CS 229 PSET #2: Supervised Learning #2

1) Logistic Regression: Training stability.

(a) What is the most notable difference in training the logistic regression model on datasets A & B?

⇒ Training on dataset A converged to epsilon almost immediately, while dataset B does not seem to converge ever.

(b) Investigate why the training procedure behaves unexpectedly on dataset B, but not on A.

⇒ When graphing the two datasets; we see that dataset B is linearly separable; (i.e. there $\exists$ a hyperplane such that separates the two classes with zero error.

- This is an issue for the MLE for logistic regression because:

$$\theta_{MLE} = \arg\max_{\theta} \ell(\theta)$$

$$\text{where } \ell(\theta) = -\sum_{i=1}^{m} \log\left(1 + e^{-y^i \cdot \theta^T x^i}\right)$$

- You want to $\arg\max \ell(\theta)$; since $\ell(\theta)$ is negated; you want the maximum, meaning as least negated as possible (smaller negative value)

⇒ so you want

$$\log\left(1 + e^{-y^i \cdot \theta^T x^i}\right) \to 0$$

to be as small as possible; so approach 0.

- looking specifically you want log(value) to be small as possible; so you want value to be as small as possible.

So you want $1 + e^{-y^i \cdot \theta^T x^i} \to 0$

so you want $e^{-y^i \cdot \theta^T x^i} \to 0$ ...

- Looking at this; since its $e^{-\text{exponent}}$; you want exponent $\to \infty$ for $e^{-y^i \cdot \theta^T x^i}$ to approach 0
- So objective is essentially $y^i \cdot \theta^T x^i \to \infty$

⇒ DATASET B case (linearly separable)

- ∃ a hyperplane when your classification error = 0.
- All training examples i: $\{x^i, y^i\} \Rightarrow y^i \cdot \theta^T x^i > 0$ (because all correctly classified)
- so given $y^i \cdot \theta^T x^i > 0 \; \forall \; (x^i, y^i)$ to make $y^i \cdot \theta^T x^i$ we can blow up θ; make it bigger & bigger; so as $\theta \to \infty$, our MLE continues to drop.

- hence; we never converge; as simply $\uparrow \theta$; our performance on MLE; gets better.

Versus Dataset A (not linerly separable)

- $\nexists$ a hyperplane where classification error $= 0$
  - $\exists$ points where $\{x^i, y^i\} \; y \Rightarrow y^i \cdot \theta^T x^i < 0$
  - so there are points $\{x^i, y^i\}$ such that $\sum y^i \cdot \theta^T x^i$ is $\sum -y^i \cdot \theta^T x^i$ i.e. decrease $Y \cdot \theta^T X$
- what does this mean? As $\theta \uparrow$; these points that "decrease" $Y \cdot \theta^T X$; get bigger as well; as θ is bigger
- $\therefore \uparrow \theta$ eventually "hurts" performance on MLE as $\exists$ a θ; such that $Y \cdot \theta^T X$ is maximized where $\theta \neq \infty$.
- So we have a point of convergence; no more improvements...

- TLDR: Dataset A is not linearly separable; so it converges because there is a max value of MLE; where $\theta \neq \infty$.

Dataset B is linearly separable; so the Max value of MLE is $\theta = \infty$; so θ keeps getting pushed bigger ⇒ to infinity & does not converge.

1c) For each of these possible modifications, state whether or not it would lead to the training algo converging on datasets, like B.

i. Using a different constant - learning rate

⇒ No, the key issue is that the stopping condition is

$$||\theta_{old} - \theta_{new}|| < e^{-15}$$ ✓

- depends on $\theta_{new} = \theta_{old} - \text{learning\_rate} * \text{grad}$.

- grad won't change; but theoretically learning rate could be small enough such that our $\Delta\theta$ hits the stopping condition. But the issue is that for this to happen the condition would be hit immediately in the first iteration since it is constant, so there would be no change to converge.

ii. Decreasing the learning rate over time (i.e. scale the initial learning rate by $1/t$, where t is the number of grad descent iterations thus far).

Yes this would work, we would reach an optimal solution; but it would eventually stop because learning rate would be small enough to miss stopping condition ✓

iii. Liner scaly of the input feature ✓

⇒ No. The resulting points will still be linearly separable.

iv. Adding a regularization term $\|\theta\|_2^2$ to the loss function ✓

Yes. This would penalize making θ infinitely big, eventually forcing a convergence

v. Adding a zero-mean Gaussian noise to the training data or labels

Yes, adding noise could make the data not linearly separable. ✓

d) Are support vector machines, which use the hinge loss, vulnerable to datasets like B? Why or why not? Give an informal justification

---

Quick intermission on hinge loss.

Hinge loss: For each sample (x^i, y^i), where $y^i \in \{-1, 1\}$: Hinge Loss $= \max(0, 1 - y^i \cdot w^T x^i)$

- you either get 0; or $(1 - y^i \cdot w^T x^i)$ as your value.
- This is loss, you want to keep it as low as possible

Correct prediction:

$w^T x^i$ = same sign as y^i

$\therefore y^i \cdot w^T x^i > 0$

$\therefore (1 - y^i \cdot w^T x^i) < 0$ (If margin ≥ 1 as well)

$\therefore$ Hinge loss $= \max(0, <0) = 0$

- so for correct predictions you get hinge loss $= 0$

Incorrect prediction: ☹

$w^i x^i \neq$ same sign as y^i

$\therefore y^i \cdot w^T x^i < 0$

$\therefore (1 - y^i \cdot w^T x^i) > 0$ (or if margin < 1 as well)

$\therefore$ Hinge loss $= \max(0, >0) =$ ⊕ positive number

- so for incorrect predictions you get hinge loss = pos number = you get punished.

SVM minimizes:

$$\min \frac{1}{2}\|w\|^2 + C \cdot \sum_{i=1}^{m} \text{Hinge Loss}$$

$\frac{1}{2}\|w\|^2$: keeps the margin large (regularization)

$\sum_{i=1}^{m}$ Hinge Loss: minimize classification error (using hinge loss)

- Hinge loss penalizes points that are inside the margin or misclassified

- encourages not just correct classification, but also a safe margin from the decision boundary.

---

back to d) Are SVM's which use hinge loss, vulnerable to datasets like B?

No. This is because hinge loss penalizes wrong predictions (adding a positive loss), but keeps hinge loss as 0 for correct classification. There is no incentive to push w larger, as the loss doesn't get better.

