CS 229, Fall 2018
Problem Set #2 Solutions: Supervised Learning II

YOUR NAME HERE (`YOUR SUNET HERE`)

Due Wednesday, Oct 31 at 11:59 pm on Gradescope.

Notes: (1) These questions require thought, but do not require long answers. Please be as concise as possible. (2) If you have a question about this homework, we encourage you to post your question on our Piazza forum, at `http://piazza.com/stanford/fall2018/cs229`. (3) If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on Handout #1 (available from the course website) before starting work. (4) For the coding problems, you may not use any libraries except those defined in the provided `environment.yml` file. In particular, ML-specific libraries such as scikit-learn are not permitted. (5) To account for late days, the due date listed on Gradescope is Nov 03 at 11:59 pm. If you submit after Oct 31, you will begin consuming your late days. If you wish to submit on time, submit before Oct 31 at 11:59 pm.

All students must submit an electronic PDF version of the written questions. We highly recommend typesetting your solutions via LaTeX. If you are scanning your document by cell phone, please check the Piazza forum for recommended scanning apps and best practices. All students must also submit a zip file of their source code to Gradescope, which should be created using the `make_zip.py` script. In order to pass the auto-grader tests, you should make sure to (1) restrict yourself to only using libraries included in the `environment.yml` file, and (2) make sure your code runs without errors when running `p05_percept.py` and `p06_spam.py`. Your submission will be evaluated by the auto-grader using a private test set.

1. **[15 points] Logistic Regression: Training stability**

In this problem, we will be delving deeper into the workings of logistic regression. The goal of this problem is to help you develop your skills debugging machine learning algorithms (which can be very different from debugging software in general).

We have provided an implementation of logistic regression in `src/p01_lr.py`, and two labeled datasets A and B in `data/ds1_a.txt` and `data/ds1_b.txt`.

Please do not modify the code for the logistic regression training algorithm for this problem. First, run the given logistic regression code to train two different models on A and B. You can run the code by simply executing `python p01_lr.py` in the `src` directory.

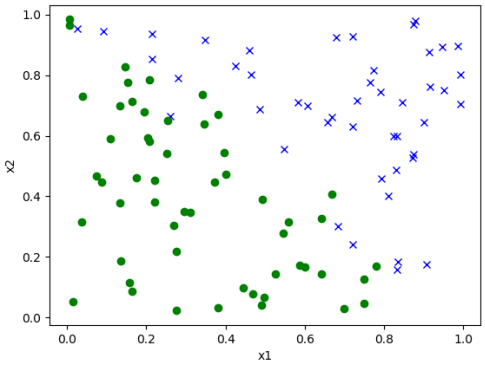
(a) [2 points] What is the most notable difference in training the logistic regression model on datasets A and B?

Answer: the model can converge easily on Dataset A, while the model can hardly converge on Dataset B.

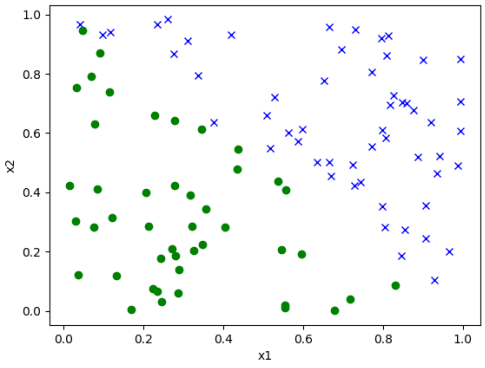
(b) [5 points] Investigate why the training procedure behaves unexpectedly on dataset B, but not on A. Provide hard evidence (in the form of math, code, plots, etc.) to corroborate your hypothesis for the misbehavior. Remember, you should address why your explanation does *not* apply to A.

Hint: The issue is not a numerical rounding or over/underflow error.

Answer: The dataset B is perfectly linear separable. In this case, when the logistic regression



Dataset A



Dataset B

finds a decision boundary that can separate the dataset perfectly, the algorithm can decrease the negative log-likelihood function by multiplying the θ by a arbitrary number. However, multiplying the θ by a number doesn't change the decision boundary. Like the functional margin of SVM, we can make the θ arbitrarily large without really changing anything meaningful. Logistic regression can converge on the dataset A, because that dataset is not linearly separable.

(c) [5 points] For each of these possible modifications, state whether or not it would lead to the provided training algorithm converging on datasets such as B. Justify your answers.

i. Using a different constant learning rate.

ii. Decreasing the learning rate over time (e.g. scaling the initial learning rate by $1/t^2$, where t is the number of gradient descent iterations thus far).

iii. Linear scaling of the input features.

