary is plotted below:



For the radial basis function kernel, the decision boundary is plotted below:



We observe that the dot product kernel performs badly. The reason is that for the dot product kernel, the feature mapping ϕ is the identity function. Therefore, the feature space ϕ is the same as the input space. The perceptron algorithm learns a linear decision boundary on the feature space, and therefore on the input space. However, as clearly shown in the above figures, the data is not linearly separable.

Problem 6

(a) Please see the code for a detailed implementation.

(b) Please see the code for a detailed implementation.

(c) Please see the code for a detailed implementation.

(d) Please see the code for a detailed implementation.