↳ the problem is the "Theta to infinity problem"

⇒ hinge loss = $\max(0, 1 - y \cdot \hat{y})$, where $\hat{y} = w^T x + b$

- so with linearly separable

$y \cdot \hat{y} > 0$ (since prediction is always right

so hinge loss = $\max(0,$ negative number)

so hinge loss = 0; and objective function $J(\hat{y}) = 0$ (no incentive to make w bigger).

2) Model calibration:

- Will try to understand the output $h_\theta(x)$ of the hypothesis function of a logistic regression model, in particular when we might treat the output as a probability.

- When the probabilities outputted by a model match empirical observation, the model is "well-calibrated"

- ex; a well calibrated models for $S \in \{x^i\}$ for which $h_\theta(x^i) \approx 0.7$, around 70% of those examples should have positive labels. In a well-calibrated model; this is true at every probability value

- Log reg tends to output well-calibrated probabilities; often not true for other classifiers like Naive Bayes or SVM's

- Suppose we have a training set $\{x^i, y^i\}_{i=1}^m$ with $x^i \in \mathbb{R}^{n+1}$ and $y^i \in \{0,1\}$. Assume we have an intercept term $x_0^i = 1$ for all i. Let $\theta \in \mathbb{R}^{n+1}$ be the maximum likelihood parameters learned after training a log regression model, in order for the model to be considered well-calibrated, given any range probability (a, b) such that $0 \leq a \leq b \leq 1$ and the examples x^i where the model outputs $h_\theta(x^i)$ fall in the range (a,b) the fraction of positives in that set of examples should be equal to the average of the model outputs for those examples. That is

$$\Big\downarrow \text{ must hold}$$

$$\frac{\sum_{i \in I_{a,b}} P(y^i = 1 \mid x^i; \theta)}{|i \in I_{a,b}|} = \frac{\sum_{i \in I_{a,b}} \mathbb{I}\{y^i = 1\}}{|\{i \in I_{a,b}\}|}$$

$P(y=1 \mid x; \theta) = h_\theta(x) = \frac{1}{1+\exp(-\theta^T x)}$

$I_{a,b} = \{i \mid i \in \{1, \dots, m\}, h_\theta(x^i) \in (a,b)\}$ is an index set of tray examples x^i where $h_\theta(x^i) \in (a,b)$ and $|S|$ denotes the size of the set S.

a) Show that the above property holds true for the described log reg model over the range $(a,b) = (0,1)$

Hint use the fact that we include a bias term

⇒ MLE for logistic regression.

⇒ Log reg:

$$f(\theta) = \frac{1}{1+e^{-\theta^T x}} = \frac{1}{1+\exp(-\theta^T x)}$$

MLE:

$$f(y=1 \mid x; \theta) = h_\theta(x)$$

$$f(y=0 \mid x; \theta) = 1 - h_\theta(x)$$

$\Rightarrow$ since $L(\theta) = \prod_{i=1}^{m} f(x^i|\theta)$

$\Rightarrow L(\theta) = \prod_{i=1}^{m} h_\theta(x) \cdot \prod_{i=1}^{m} (1 - h_\theta(x))$

sin $y_i = 1$; sin $y_i = 0$

$\Rightarrow L(\theta) = \prod_{i=1}^{m} h_\theta(x)^{y^i} \cdot \prod_{i=1}^{m} (1 - h_\theta(x))^{1-y^i}$

$\Rightarrow \ell(\theta) = \sum_{i=1}^{m} y^i \log h(x^i) + (1-y^i) \log (1 - h(x^i))$

- MLE occurs when $\nabla \ell(\theta) = 0$

$0 = \nabla \ell(\theta) = (y - h_\theta(x)) x_j$ ← from prev derivation (notes 1)

$0 = (y - h_\theta(x)) x_j$

$\Rightarrow x_j = j^{th}$ feature. Let $j = 0$ (0^{th} feature).

- given bias term $x_0^i = 1$; plug in

$\Rightarrow 0 = (y - h_\theta(x))\,1$

$0 = y - h_\theta(x)$

$y = h_\theta(x)$

$y^i = \mathbb{1}\{y^i = 1\}$

$h_\theta(x^i) = p(y^i = 1 | x^i; \theta)$

equivalent statements (plug in)

$\sum_{i \in I_{0,1}} \mathbb{1}\{y^{(i)} = 1\} = \sum_{i \in I_{0,1}} p(y^i = 1 | x^i; \theta)$

• divide both by the same val $|\{i \in I_{a,b}\}|$

$$= \frac{\sum_{i \in I_{0,1}} \mathbb{I}\{y^{(i)} = 1\}}{|\{i \in I_{a,b}\}|} = \frac{\sum_{i \in I_{0,1}} p(y^{(i)} = 1 \mid x^{(i)}; \theta)}{|\{i \in I_{a,b}\}|}$$

2b) If we have a binary classification model that is perfectly calibrated -- that is, the property we just proved holds for any $(a, b) \subset [0, 1]$ -- does this necessarily imply that the model achieves perfect accuracy? Is the converse necessarily true? Justify your answers.

⇒ No, since it only guarantees that the fraction of all total answers are the same prediction as label; it can still be the case that individual values of $x^{(i)}$ don't match.

---

* crash course on calibration

Well-calibrated: confidence matches reality.

Poorly calibrated: confidence is misleading.

Calibration means that the model's confidence score reflects the true likelihood of correctness

---

* A model can be overly or under-confident
* A well calibrated model ⇒ Accuracy ≈ confidence

* Hence the converse ⇒ A perfectly accurate model does not mean its well calibrated; because the model can still have confidence that doesn't match its accuracy. I.e. 100% accuracy; but the model is only 80% confident. A well calibrated model with 80% confidence would have ≈ 80% accuracy.

2c) Discuss what effect including L_2 regularization in the logistic regression objective has on model calibration.

⇒ The L_2 regularization you add to the objective function

$$\ell(\theta) = \sum_{i=1}^{m} y^i \log h(x^i) + (1-y^i)\log(1-h(x^i)) + \lambda \|\theta\|_2^2$$

$$\nabla \ell(\theta) = \sum_{i=1}^{m} (y - h_\theta(x^i))\, x^i + \nabla \lambda \|\theta\|_2^2$$

$x_j = 1$

$$\nabla \ell(\theta) = 0 = \sum_{i=1}^{m} y - h_\theta(x^i) + \nabla \lambda \left(\sum_{j=1}^{n} \theta_j^2 \right)$$

$$\Rightarrow \lambda \frac{\partial}{\partial \theta} \sum_{j=1}^{n} \theta_j^2$$

$$\Rightarrow \lambda \sum_{j=1}^{n} 2\theta_j$$

$$0 = \sum_{i=1}^{m} y - h_\theta(x^i) + \lambda \sum_{j=1}^{n} 2\theta_j$$

$$\sum_{i=1}^{m} h_\theta(x^i) - \lambda \sum_{j=1}^{n} 2\theta_j = \sum_{i=1}^{m} y^i$$

- no longer fits well-calibrated property

$$\frac{\sum_{i \in I_{a,b}} p(y^i = 1 \mid x^i; \theta)}{|\{i \in I_{a,b}\}|} = \frac{\sum_{i \in I_{a,b}} \mathbb{1}\{y^i = 1\}}{|\{i \in I_{a,b}\}|}$$

∴ It makes the logistic regression model no longer well-calibrated.