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

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

Answer:
(i) No. The theta will still arbitrarily increase but in a slow speed.
(ii)Yes. This modification will make the learning rate decreasing very quickly in order to force the algorithm to stop updating the θ.
(iii) No. The dataset will still be linear separable.
(iv) Yes. This will prevent θ to be arbitrarily large.
(v) No. Actually, the result depends on how much the noise is. It should be large enough to make the dataset not linearly separable. If the noise is small, the linear-separable-dataset problem is still not solved.

(d) [3 points] Are support vector machines, which use the hinge loss, vulnerable to datasets like B? Why or why not? Give an informal justification.

Answer: Support vector machines are not vulnerable to datasets like B. Maxmizing the geometric margin contains a restriction that the 2-norm of θ must be 1. So, the SVM algorithm can't maximzing the margin by increasing θ arbitrarily.

Hint: Recall the distinction between functional margin and geometric margin.

2. **[10 points] Model Calibration**

In this question we will try to understand the output $h_\theta(x)$ of the hypothesis function of a logistic regression model, in particular why we might treat the output as a probability (besides the fact that the sigmoid function ensures $h_\theta(x)$ always lies in the interval $(0, 1)$).

When the probabilities outputted by a model match empirical observation, the model is said to be *well calibrated* (or reliable). For example, if we consider a set of examples $x^{(i)}$ for which $h_\theta(x^{(i)}) \approx 0.7$, around 70% of those examples should have positive labels. In a well calibrated model, this property will hold true at every probability value.

Logistic regression tends to output well calibrated probabilities (this is often not true with other classifiers such as Naive Bayes, or SVMs). We will dig a little deeper in order to understand why this is the case, and find that the structure of the loss function explains this property.

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 logistic regression model. In order for the model to be considered well calibrated, given any range of probabilities (a, b) such that $0 \leq a < b \leq 1$, and training 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, the following property must hold:

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

where $P(y = 1|x; \theta) = h_\theta(x) = 1/(1 + \exp(-\theta^\top x))$, $I_{a,b} = \{i | i \in \{1, ..., m\}, h_\theta(x^{(i)}) \in (a, b)\}$ is an index set of all training examples $x^{(i)}$ where $h_\theta(x^{(i)}) \in (a, b)$, and $|S|$ denotes the size of the set S.

(a) [5 points] Show that the above property holds true for the described logistic regression model over the range $(a, b) = (0, 1)$.

Hint: Use the fact that we include a bias term.

Answer: We get the parameter by maximizing the log-likelihood. So, we calculate the derivative of the log-likelihood and set it to zero.

$$\frac{\partial \sum (y \log h_\theta(x) + (1 - y) \log(1 - h_\theta(x)))}{\partial \theta} = \sum (y - h_\theta(x)) x$$

$$\sum (y - h_\theta(x)) \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} = 0$$

consider the first row, we can get

$$(\sum (y - h_\theta(x))) \cdot 1 = 0$$

Thus, by rewriting y and $h_\theta(x)$ as $\mathbb{I}\{y^{(i)} = 1\}$ and $P(y^{(i)} = 1|x^{(i)}; \theta)$ respectively, we can get $\sum_{i \in I_{0,1}} P(y^{(i)} = 1|x^{(i)}; \theta) = \sum_{i \in I_{0,1}} \mathbb{I}\{y^{(i)} = 1\}$. Then, we divide both side by $|\{i \in I_{0,1}\}|$,

which is $|S|$:

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

(b) [3 points] 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.

Answer: At first, I think the description about well calibrated model is not accurate. If we choose an interval (a, b) whose $\{i \in I_{a,b}\}$ only has positive examples(or negative samples). It's impossible for $\frac{\sum_{i \in I_{a,b}} P(y^{(i)}=1|x^{(i)};\theta)}{|\{i \in I_{a,b}\}|} = \frac{\sum_{i \in I_{a,b}} \mathbb{I}\{y^{(i)}=1\}}{|\{i \in I_{a,b}\}|}$ to hold true, because $P(y^{(i)} = 1|x^{(i)}; \theta)$ can't be 1(or 0). Thus, no model is perfectly calibrated.

For a binary classification model, we should only consider intervals whose $\{i \in I_{a,b}\}$ has both positive and negative examples. If we consider a set S of only one positive example and only one negative example, it's clear that a model that is perfectly calibrated doesn't imply that the model achieves perfect accuracy. For example, the model that outputs 0.8 for the negative example and 0.2 for the positive example is well calibrated, but it doesn't achieve perfect accuracy.

