STANFORD UNIVERSITY
CS 229, Autumn 2018
Midterm Examination Solutions

Question	Points
1 Multiple Choice	/47
2 Neural Networks	/19
3 Naive Bayes	/15
4 Kernels	/36
5 Trees and Random Forests	/26
Total	/143

Name of Student: _____

SUNetID: _____@stanford.edu

The Stanford University Honor Code:

I attest that I have not given or received aid in this examination, and that I have done my share and taken an active part in seeing to it that others as well as myself uphold the spirit and letter of the Honor Code.

Signed: _____

1. **[47 points] Multiple Choice**

(a) [5 points] **Friendly Neighborhood Grocery Store**

Multiple answers may be correct. You need to mark all the correct answers, and leave unmarked all wrong answers to earn points. If any correct answer is left unmarked, or if any wrong answer is marked, you will not earn any points for that sub-question.

A new grocery store has opened in your neighborhood, where the staff are friendly and the produce is fresh. However, the prices of items are a mystery. When you shop here, you are only told the total price of your cart at the very end.

Given your new machine learning knowledge, you want to come up with a way to estimate the prices of individual items at that store. So we go about collecting data. Let's say the store has n distinct types of items. We make a total of m visits to the store, every time shopping a different quantity of each item. We record the final check out price at the end of each grocery trip.

We construct a linear regression model with the following MSE cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(y^{(i)} - \theta^T x^{(i)})^2,$$

where $\theta \in \mathbb{R}^n$ are the model parameters, $x_j^{(i)} \in \mathbb{R}$ corresponds to the number of grocery item j purchased during trip i, and $y^{(i)} \in \mathbb{R}$ is the total cost of grocery shopping in trip i. We train the model with gradient descent until the algorithm converges.

i. [1 points]
We want to know what θ represents. Specifically, what will θ_j be at the end of training?

A. Total stock quantity of item j in the store.

B. Price for one unit of item j.

C. Total number of units of item j we purchased over m trips.

Answer: We want to compute the price of grocery shopping with this linear relation : $y = \theta^T x$. **B** : θ_j represents the price of one item j.

ii. [4 points]
Unfortunately, we neither have enough money (we are students after all) nor enough time (all that CS229 homework!) to create a large grocery shopping dataset. Therefore our dataset is fairly small.
However, for all the items that are available in the store, we have a good guess of how much they might approximately cost. So, to compensate from the lack of data, we decide to give our machine learning algorithm access to this prior knowledge. Let $c \in \mathbb{R}^n$ denote a fixed vector of our guesses of cost of each item in the store. Specifically, we believe item j costs c_j. We want to incorporate this vector (i.e., prior knowledge) into the linear regression

algorithm in a meaningful way. Our approach is to include an additional term in the loss function, such that it regularizes the parameters towards c in the absence of a lot of data. Specifically, we add a term $R(\theta, c)$ into the loss function like this:

$$J(\theta) = R(\theta, c) + \frac{1}{2}\sum_{i=1}^{m}(y^{(i)} - \theta^T x^{(i)})^2$$

Which of the following are reasonable choices for $R(\theta, c)$? Mark all the appropriate choices.

A. $R(\theta, c) = |\theta^T c|$
B. $R(\theta, c) = \frac{1}{2}\|\theta - c\|_2^2$
C. $R(\theta, c) = \|\theta - c\|_1$
D. $R(\theta, c) = \max_{1 \le i \le m} max(0, 1 - (\theta - c)^T x^{(i)})$

Answer: In this setup, θ corresponds to cost of each item. According to your prior knowledge, θ should be close to c. The followings $R(\theta) = \frac{1}{2}\|\theta - c\|_2^2$ and $R(\theta) = \|\theta - c\|_1$ are a sensible choices to enforce this constraint.
Correct answers:**B** ; **C**

(b) [12 points] **Monsters at Yosemite**

Multiple answers may be correct. You need to mark all the correct answers, and leave unmarked all wrong answers to earn points. If any correct answer is left unmarked, or if any wrong answer is marked, you will not earn any points for that sub-question.

The National Park Service (NPS) is in need of a Machine Learning expert. They find out that you did well in the CS229 midterm, and reach out to you for help. In your first meeting, they describe their problem: The campgrounds of Yosemite national park have recently had attacks by Bigfoots[1]. So the NPS has installed video cameras in the park to spot them. Now, they need your help to create a machine learning model to detect Bigfoots from footage captured by these cameras, so that they can set up an automatic alert system for the park rangers.

i. [4 points] They provide you with the following dataset. 1,000 images (still video frames) captured by the NPS cameras. Only 10 of these images are labelled as containing Bigfoots ($y = 1$) (they are sneaky, and hard to capture on film!), and the other 990 are labelled as not containing Bigfoots ($y = 0$). This is a rather small dataset for image classification. Which of the following approaches are ways of increasing our data set size that will improve our model performances?