* Note: When logistic regression (GLM assumption) holds, all ranges $(a, b) \subset [0, 1]$ are well-calibrated.

In addition when S_{train} & S_{test} are from the same distribution and when the model has not overfit or underfit, logistic regression tends to be well-calibrated on unseen test data as well. It is therefore good in practice; when we care about level of uncertainty in model output.

3) Bayesian Interpretation of Regularization

- MAP estimation is diff from estimation techniques we learned so far; like MLE because it allows us to incorporate prior knowledge into the estimate.

a) Show that $\theta_{MAP} = \arg\max_\theta P(y|x;\theta)P(\theta)$ if we assume that $P(\theta) = P(\theta|x)$

$$\theta_{MAP} = \arg\max P(\theta|x,y)$$

$$\Rightarrow P(\theta|x;y) = \frac{P(y|x,\theta)P(\theta|x)}{P(y|x)}$$

$$\theta_{MAP} = \arg\max_\theta = \frac{P(y|x,\theta)P(\theta|x)}{P(y|x)}$$

$\arg\max_\theta \Rightarrow$ find the val of θ that maximizes

$$\therefore \arg\max_\theta \frac{P(y|x,\theta)P(\theta|x)}{P(y|x)}$$

$= \arg\max_{\theta} P(y|x;\theta) P(\theta|x)$ (since $P(y|x)$ independent of θ)

$\Rightarrow P(\theta) = P(\theta|x)$ subbing

$\therefore \arg\max_{\theta} P(y|x;\theta) P(\theta|x)$

$\Rightarrow \therefore \arg\max_{\theta} P(y|x;\theta) P(\theta)$

b) Recall that L_2 regularization penalizes the L_2 norm of that parameter while minimizing the loss (i.e. neg log likelihood in case of probabilistic models. Now we will show that MAP estimation with a zero-mean gaussian prior over θ, specifically $\theta \sim \mathcal{N}(0, \eta^2 I)$ is equivalent to apply L_2 regularization with MLE estim. Spec. show that.

$$\theta_{MAP} = \arg\min_{\theta} - \log P(y|x,\theta) + \lambda \|\theta\|_2^2$$

$\theta_{MAP} = \arg\max_{\theta} P(y|x;\theta) P(\theta)$

$= \arg\min_{\theta} - P(y|x;\theta) P(\theta)$

$= \arg\min_{\theta} -[\log P(y|x;\theta) + \log P(\theta)]$

$$\theta_{MAP} = \arg\min_{\theta} - \log P(y|x,\theta) - \log P(\theta)$$

$= \lambda \|\theta\|_2^2$ in MLE

try: $\log p(\theta)$ given $\theta \sim \mathcal{N}(0, \eta^2 I)$

pdf of a d-dimensional multivariate normal distribution $\mathcal{N}(\mu, \Sigma)$ for a vector x is given by:

$$p(x) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

$$p(\theta) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\theta-\mu)^T \Sigma^{-1} (x-\theta)\right)$$

$$= p(\theta) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\theta)^T \Sigma^{-1} (\theta)\right)$$

$$= \frac{1}{2\pi^{\frac{n}{2}} |\eta^2 I|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\theta)^2 (\eta^2 I)^{-1}\right)$$

$$= (2\pi)^{-\frac{n}{2}} (\eta^2 I)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\theta)^2 (\eta^2 I)^{-1}\right)$$

$\Rightarrow$ sub into θ_{MAP}

$$\theta_{MAP} = \arg\min_{\theta} \left(-\log p(y|x;\theta) - \log p(\theta)\right)$$

$$\theta_{MAP} = \underset{\theta}{\operatorname{argmin}} \left(-\log P(y|x;\theta) - \log \left[(2\pi)^{-\frac{n}{2}} (\eta^2 I)^{-\frac{1}{2}} \exp\left(-\frac{1}{2} (\theta)^2 (\eta^2 I)^{-1} \right) \right] \right)$$

$$\dots \operatorname{argmin} \left[\log \left[(2\pi)^{-\frac{n}{2}} (\eta^2 I)^{-\frac{1}{2}} \exp\left(-\frac{1}{2} (\theta)^2 (\eta^2 I)^{-1} \right) \right] \right]$$

$$= \underset{\theta}{\operatorname{argmin}} \left[\log (2\pi)^{-\frac{n}{2}} + \log (\eta^2 I)^{-\frac{1}{2}} + \left(-\frac{1}{2}\theta \right)^2 (\eta^2 I)^{-1} \right]$$

constants w.r.t. $\operatorname{argmin}_\theta$ = drop

$$= \underset{\theta}{\operatorname{argmin}} - \left[\frac{1}{2} \theta^2 (\eta^2 I)^{-1} \right]$$

⋮ plug back in

$$\theta_{MAP} = \underset{\theta}{\operatorname{argmin}} \left(-\log P(y|x;\theta) - \left[-\frac{1}{2} \theta^2 (\eta^2 I)^{-1} \right] \right)$$

$$= \underset{\theta}{\operatorname{argmin}} -\log P(y|x;\theta) + \frac{1}{2} \theta^2 (\eta^2 I)^{-1}$$

$$= \underset{\theta}{\operatorname{argmin}} -\log P(y|x;\theta) + \frac{1}{2} \frac{\theta^2}{\eta^2}$$

⇒ original form:

$$\theta_{MAP} = \underset{\theta}{\operatorname{argmin}} -\log P(y|x;\theta) + \lambda \|\theta\|_2^2$$

$$\frac{1}{2} \frac{\theta^2}{\eta^2} = \lambda \|\theta\|_2^2$$

$$\frac{1}{2\eta^2}\theta^2 = \lambda \|\theta\|_2^2$$

$$\therefore \lambda = \frac{1}{2\eta^2}$$

c) Now consider a specific instance, a linear regression model given by $y = \theta^T x + \epsilon$; where $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

Like before, assume a Gaussian prior on this model such that $\theta \sim \mathcal{N}(0, \eta^2 I)$. For notation, let X be the design matrix of all the training example inputs, where each row vector is one example input, and $\vec{y}$ be the column vector of all the example outputs. Come up with closed form for θ_{MAP}

$$\theta_{MAP} = \arg\max_{\theta} p(\theta \mid X, y)$$

* MAP $\Rightarrow$ probability of θ given X & y. "Let's find the most likely θ given the θ": which means both X & Y.