Conversely, perfect accuracy doesn't lead to perfect calibration. We consider a train set of one positive example and one negative example again. If the model we got by training outputs 0.6 for the positive one and 0.1 for the negative one, the model achieves perfect accuracy. However, the property doesn't hold true.

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

Answer: By add an regularization term, the loss function becomes

$$J(\theta) = \lambda ||\theta||^2 + \sum (y \log h_\theta(x) + (1 - y) \log(1 - h_\theta(x)))$$

The derivative of the loss function becomes

$$J'(\theta) = 2\lambda\theta + \sum (y - h_\theta(x))x$$

set the derivative to zero

$$2\lambda \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \dots \\ \theta_n \end{bmatrix} + \sum (y - h_\theta(x)) \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = 0$$

consider the first row

$$2\lambda\theta_0 + \sum (y - h_\theta(x)) = 0$$
$$\sum h_\theta(x) = 2\lambda\theta_0 + \sum \mathbb{I}\{y = 1\}$$

So, the prediction is biased by a constant $2\lambda\theta_0$

Remark: We considered the range $(a, b) = (0, 1)$. This is the only range for which logistic regression is guaranteed to be calibrated on the training set. When the GLM modeling assumptions hold, all ranges $(a, b) \subset [0, 1]$ are well calibrated. In addition, when the training and test set 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. This makes logistic regression a very popular model in practice, especially when we are interested in the level of uncertainty in the model output.

3. **[20 points] Bayesian Interpretation of Regularization**

Background: In Bayesian statistics, almost every quantity is a random variable, which can either be observed or unobserved. For instance, parameters θ are generally unobserved random variables, and data x and y are observed random variables. The joint distribution of all the random variables is also called the *model* (*e.g.,* $p(x, y, \theta)$). Every unknown quantity can be estimated by conditioning the model on all the observed quantities. Such a conditional distribution over the unobserved random variables, conditioned on the observed random variables, is called the *posterior distribution*. For instance $p(\theta|x, y)$ is the posterior distribution in the machine learning context. A consequence of this approach is that we are required to endow our model parameters, *i.e.,* $p(\theta)$, with a *prior distribution*. The prior probabilities are to be assigned *before* we see the data—they capture our prior beliefs of what the model parameters might be before observing any evidence.

In the purest Bayesian interpretation, we are required to keep the entire posterior distribution over the parameters all the way until prediction, to come up with the *posterior predictive distribution*, and the final prediction will be the expected value of the posterior predictive distribution. However in most situations, this is computationally very expensive, and we settle for a compromise that is *less pure* (in the Bayesian sense).

The compromise is to estimate a point value of the parameters (instead of the full distribution) which is the mode of the posterior distribution. Estimating the mode of the posterior distribution is also called *maximum a posteriori estimation* (MAP). That is,

$$\theta_{\text{MAP}} = \arg\max_{\theta} p(\theta|x, y).$$

Compare this to the *maximum likelihood estimation* (MLE) we have seen previously:

$$\theta_{\text{MLE}} = \arg\max_{\theta} p(y|x, \theta).$$

In this problem, we explore the connection between MAP estimation, and common regularization techniques that are applied with MLE estimation. In particular, you will show how the choice of prior distribution over θ (*e.g.,* Gaussian or Laplace prior) is equivalent to different kinds of regularization (*e.g.,* L_2, or L_1 regularization). To show this, we shall proceed step by step, showing intermediate steps.

(a) [3 points] Show that $\theta_{\text{MAP}} = \text{argmax}_\theta\, p(y|x, \theta)p(\theta)$ if we assume that $p(\theta) = p(\theta|x)$. The assumption that $p(\theta) = p(\theta|x)$ will be valid for models such as linear regression where the input x are not explicitly modeled by θ. (Note that this means x and θ are marginally independent, but not conditionally independent when y is given.)

Answer:

By the assumption $P(\theta) = P(\theta|x)$, we know $P(\theta, x) = P(\theta)P(x)$ and $P(x|\theta) = P(x)$

$$\begin{aligned}
\theta_{MAP} &= \arg\max_{\theta} p(\theta|x, y) \\
&= \arg\max_{\theta} p(x, y|\theta)p(\theta) \\
&= \arg\max_{\theta} p(y|x, \theta)p(x|\theta)p(\theta) \\
&= \arg\max_{\theta} p(y|x, \theta)p(x)p(\theta) \\
&= \arg\max_{\theta} p(y|x, \theta)p(\theta)
\end{aligned}$$

(b) [5 points] Recall that L_2 regularization penalizes the L_2 norm of the parameters while minimizing the loss (*i.e.,* negative 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 applying L_2 regularization with MLE estimation. Specifically, show that

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

Also, what is the value of λ?