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

$$\Rightarrow \theta_{MAP} = \arg\max_{\theta} p(\theta \mid X, y) = \frac{p(y \mid X, \theta) \cdot p(\theta \mid X)}{p(y \mid X)}$$

$\arg\max_\theta \Rightarrow P(y|X)$ is irrelevant for maximizing since it doesn't depend on θ

$$\Rightarrow \theta_{MAP} = \arg\max_\theta P(y|X,\theta) \cdot P(\theta|X)$$

*Bayesian stats assumes θ is not something we observe, its something we believe. Even before seeing any data, we have a prior belief about θ. θ exists independently outside of data we observe. Hence, "just knowing X doesn't change what we believe about θ"

Hence $P(\theta|X) = P(\theta)$

$$\Rightarrow \theta_{MAP} = \arg\max_\theta P(y|X,\theta) \cdot P(\theta)$$

PDF for theta:

$$P(x) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

$$P(\theta) = \left(\frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}}\right) \exp\left(-\frac{1}{2}(\theta)^T\Sigma^{-1}(\theta)\right)$$

$P(y|X)$ ... multivariate gaussian as X is also a vector ... viewed as μ & Σ

$$\epsilon \sim \mathcal{N}(0, \sigma^2)$$

$$\theta \sim \mathcal{N}(0, \eta^2 I)$$

$p(y|X,\theta) = ?$

- Find μ of distribution

$y = \theta^T x + \epsilon$

$\epsilon \to 0$

$\therefore y = X\theta^T = \mu$

- covariance, $X\theta$ is deterministic, so covariance comes from ϵ

$= \sigma^2$

$\therefore \Sigma = \sigma^2 I$

$$p(y|X,\theta) \sim \mathcal{N}(X\theta, \sigma^2 I)$$

⇒ find p.d.f. of single variable normal distribution for y^i

given $Y \sim N(\mu, \sigma^2)$

$$p(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(y-\mu)^2}{2\sigma^2}\right\} \quad ; \text{ where } y \sim N(\mu, \sigma^2)$$

$\Rightarrow p(y|X,\theta) \Rightarrow N(X\theta, \sigma^2 I)$

↳ for each individual x^i

$p(y^i|x^i,\theta) \; N \Rightarrow (x^i\theta, \sigma^2)$

$\mu = \theta^T x^i$

variance $= \sigma^2$

$y = ?$ ⇒ univariate PDF, we calculate

the pdf at each point y^i ... so $y = y^i$

$$P(y^i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ - \frac{(y^i - \theta^T x^i)}{2\sigma^2} \right\}$$

$P(y^i)$ is a single point

$$P(y \mid X, \theta) = \prod_{i=1}^{i=n} \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left\{ - \frac{(y^i - \theta^T x^i)^2}{2\sigma^2} \right\}$$

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

$$= \frac{1}{2\pi^{\frac{m}{2}} \sigma^m} \exp\left\{ - \frac{1}{2\sigma^2} (y^i - \theta^T x^i)^2 \right\}$$

$$\Rightarrow \|x\|_2 = \sqrt{x_i^2 + x_2^2}$$

$$\|\vec{y} - X\theta\|_2 = \sqrt{(y^i - \theta^T x^i)^2}$$

$$\Rightarrow \|\vec{y} - X\theta\|_2^2 = (y^i - \theta^T x^i)^2$$

$$P(y|X,\theta) = \frac{1}{2\pi^{\frac{m}{2}} \sigma^m} \exp\left\{ - \frac{1}{2\sigma^2} \|\vec{y} - X\theta\|_2^2 \right\}$$

from before... $\log p(\theta) = \frac{1}{2\eta^2}\|\theta\|_2^2$

$\Rightarrow$ plug back in...

$$\theta_{MAP} = \arg\min_\theta \; -\log p(y|x,\theta) - \log p(\theta)$$

$$\theta_{MAP} = \arg\min_\theta \; -\log p(y|x,\theta) + \frac{1}{2\eta^2}\|\theta\|_2^2$$

$$\theta_{MAP} = \arg\min_\theta \left(-\log \left[\frac{1}{2\pi^{\frac{m}{2}}\sigma^m} \exp\left(-\frac{1}{2\sigma^2}\|\vec{y} - X\theta\|_2^2 \right) \right] \right) \cdots$$

$$= \arg\min_\theta \left(-\log \frac{1}{2\pi^{\frac{m}{2}}\sigma^m} - \frac{1}{2\sigma^2}\|\vec{y} - X\theta\|_2^2 \right) \cdots$$

drop b/c $\arg\min_\theta$ (constant)

$$= \arg\min_\theta \left(-\frac{1}{2\sigma^2}\|\vec{y} - X\theta\|_2^2 - \frac{1}{2\eta^2}\|\theta\|_2^2 \right)$$

$$\therefore J(\theta) = -\frac{1}{2\sigma^2}\|\vec{y} - X\theta\|_2^2 - \frac{1}{2\eta^2}\|\theta\|_2^2$$

$$= \frac{1}{2\sigma^2}\|X\theta - \vec{y}\|_2^2 - \frac{1}{2\eta^2}\|\theta\|_2^2$$

$\nabla \|X\| = \sqrt{X^T X}$

$\nabla \|X\|^2 = X^T X$ (u-sub)

$$= \frac{1}{2\sigma^2}\nabla (X\theta - \vec{y})^T (X\theta - \vec{y}) - \frac{1}{2\eta^2}\|\theta\|_2^2$$

$$= \frac{1}{2\sigma^2} \nabla (X\theta - \vec{y})^T (X\theta - \vec{y}) - \frac{1}{2\eta^2} \|\theta\|_2^2$$

gradient of quadratic sum

$$= 2X^T (X\theta - \vec{y})$$

$$= \frac{1}{2\sigma^2} \cdot 2X^T (X\theta - \vec{y}) + \frac{1}{2\eta^2} \|\theta\|_2^2$$

$$= \frac{1}{\sigma^2} X^T (X\theta - \vec{y}) + \frac{1}{\eta^2} \cdot \nabla \theta^T \theta$$

$$\nabla J = \frac{1}{\sigma^2} \left(X^T X\theta - X^T \vec{y} \right) + \frac{1}{\eta^2} \theta$$