Answer:

the negative log-likelihood of MAP is

$$-\log p(y|x,\theta) - \log p(\theta)$$

$$\begin{aligned}
\theta_{MAP} &= \arg\min_{\theta} -\log p(y|x,\theta) - \log(\frac{1}{(2\pi)^{\frac{n}{2}}\eta^n} exp(-\frac{1}{2}\theta^T \frac{I}{\eta^2}(\theta)) \\
&= \arg\min_{\theta} -\log p(y|x,\theta) + \frac{1}{2}\theta^T \frac{I}{\eta^2}\theta \\
&= \arg\min_{\theta} -\log p(y|x,\theta) + \frac{1}{2\eta^2}\sum_i \theta_i^2
\end{aligned}$$

Thus, that MAP estimation with a zero-mean Gaussian prior over θ, specifically $\theta \sim \mathcal{N}(0, \eta^2 I)$, is equivalent to applying L_2 regularization with MLE estimation. Also, the value of λ is $\frac{1}{2\eta^2}$

(c) [7 points] 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 a closed form expression for θ_{MAP}.

Answer:

$$\begin{aligned}
\theta_{MAP} &= \arg\max_{\theta} \log p(y|x,\theta) + \log p(\theta) \\
&= \arg\max_{\theta} -\frac{1}{2\sigma^2}\sum(y - \theta^T x)^2 - \frac{1}{2\eta^2}\theta^T\theta \\
&= \arg\max_{\theta} -\frac{1}{2\sigma^2}(\vec{y} - X\theta)^T(\vec{y} - X\theta) - \frac{1}{2\eta^2}\theta^T\theta \\
&= \arg\max_{\theta} tr(-\frac{1}{2\sigma^2}(\vec{y}^T\vec{y} - 2\vec{y}^T X\theta + \theta^T X^T X\theta) - \frac{1}{2\eta^2}\theta^T\theta)
\end{aligned}$$

We calculate the derivative of the above function (without $\arg\max$ operation) and set the derivative to zero.

$$\begin{aligned}
&\frac{\partial \log p(y|x,\theta) + \log p(\theta)}{\partial\theta} = \frac{1}{\sigma^2}(X^T\vec{y} - X^T X\theta) - \frac{1}{\eta^2}\theta \\
&\frac{1}{\eta^2}\theta + \frac{1}{\sigma^2}X^T X\theta = \frac{1}{\sigma^2}X^T\vec{y} \\
&\theta = (X^T X + \frac{\sigma^2}{\eta^2}I)^{-1}X^T\vec{y}
\end{aligned}$$

(d) [5 points] Next, consider the Laplace distribution, whose density is given by

$$f_{\mathcal{L}}(z|\mu, b) = \frac{1}{2b} \exp\left(-\frac{|z-\mu|}{b}\right).$$

As before, consider a linear regression model given by $y = x^T\theta + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Assume a Laplace prior on this model, where each parameter θ_i is marginally independent, and is distributed as $\theta_i \sim \mathcal{L}(0, b)$.

Show that θ_{MAP} in this case is equivalent to the solution of linear regression with L_1 regularization, whose loss is specified as

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

Also, what is the value of γ?

Answer: the negative log-likelihood of MAP is

$$-\log p(y|x, \theta) - \log p(\theta)$$

$$\begin{aligned}
\theta_{MAP} &= \arg\min_\theta \frac{1}{2\sigma^2}(\vec{y} - X\theta)^T(\vec{y} - X\theta) + \sum_i \frac{|\theta_i|}{b} \\
&= \arg\min_\theta (\vec{y} - X\theta)^T(\vec{y} - X\theta) + \frac{2\sigma^2}{b}\sum_i |\theta_i| \\
&= \arg\min_\theta ||\vec{y} - X\theta||_2^2 + \frac{2\sigma^2}{b}||\theta||_1
\end{aligned}$$

So, that θ_{MAP} in this case is equivalent to the solution of linear regression with L_1 regularization. Also, $\gamma = \frac{2\sigma^2}{b}$

Note: A closed form solution for linear regression problem with L_1 regularization does not exist. To optimize this, we use gradient descent with a random initialization and solve it numerically.

Remark: Linear regression with L_2 regularization is also commonly called *Ridge regression*, and when L_1 regularization is employed, is commonly called *Lasso regression*. These regularizations can be applied to any Generalized Linear models just as above (by replacing $\log p(y|x, \theta)$ with the appropriate family likelihood). Regularization techniques of the above type are also called *weight decay*, and *shrinkage*. The Gaussian and Laplace priors encourage the parameter values to be closer to their mean (*i.e.,* zero), which results in the shrinkage effect.