$$\nabla J_\theta = 0$$

$$\Rightarrow 0 = \frac{1}{\sigma^2} (X^T X\theta - X^T \vec{y}) + \frac{1}{\eta^2} \theta$$

$$0 = \frac{1}{\sigma^2} X^T X\theta - \frac{1}{\sigma^2} X^T \vec{y} + \frac{1}{\eta^2} \theta$$

$$\frac{1}{\sigma^2} X^T \vec{y} = \frac{1}{\sigma^2} X^T X\theta + \frac{1}{\eta^2} \theta$$

$$\frac{1}{\sigma^2} X^T \vec{y} = \theta \left(\frac{1}{\sigma^2} X^T X + \frac{1}{\eta^2} \right)$$

$$\frac{\frac{1}{\sigma^2} X^T \vec{y}}{\frac{1}{\sigma^2} X^T X + \frac{1}{\eta^2}} = \theta$$

$$X^T \vec{y} \cdot \left(X^T X + \frac{\sigma^2}{\eta^2}\right)^{-1} = \theta$$

$$\theta_{MAP} = \underset{\theta}{\operatorname{argmin}}\ J(\theta) = X^T \vec{y} \cdot \left(X^T X + \frac{\sigma^2}{\eta^2}\right)^{-1}$$

d) Laplace

$$f_L(z \mid \mu, b) = \frac{1}{2b} \exp\left(-\frac{|z-\mu|}{b}\right)$$

- consider a linear regression model given by $y = X^T\theta + \epsilon$ $\epsilon \sim N(0, \sigma^2)$. Assume a laplace prior $\theta \sim L(0, bI)$
- Find θ_{MAP}

$$\theta_{MAP} = \operatorname{argmin}_\theta -\log p(y \mid x, \theta) - \log p(\theta)$$

$$f_L(\theta^i) = \frac{1}{2b} \exp\left(-\frac{|z-\mu|}{b}\right) \qquad \theta = (z \mid \mu, b)$$

$\theta = z$, $\mu = 0$ since our prior

$$f_L(\theta^i) = \frac{1}{2b} \exp\left(-\frac{|\theta^i - 0|}{b}\right)$$

$$f_L(\theta) = \prod_{i=1}^{i=n} \frac{1}{2b} \exp\left(-\frac{|\theta^i|}{b}\right)$$

$f_L(y|x,\theta) = ?$

$\Rightarrow y = x^T\theta + \epsilon$; where $\epsilon \sim N(0, \sigma^2)$

$y \sim N(0, \sigma^2)$

$$f_L(y|x,\theta) = \prod \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^i - \theta^T x^i)^2}{2\sigma^2}\right)$$

$\Rightarrow$

$$\theta_{MAP} = \arg\min_\theta -\log p(y|x,\theta) - \log p(\theta)$$

$$= \arg\min_\theta -\log\left[\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^i - \theta^T x^i)^2}{2\sigma^2}\right)\right] - \log\left[\left[\frac{1}{(2b)^n} \exp\left(\frac{-|\theta^i|}{b}\right)^n\right]\right]$$

$$= \arg\min_\theta \sum_{i=1}^{m}\left(\frac{(y^i - \theta^T x^i)^2}{2\sigma^2}\right) + \sum_{i=1}^{n} \frac{1}{b}|\theta_i|$$

$$= \arg\min_\theta \frac{1}{2\sigma^2} \sum_{i=1}^{m} (y^i - \theta^T x^i)^2 + \sum_{i=1}^{n} \frac{1}{b}|\theta_i|$$

$$\|X\|_2 = \sqrt{\sum x^{i\,2}} \Rightarrow \sum x^{i\,2} = \|X\|_2^2$$

$$= \arg\min_\theta \frac{1}{2\sigma^2} \|X\theta - \vec{y}\|_2^2 + \frac{1}{b}\|\theta\|_1$$

$$= \arg\min_\theta \|X\theta - \vec{y}\|_2^2 + \frac{2\sigma^2}{b}\|\theta\|_1$$

$\therefore J(\theta) = \|X\theta - \vec{y}\|_2^2 + \frac{2\sigma^2}{b}\|\theta\|_1$

$\theta_{MAP} = \arg\min_\theta J(\theta)$

• what is τ?

$\Rightarrow J(\theta) = \|X\theta - \vec{y}\|_2^2 + \tau\|\theta\|_1$

$J(\theta) = \|X\theta - \vec{y}\|_2^2 + \frac{2\sigma^2}{b}\|\theta\|_1$

$\therefore \tau = \frac{2\sigma^2}{b}$

---

4) Constructing Kernels:

⇒ First; kernels walkthrough:

1) What is a kernel?

⇒ A kernel func $K(x, z)$ is a way to compute a dot product between feature mappings without explicitly mapping inputs to a high dimensional space. That is, you may have a mapping

$\phi : \mathbb{R}^n \to \mathcal{H}$

$\mathcal{H}$ is some Hilbert space (his/her dimensional) and the kernel function is further using this mapping ϕ

$K(x, z) = \langle \phi(x), \phi(z) \rangle$

• This allows kernel-based algos; like SVMs to work in $\mathcal{H}$ without ever computing $\phi(x)$ directly.

Next we need to know a valid kernel:

• A func $K(x, z)$ is a valid kernel iff:

1) Symmetric: $K(x, z) = K(z, x)$

2) positive semi definite: (PSD): for any finite set $\{x^1, \dots x^m\}$ the matrix $K \in \mathbb{R}^{m \times m}$ with entries $K_{ij} = k(x^i, x^j)$ satisfies $\forall c \in \mathbb{R}^m$, $c^T K c \geq 0$ (all eigenvalues of kernel matrix K are non-negative)

* Core Idea: A kernel is a symmetric, positive semidefinite function. Any operation that preserves symmetry & positive semi definiteness gives another valid kernel

---

Kernel construction rules

- If K_1 and K_2 are valid kernels then:

1) Addition: $K(x, z) = K_1(x, z) + K_2(x, z) \Rightarrow$ valid kernel

2) Scalar multiplication: $K(x, z) = a K_1(x, z)$ for $a > 0$ is valid

3) Multiplication: $K(x, z) = K_1(x, z) K_2(x, z) \Rightarrow$ valid

4) Function application: If f is a function and $K(x, z) = f(x) f(z)$ then this is a valid kernel because $\phi(x) = f(x)$, a 1-D feature mapper

5) composition: If $\phi: \mathbb{R}^n \to \mathbb{R}^d$, and K_3 is a kernel over $\mathbb{R}^d \times \mathbb{R}^d$, then $K(x, z) = K_3(\phi(x), \phi(z))$ is also a kernel

6) polynomials with Positive coefficients: If $P(x)$ is a polynomial with positive coefficients, and K_1 is a kernel

then $f(K_1(x,z))$ is also a kernel.