Remark: Lasso regression (*i.e.,* L_1 regularization) is known to result in sparse parameters, where most of the parameter values are zero, with only some of them non-zero.

4. **[18 points] Constructing kernels**

In class, we saw that by choosing a kernel $K(x, z) = \phi(x)^T\phi(z)$, we can implicitly map data to a high dimensional space, and have the SVM algorithm work in that space. One way to generate kernels is to explicitly define the mapping ϕ to a higher dimensional space, and then work out the corresponding K.

However in this question we are interested in direct construction of kernels. I.e., suppose we have a function $K(x, z)$ that we think gives an appropriate similarity measure for our learning problem, and we are considering plugging K into the SVM as the kernel function. However for $K(x, z)$ to be a valid kernel, it must correspond to an inner product in some higher dimensional space resulting from some feature mapping ϕ. Mercer's theorem tells us that $K(x, z)$ is a (Mercer) kernel if and only if for any finite set $\{x^{(1)}, \ldots, x^{(m)}\}$, the square matrix $K \in \mathbb{R}^{m\times m}$ whose entries are given by $K_{ij} = K(x^{(i)}, x^{(j)})$ is symmetric and positive semidefinite. You can find more details about Mercer's theorem in the notes, though the description above is sufficient for this problem.

Now here comes the question: Let K_1, K_2 be kernels over $\mathbb{R}^n \times \mathbb{R}^n$, let $a \in \mathbb{R}^+$ be a positive real number, let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be a real-valued function, let $\phi : \mathbb{R}^n \to \mathbb{R}^d$ be a function 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 $p(x)$ a polynomial over x with *positive* coefficients.

For each of the functions K below, state whether it is necessarily a kernel. If you think it is, prove it; if you think it isn't, give a counter-example.

(a) [1 points] $K(x, z) = K_1(x, z) + K_2(x, z)$

(b) [1 points] $K(x, z) = K_1(x, z) - K_2(x, z)$

(c) [1 points] $K(x, z) = aK_1(x, z)$

(d) [1 points] $K(x, z) = -aK_1(x, z)$

(e) [5 points] $K(x, z) = K_1(x, z)K_2(x, z)$

(f) [3 points] $K(x, z) = f(x)f(z)$

(g) [3 points] $K(x, z) = K_3(\phi(x), \phi(z))$

(h) [3 points] $K(x, z) = p(K_1(x, z))$

[**Hint:** For part (e), the answer is that K *is* indeed a kernel. You still have to prove it, though. (This one may be harder than the rest.) This result may also be useful for another part of the problem.]

Answer:

Let's take a finite set $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$, and we will build kernel matrix on this data set.

(a) $K(x, z) = K_1(x, z) + K_2(x, z)$

We already know K_1 and K_2 are kernels. So the kernel matrices K_1 and K_2 are symmetric and positive semidefinite. For the i th and j th examples in the data set, it's easy to see $K_{ij} = K_{1ij} + K_{2ij}$. So, for kernel matrices, $K = K_1 + K_2$.

i) Because $K_1 = K_1^T$ and $K_2 = K_2^T$, we can know $K^T = (K_1 + K_2)^T = K_1^T + K_2^T = K_1 + K_2 = K$. So, the kernel matrix K is symmetric.

ii) Because $\forall z \in \mathbb{R}^n, z^T K_1 z \geq 0$ *and* $z^T K_2 z \geq 0$, we can get $\forall z \in \mathbb{R}^n, z^T K z = z^T(K_1 + K_2)z = z^T K_1 z + z^T K_2 z \geq 0$. So, the kernel matrix K is positive semidefinite.

Based on i) and ii), we can know the kernel matrix K is symmetric and positive semidefinite, the kernel function K, it is **necessarily** a kernel.

(b) $K(x,z) = K_1(x,z) - K_2(x,z)$
It's easy to see, for kernel matrices, $K = K_1 - K_2$.
i) Because $\forall z \in \mathbb{R}^n, z^T K_1 z \geq 0 \; and \; z^T K_2 z \geq 0$, we can get $\forall z \in \mathbb{R}^n, z^T K z = z^T (K_1 - K_2) z = z^T K_1 z - z^T K_2 z$.
So, the kernel function K is **not necessarily** a kernel.
Counter Example:
$K_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, K_2 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, K = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$. K is not positive semidefinite.