Finding counterexamples ⇒ prove that its not PSD?

- using the quadratic form ⇒ $x^T A x$ that takes a vector x and outputs a scalar value, where A is a symmetric matrix and x^T represents the transpose of x.
- K is not positive semi-definite if you can find a vector z, such that

$$z^T K z < 0$$

- for example

$K = K_1 - K_2$

let $K_1 = 2K_2$

then $K_2 = 2K_1$

then: $K = K_1 - 2K_1 = -K_1$

and therefore $z^T K z = z^T - K_1 z \leq 0$ ...

- therefore $z^T K z < 0$; and K is not positive semidefinite

* Remember that positive semidefinite dfn is for a matrix K, its quadratic form $z^T K z \geq 0$.

Back to Q4.

k_1 & k_2 are kernels $\mathbb{R}^n \times \mathbb{R}^n$, $\forall a \in \mathbb{R}^+$ & $f: \mathbb{R}^n \mapsto$ be a real valued funcn; let $\phi: \mathbb{R}^n \to \mathbb{R}^d$ be a funcn mapping from $\mathbb{R}^n$ to $\mathbb{R}^d$. Let k_3 be a kernel over $\mathbb{R}^d \times \mathbb{R}^d$, and let $p(x)$ be a polynomial over x with positive coefficients.

(a) $k(x, z) = k_1(x, z) + k_2(x, z)$

k_1 is PSD & k_2 is PSD

$k_1 + k_2$ is also PSD

$k = k_1 + k_2$

PSD defn:

$z^T K z \geq 0$

$\Rightarrow z^T (k_1 + k_2) z$

$(z^T k_1 + z^T k_2) z$

$z^T k_1 z + z^T k_2 z \geq 0$; $\therefore k = k_1 + k_2$ is PSD & a valid kernel

(b) $k(x, z) = k_1(x, z) - k_2(x, z)$

$\Rightarrow z^T K z \geq 0$

let $k_2 = 2k_1$

$\Rightarrow K = K_1 - K_2 = K_1 - 2K_1 = -K_1$

$\Rightarrow z^T K z = z^T - K_1 z = -z^T K_1 z$

$\because -z^T K_1 z < 0$

$\therefore z^T K z < 0$

$\therefore K = K_1 - K_2$ is not PSD & not a valid kernel

c) $k(x, z) = a k_1(x, z)$; $a \in \mathbb{R}^+$ (positive real num)

PSD defn: $z^T K z \geq 0$

$z^T K z = z^T a K_1 z$

$= a z^T K_1 z$

K_1 is a valid kernel ; $\therefore z^T K_1 z \geq 0$

a is a positive real number: $a > 0$

$\therefore a z^T K_1 z =$ pos

(+) · (+)

$\therefore a z^T K_1 z \geq 0$

$\therefore K = a K_1$ is PSD & a valid kernel

d) $k(x, z) = -a k_1(x, z)$

PSD defn: $z^T K z \geq 0$

$K = -aK_1(x, z)$

$z^T K z = z^T - aK_1 z = -a z^T K_1 z$

K_1 is PSD; so $z^T K_1 z \geq 0$

$\therefore -a z^T K_1 z < 0$

$\ominus \times \oplus$

$\therefore z^T K z < 0$

$\therefore K = -aK_1$ is not PSD & not a valid kernel

e) $K(x, z) = K_1(x, z) K_2(x, z)$

PSD defn: $z^T K z \geq 0$

$z^T K z = z^T K_1 K_2 z$

- Mercer's theorem: every valid kernel is an inner product in some (possibly infinite-dimensional) space

$K(x, x') = \langle \phi(x), \phi(x') \rangle$

x & x' are in your original input space

$\phi(x)$, $\phi(x')$ are the higher dimensional feature values, given by a ϕ (feature map)

- $K(x, x')$ = the inner product of $\phi(x)$ and $\phi(x')$

- $K(x, x')$ (kernel function) gives you the inner product between $\phi(x)$ & $\phi(x')$ without ever the need to compute $\phi(x)$

Mercer's Theorem:

If $K(x, x')$ is a symmetric, posite definite kernel, then it can be written as: $K(x, x') = \sum_{i=1}^{\infty} \lambda_i \phi_i(x) \phi_i(x')$

where:

- $\lambda_i \geq 0$ are eigenvalues
- $\phi_i(x)$ are orthogonal eigen functions
- This sum acts like a dot product in a (possibly infinite-dimensional) space
- So it gives us a feature mapping

$$\phi(x) = (\sqrt{\lambda_1}\phi_1(x), \sqrt{\lambda_2}\phi_2(x), \ldots)$$

such $K(x, x') = \langle \phi(x), \phi(x') \rangle$

- So mercer guarantees that a valid kernel function K, is an inner product in some feature space.
- guarantees the existence of a mapping $\phi(x)$.
- kernel trick writes: "You can trust this kernel -- it's

the dot product of some high-dimensional representation, even if you never see it

$$K_1(x, z) = \langle \phi(x), \phi(z) \rangle$$

$$K_2(x, z) = \langle \phi_2(x), \phi_2(z) \rangle$$

(from Mercer's theorem)

ϕ & ϕ_2 are just functions that create a M dimensional & N dimensional vector (higher feature space)

• write the product K_1 & K_2 in terms of ϕ_1 & ϕ_2..

$$K = K_1 \cdot K_2$$

$$K(x, z) = \langle \phi(x), \phi(z) \rangle \cdot \langle \phi_2(x), \phi_2(z) \rangle$$

$$= \sum_{i=1}^{m} \phi(x^i)\, \phi(z^j) \cdot \sum_{i=1}^{n} \phi_2(x^i)\, \phi_2(z^i)$$

$$= \sum_{i=1}^{m} \sum_{i=1}^{n} \left[\phi(x^i)\, \phi_2(x^i)\right] \left[\phi(z^i)\, \phi_2(z^i)\right]$$

let $c(x^i) = \phi(x^i)\, \phi_2(x^i)$

$$= \sum_{i=1}^{m} \sum_{i=1}^{n} c(x^i)\ (z^i)$$

$= C(x)^T C(z)$

C is a (m, n) dimensional vector s.t. $C = \phi(x)\phi_2(x)$

since there is a feature map C; that can map values x to $C(x)$; and we can denote $K(x,z)$ as a inner product; $C(x)^T C(z)$; by Mercer's theorem, $K(x,z)$ is a valid kernel.