(c) $K(x,z) = aK_1(x,z)$
It's easy to see, for kernel matrices, $K = aK_1$.
i) Because $K_1 = K_1^T$, we can know $K^T = (aK_1)^T = aK_1^T = aK_1 = K$. So, the kernel matrix K is symmetric.
ii) Because $\forall z \in \mathbb{R}^n, z^T K_1 z \geq 0$, we can get $\forall z \in \mathbb{R}^n, z^T K z = z^T (aK_1) z = a(z^T K_1 z) \geq 0$.
So, the kernel matrix K is positive semidefinite.
Based on i) and ii), we can know the kernel matrix K is symmetric and positive semidefinite, the kernel function K, it is **necessarily** a kernel.

(d) $K(x,z) = -aK_1(x,z)$
It's easy to see, for kernel matrices, $K = aK_1$.
i) Because $\forall z \in \mathbb{R}^n, z^T K_1 z \geq 0$, we can get $\forall z \in \mathbb{R}^n, z^T K z = z^T (aK_1) z = a(z^T K_1 z) \leq 0$.
So, the kernel matrix K is not positive semidefinite. Of course, the kernel function K is **not necessarily** a kernel.
Counter Example:
$K_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, a = 1, K = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$. K is not positive semidefinite.

(e) $K(x,z) = K_1(x,z)K_2(x,z)$
It's easy to see, for kernel matrices, $K = K_1 \circ K_2$. $\circ$ is Hadamard multiplication. Let's assume that $K_1(x,y) = \phi_1(x)\phi_1(y), K_2(x,y) = \phi_2(x)\phi_2(y)$
i) Because $K_1 = K_1^T, K_2 = K_2^T$, we can know $K^T = (K_1 \circ K_2)^T = K_1^T \circ K_2^T = K_1 \circ K_2 = K$.
So, the kernel matrix K is symmetric.
ii) Because $\forall z \in \mathbb{R}^n, z^T K_1 z \geq 0 \; and \; z^T K_2 z \geq 0$, we can get

$$\begin{aligned}
\forall z \in \mathbb{R}^n, z^T K z &= z^T (K_1 \circ K_2) z \\
&= \sum_i \sum_j z_i \phi_1(x_i) \phi_1(x_j) \phi_2(x_i) \phi_2(x_j) z_j \\
&= \sum_i \sum_j z_i (\sum_k \phi_1(x_i)_k \phi_1(x_j)_k)(\sum_p \phi_2(x_i)_p \phi_2(x_j)_p) z_j \\
&= \sum_k \sum_p \sum_i \sum_j z_i \phi_1(x_i)_k \phi_1(x_j)_k \phi_2(x_i)_p \phi_2(x_j)_p z_j \\
&= \sum_k \sum_p \sum_i z_i \phi_1(x_i)_k \phi_2(x_i)_p \sum_j z_j \phi_1(x_j)_k \phi_2(x_j)_p \\
&= \sum_k \sum_p (\sum_i z_i \phi_1(x_i)_k \phi_2(x_i)_p)^2 \geq 0
\end{aligned}$$

So, the kernel matrix K is positive semidefinite.
Based on i) and ii), we can know the kernel matrix K is symmetric and positive semidefinite, the kernel function K, it is **necessarily** a kernel.

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

$$K = \begin{bmatrix} f(x_1)f(x_1) & f(x_1)f(x_2) & \dots & f(x_1)f(x_m) \\ f(x_2)f(x_1) & f(x_2)f(x_2) & \dots & f(x_2)f(x_m) \\ \vdots & \vdots & \dots & \vdots \\ f(x_m)f(x_1) & f(x_m)f(x_2) & \dots & f(x_m)f(x_m) \end{bmatrix} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_m) \end{bmatrix} \begin{bmatrix} f(x_1) & f(x_2) & \dots & f(x_m) \end{bmatrix}$$

Let's assume $b = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_m) \end{bmatrix}$. So, $K = bb^T$.

i)$K^T = (bb^T)^T = (b^T)^T b^T = bb^T = K$ So, K is symmetric.

ii)$\forall z \in \mathbb{R}^n, z^T K z = z^T bb^T z = (z^T b)^2 \geq 0$ So, K is positive semidefinite.

Based on i) and ii), K is a valid kernel.

(g) $K(x,z) = K_3(\phi(x), \phi(z))$

The kernel matrix K is equal to K_3. So, K is symmetric and positive semidefinite. K is a valid kernel.

(h) $K(x,z) = p(K_1(x,z))$

Let's assume $p(x) = \sum_k a_k x^k, a_k > 0$ and use $A^{(k)}$ to denote the hadamard product of k matrices A. So, $K = p(K_1) = \sum_k a_k K_1^{(k)}$

Based on the problem e), $K_1^{(k)}$ is a valid kernel.

Based on the problem c) and the fact that $a_k \geq 0$, $a_k K_1^{(k)}$ is a valid kernel.