---

f) $K(x,z) = f(x) f(z)$

$\Rightarrow f: \mathbb{R}^n \Rightarrow \mathbb{R}$ be a real-valued function

PSD: $z^T K z \geq 0$

$z^T f(x) f(z) z$

$f(x) \Rightarrow \mathbb{R}^n \rightarrow \mathbb{R}$; so inner product

$$K(x,z) = \sum_{i=1}^{m} \sum_{i=1}^{n} f(x)^i f(z)^i$$

$$= C(x)^T C(z)$$

- we can write $K(x,z)$ as an inner product using feature map C, we know K_p is a valid kernel.

g) $k(x,z) = k_3(\phi(x), \phi(z))$

$\Rightarrow$ given: k_3 is a valid kernel, k is a valid kernel as $k = k_3$, and $k_3(x,z)$ is a kernel no matter the inputs.

h) $k(x,z) = p(k_1(x,z))$

$p(x)$ is a polynomial over x, with pos coefficients.

$$k(x,z) = p(k_1(x,z))$$

$\Rightarrow z^T k(x,z) z \geq 0$

$$z^T k(x,z) z = z^T p(k_1(x,z)) z$$

- $k_1(x,z)$ is a valid kernel, $k_1(x,z) \geq 0$ & p is pos only coefficients.

$z^T [\oplus \cdot \oplus] z$ (pos coeff times ≥ 0 kernel fnc)

$\therefore z^T p(k_1(x,z)) z \geq 0$

$\therefore K = p(k_1(x,z))$ is a valid kernel as $z^T p(k_1(x,z)) z \geq 0$

5) Kernelizing the perceptron.

Rmbr: perceptron:

input: $X = [x_1, x_2 \dots x_n]$

Applies: weights $\theta = [\theta_1, \theta_2 \dots]$
bias $= b$

- Computes weighted sum:

$$z = \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \dots$$

- Applies activation function

$$\text{output} = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Kernel trick:

- we replace x with $\phi(x)$ (we don't know $\phi(x)$)
- But eventually we have inner product

$$\phi(x_i) \cdot \phi(x_j) \Rightarrow K(x_i, x_j)$$

- For each misclassified point, we update the prediction

Kernelizing the perceptron

1) how to represent the weights?

- we can't store w directly; since we don't know ϕ

$$w = \sum_i a_i y_i \phi(x_i)$$

2) How to represent the new predict function

- plug back into decision fcn:

$$f(x) = \text{sign}(w \cdot \phi(x))$$

normally its $w \cdot x$; but its $\phi(x)$ because we are mapping x to higher dimensional...

- put w back into it

$$f(x) = \text{sign}\left(\sum_i a_i y_i \phi(x_i) \cdot \phi(x)\right)$$

3) We don't know ϕ, what do we do?

apply the kernel trick

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$$

- sub in the kernel where possible

$$f(x) = \text{sign}\left(\sum_i a_i y_i \underbrace{\phi(x_i) \cdot \phi(x)}_{K(x_i, x_j)} \right)$$

$$f(x) = \text{sign}\left(\sum_i a_i y_i \cdot K(x_i, x) \right)$$

---

a is a list of points that is a record of mistakes.

- we need to predict $f(x_i)$ from updates with a.
- To update "a" to include mistakes, set $a_i = 0$ for correct; and $a_i = 1$ for incorrect.

How to add a learning rate?

- Make a = learning rate on updates for a_i. Since we want to multiply the effect of every training point by a learning rate, we can do this by setting $a_i = \alpha$ since a_i is multiplied for every point in the predict function:

$$f(x) = \text{sign}\left(\sum_i a_i y_i \phi(x_i) \cdot \phi(x) \right)$$

a)

i. How will you implicitly represent the high-dimensional parameter vector θ^i, including how the initial value $\theta^0 = 0$ is represented?

By keeping track & adding all previous misclassified outputs $y_i \phi(x^i)$ in the predict function, we get a predict function

$$f(x) = \sum_{i=1}^{m} a_i y_i \phi(x_i) \phi(x).$$

By the kernel rule, this simplifies to

$$f(x) = \sum_{i=1}^{m} a_i y_i K(x_i, x)$$; allowing us to represent θ^i, a vector whose dimension is the same as the feature vector $\phi(x)$ implicitly without knowing ϕ; by using the kernel function in the predict function.

---

ii) How will you efficiently make a prediction on a new input x^{i+1} i.e. how will you compute $h_\theta(x^{i+1}) = g(\theta^{i^T} \phi(x^{i+1}))$, using your representation of θ^i

$\Rightarrow$ we take our prediction function that implicitly represents ϕ, by including the kernel function:

$$f(x) = \sum_{i=1}^{m} a_i y_i K(x_i, x)$$

and plug in x^{i+1}; and apply $g(x)$ to it: so:

$$g(f(x^{i+1}))$$

iii) Update rule update

- For misclassified values we add to the prediction function, by adding the term:

$$a_i y_i \phi(x_i)$$

a_i will be the learning rate, so that for misclassified points we add $y_i \phi(x_i)$ scaled by λ to "move" our future predictions.