Based on the problem a), $\sum_k a_k K_1^{(k)}$ is a valid kernel. So, K is a valid kernel.

5. **[16 points] Kernelizing the Perceptron** Let there be a binary classification problem with $y \in \{0, 1\}$. The perceptron uses hypotheses of the form $h_\theta(x) = g(\theta^T x)$, where $g(z) = \text{sign}(z) = 1$ if $z \geq 0$, 0 otherwise. In this problem we will consider a stochastic gradient descent-like implementation of the perceptron algorithm where each update to the parameters θ is made using only one training example. However, unlike stochastic gradient descent, the perceptron algorithm will only make one pass through the entire training set. The update rule for this version of the perceptron algorithm is given by

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

where $\theta^{(i)}$ is the value of the parameters after the algorithm has seen the first i training examples. Prior to seeing any training examples, $\theta^{(0)}$ is initialized to $\vec{0}$.

(a) [9 points] Let K be a Mercer kernel corresponding to some very high-dimensional feature mapping ϕ. Suppose ϕ is so high-dimensional (say, ∞-dimensional) that it's infeasible to ever represent $\phi(x)$ explicitly. Describe how you would apply the "kernel trick" to the perceptron to make it work in the high-dimensional feature space ϕ, but without ever explicitly computing $\phi(x)$.

[**Note:** You don't have to worry about the intercept term. If you like, think of ϕ as having the property that $\phi_0(x) = 1$ so that this is taken care of.] Your description should specify:

i. [3 points] How you will (implicitly) represent the high-dimensional parameter vector $\theta^{(i)}$, including how the initial value $\theta^{(0)} = 0$ is represented (note that $\theta^{(i)}$ is now a vector whose dimension is the same as the feature vectors $\phi(x)$);

ii. [3 points] How you will efficiently make a prediction on a new input $x^{(i+1)}$. I.e., how you will compute $h_{\theta^{(i)}}(x^{(i+1)}) = g({\theta^{(i)}}^T \phi(x^{(i+1)}))$, using your representation of $\theta^{(i)}$; and

iii. [3 points] How you will modify the update rule given above to perform an update to θ on a new training example $(x^{(i+1)}, y^{(i+1)})$; *i.e.,* using the update rule corresponding to the feature mapping ϕ:

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

Answer:

a) The way to represent the high-dimensional parameter vector $\theta^{(i)}$: assume we have m examples, we will represent $\theta^{(i)}$ as a linear combination of the feature vectors of m examples.

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

where $\beta_j = 0$ **if** $j > i$**, otherwise** $\beta_j = \alpha(y^{(j)} - h_{\theta^{(j-1)}}(x^{(j)}))$

The $\theta^{(0)}$ is represented as a linear combination of the feature vectors of examples, where all $\beta_j = 0$

b) The most difficult part of a prediction is how to compute ${\theta^{(i)}}^T \phi(x^{(i+1)})$. We can represent $\theta^{(i)}$ as a linear combination of the feature vectors of first i training examples.

$${\theta^{(i)}}^T \phi(x^{(i+1)}) = (\sum_{k=1}^{i} \beta_k \phi(x^{(k)}))^T \phi(x^{(i+1)}) = \sum_{k=1}^{i} \beta_k K(\phi(x^{(k)}), \phi(x^{(i+1)}))$$

So, we can use kernel K to efficiently predict the label of a new input $x^{(i+1)}$.

c) we represent $\theta^{(i)}$ as a liner combination of $x^{(1)} - x^{(i)}$. So, we only need to use a vector $\lambda^{(i)}$

record $\alpha(y^{(j)} - h_{\theta^{(j-1)}}(x^{(j)}))$ for each $x^{(j)}$, where $j \leq i$.
The update rule is $\lambda^{(i)} = [\lambda^{(i-1)}, \alpha(y^{(i)} - h_{\theta^{(i-1)}}(x^{(i)}))]$. In othe words, we only need append a scalar $\alpha(y^{(i)} - h_{\theta^{(i-1)}}(x^{(i)}))$ at the end of $\lambda^{(i-1)}$ to get $\lambda^{(i)}$

(b) [5 points] Implement your approach by completing the `initial_state`, `predict`, and `update_state` methods of `src/p05_percept.py`.

(c) [2 points] Run `src/p05_percept.py` to train kernelized perceptrons on `data/ds5_train.csv`. The code will then test the perceptron on `data/ds5_test.csv` and save the resulting predictions in the `src/output` folder. Plots will also be saved in `src/output`. We provide two kernels, a dot product kernel and an radial basis function (rbf) kernel. One of the provided kernels performs extremely poorly in classifying the points. Which kernel performs badly and why does it fail?

Answer: Dot product kernel performs extremely badly. For dot product kernel, the mapping function is $\phi(x) = x$ and the original data is not linearly separable. So, we need to map the data set to a higher space.

6. **[22 points] Spam classification**

In this problem, we will use the naive Bayes algorithm and an SVM to build a spam classifier.

In recent years, spam on electronic media has been a growing concern. Here, we'll build a classifier to distinguish between real messages, and spam messages. For this class, we will be building a classifier to detect SMS spam messages. We will be using an SMS spam dataset developed by Tiago A. Almedia and José María Gómez Hidalgo which is publicly available on `http://www.dt.fee.unicamp.br/~tiago/smsspamcollection` [1]

We have split this dataset into training and testing sets and have included them in this assignment as `data/ds6_spam_train.tsv` and `data/ds6_spam_test.tsv`. See `data/ds6_readme.txt` for more details about this dataset. Please refrain from redistributing these dataset files. The goal of this assignment is to build a classifier from scratch that can tell the difference the spam and non-spam messages using the text of the SMS message.

(a) [5 points] Implement code for processing the the spam messages into numpy arrays that can be fed into machine learning models. Do this by completing the `get_words`, `create_dictionary`, and `transform_text` functions within our provided `src/p06_spam.py`. Do note the corresponding comments for each function for instructions on what specific processing is required.

The provided code will then run your functions and save the resulting dictionary into `output/p06_dictionary` and a sample of the resulting training matrix into `output/p06_sample_train_matrix`.

Answer:

(b) [10 points] In this question you are going to implement a naive Bayes classifier for spam classification with multinomial event model and Laplace smoothing (refer to class notes on Naive Bayes for details on Laplace smoothing).

Write your implementation by completing the `fit_naive_bayes_model` and `predict_from_naive_bayes_model` functions in `src/p06_spam.py`.

`src/p06_spam.py` should then be able to train a Naive Bayes model, compute your prediction accuracy and then save your resulting predictions to `output/p06_naive_bayes_predictions`.

Remark. If you implement naive Bayes the straightforward way, you'll find that the computed $p(x|y) = \prod_i p(x_i|y)$ often equals zero. This is because $p(x|y)$, which is the product of many numbers less than one, is a very small number. The standard computer representation of real numbers cannot handle numbers that are too small, and instead rounds them off to zero. (This is called "underflow.") You'll have to find a way to compute Naive Bayes' predicted class labels without explicitly representing very small numbers such as $p(x|y)$. [**Hint:** Think about using logarithms.]

Answer: The algorithm achieves about 0.978 accuracy.

(c) [5 points] Intuitively, some tokens may be particularly indicative of an SMS being in a particular class. We can try to get an informal sense of how indicative token i is for the SPAM class by looking at:

$$\log \frac{p(x_j = i|y = 1)}{p(x_j = i|y = 0)} = \log \left(\frac{P(\text{token } i|\text{email is SPAM})}{P(\text{token } i|\text{email is NOTSPAM})} \right).$$

Complete the `get_top_five_naive_bayes_words` function within the provided code using the above formula in order to obtain the 5 most indicative tokens.

[1] Almeida, T.A., Gómez Hidalgo, J.M., Yamakami, A. Contributions to the Study of SMS Spam Filtering: New Collection and Results. Proceedings of the 2011 ACM Symposium on Document Engineering (DOCENG'11), Mountain View, CA, USA, 2011.

The provided code will print out the resulting indicative tokens and then save thm to `output/p06_top_indicative_words`.

Answer: The top five words are ['claim', 'won', 'prize', 'tone', 'urgent!'] .

(d) [2 points] Support vector machines (SVMs) are an alternative machine learning model that we discussed in class. We have provided you an SVM implementation (using a radial basis function (RBF) kernel) within `src/svm.py` (You should not need to modify that code).

One important part of training an SVM parameterized by an RBF kernel is choosing an appropriate kernel radius.

Complete the `compute_best_svm_radius` by writing code to compute the best SVM radius which maximizes accuracy on the validation dataset.

The provided code will use your `compute_best_svm_radius` to compute and then write the best radius into `output/p06_optimal_radius`.

Answer: The optimal SVM radius was 0.1. The accuracy on the test set is 0.9695

(e) Of the Naive Bayes and RBF SVM model, which performs better? Can you provide a possible explanation for why the best model performs so well?

Answer: The Naive Bayes works better. Maybe the training set is to small, we have about 2 thousand parameters(because we have about 2 thousand words) and only 4 thousand training examples. This small train set may causes that the SVM model over-fitted on the training set.