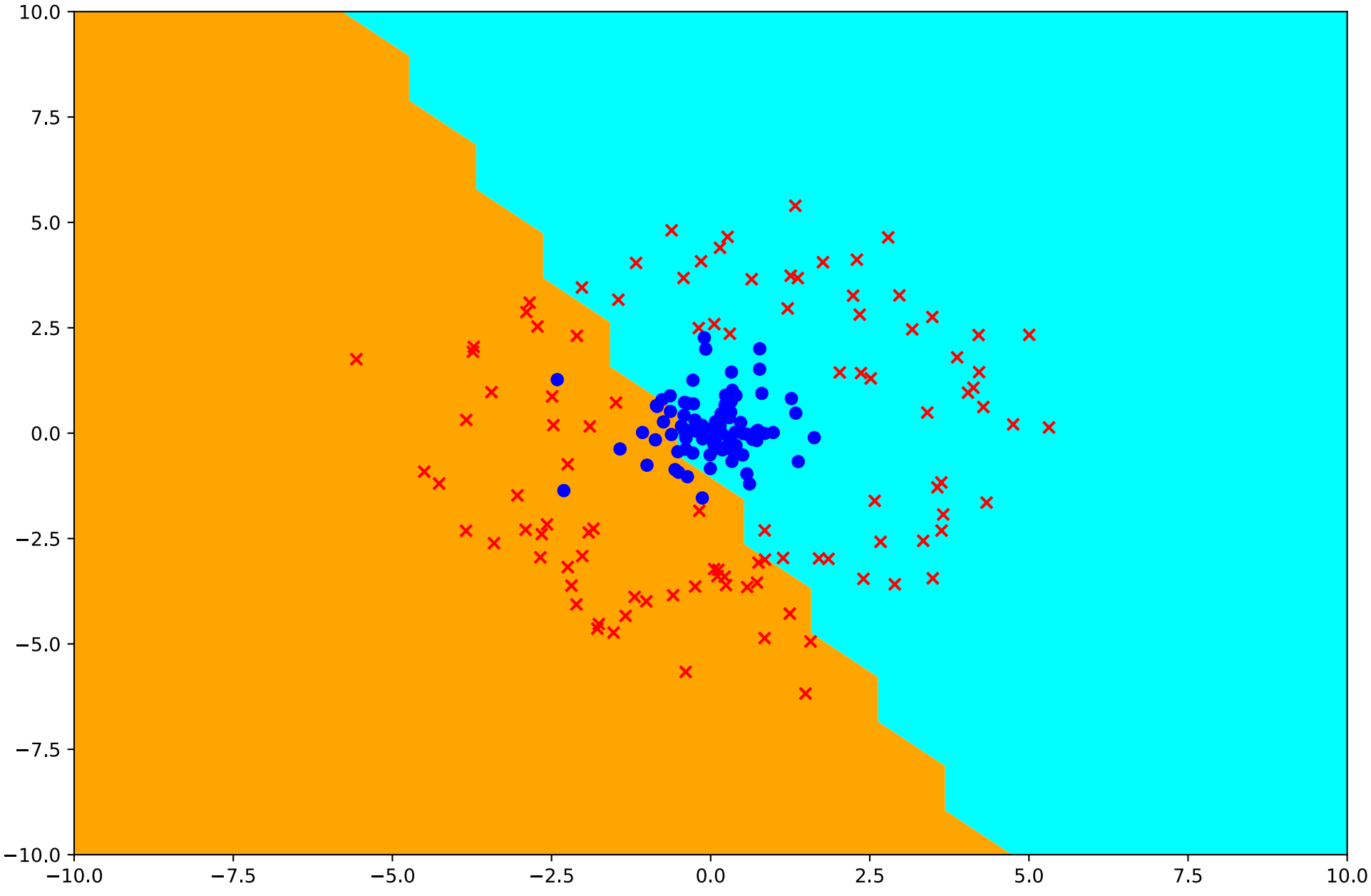
This results in:

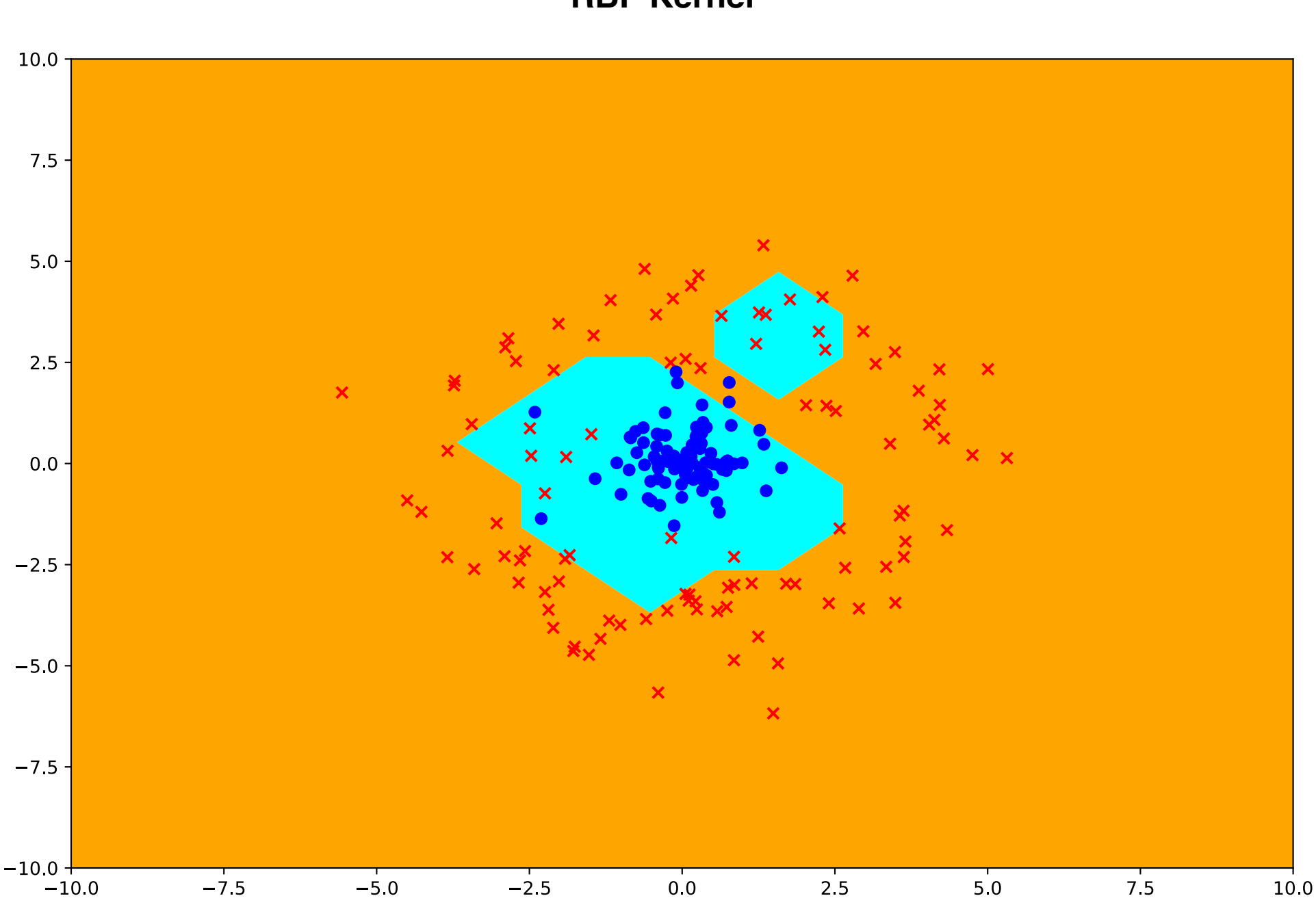
$$f(x) = \sum_{i=1}^{m} a_i y_i k(x_i, x)$$

b) In PS2/src in jupyter notebook

c)

Outputs


Dot product kernel
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


RBF Kernel
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

The dot-product kernel performs badly. This is because the data is not linearly-separable, and the dot-product (linear) kernel struggles with complex, non-linear data.