A. Include images of hand-drawn pictures of Bigfoots.

B. Collect some pictures of bears, and add them into your training set by labelling them as Bigfoots.

C. Use data augmentation techniques such as picture mirroring (flipping the image left/right).

D. Take more pictures of Bigfoots in Yosemite with the NPS cameras.

Answer: To create more examples, you can either take more video frames of Bigfoots in Yosemite (this takes a lot of time) or use data augmentation techniques to artificially increase your number of examples: rotate your original images, flip them, crop them... Adding drawings won't help as this images come from a very different distribution from your video frames. Adding pictures of bears labeled as Bigfoots is of course bad as the algorithm will also learn to detect bears.
Correct answers: **C** ; **D**

ii. [2 points] To add more examples, the NPS gives you this additional dataset: 1,000,000 Instagram pictures with amazing looking filters taken by Yosemite tourists, with 10,000 of them containing Bigfoots. The NPS camera and your algorithm does not have access to these image filters. But we still want to make the best use of the given images.
We assume that both the video frames and the Instagram pictures have the same resolution: 32 by 32 pixels. Each RGB pixel is encoded with 3 values.

[1] `https://en.wikipedia.org/wiki/Bigfoot`, not an honor code violation to click.

Which one of the following approaches would you use to create a train / dev (validation) / test splits from all the images (NPS camera and Instagram combined)?

A. Shuffle the combined datasets, then split it between your train / dev / test set as 60% / 20% / 20%.

B. Shuffle the combined datasets, then split it between your train / dev / test set as 98% / 1% / 1%.

C. Use all the Instagram pictures along with a third of the NPS camera images as your train set, then divide the remaining NPS camera images into a dev and test set.

D. Use 98% of the Instagram pictures as your train set, 2% of the Instagram pictures as your dev set and all the video frames as your test set.

Answer: With this additional dataset, you now have a fair amount of examples. If all your images came from the same source, you would split your dataset between your train / dev / test set as 98% / 1% / 1%. However in this case the Instagram pictures are different from you real target : NPS video frames. So to aim for the right target, both your dev and test set need to be composed of NPS video frames. Finally, your train set should contain some NPS video frames as well. You use all the Instagram pictures with a third of the video frames as your train set, then divide the remaining video frames between your dev and test set.
Correct answer: **C**

iii. [2 points] To solve your classification problem, you want to train a 3-layer Neural Network with the following architecture:

- 5 neurons in the first hidden layer
- 3 neurons in the second hidden layer
- 1 neuron in the output layer

Remember that the size of the picture is 32 by 32 pixels. Each pixel is encoded with 3 values (R, G and B). The input x of your Neural Network is a flattened version of a picture of this size. You train your Neural Network with batches of 16 examples at a time.
How many parameters does the Neural Network have?

A. 123,024

B. 15,378

C. 15,387

D. 123,096

Answer: Your image is flattened into a input column vector of size $n_x = 32 * 32 * 3$. Then the first hidden layer outputs an activation of size $n^{[1]} = 5$, the second hidden layer outputs an activation of size $n^{[2]} = 3$. The output layer outputs an activation of size $n^{[3]} = 1$. The parameter of a layer l are the weight matrix $W^{[l]}$ of shape $(n^{[l]}, n^{[l-1]})$ and the bias term of shape $(n^{[l]}, 1)$. So in total, we have : $\sum_{i=1}^{L} n^{[l]}(n^{[l-1]}+1) = (32*32*3+1)*5+(5+1)*3+(3+1) = 15,387$ parameters

Correct answer: **C**

iv. [1 points] What should be the activation function of the output layer?

A. ReLU

B. Softmax

C. Sigmoid

D. Tanh

Answer: This is a binary classification problem : your labels are either $1 =$ 'a Bigfoot on the picture' or $0 =$ 'no Bigfoot on the picture'. You use the sigmoid function as your activation function.
Tanh is also accepted as we did not precise in an early version if the labels were $0, 1$ or $-1, 1$.
Correct answer: **C** ; **D**

v. [2 points] You choose to minimize the log-loss function.
After training, your model achieves 99% accuracy on the dev set. But after looking at your dev set prediction, you find out that all your examples were predicted as negative (your algorithm predicted that they were no Bigfoots on any of the dev set images)!
What is the most likely cause of this phenomenon? (Select only ONE most likely answer)

A. Your image resolution is too small.

B. You have a class imbalance problem (there are many more examples of one class than the other).

C. Your minibatch size is too small.

D. Your network's hidden layers are too small and too few.

Answer: 99% of your examples are negative examples. This is called a 'class imbalance' problem, where one class is much more predominant in your dataset.
In this setup, a naive classifier outputting always 0 will get 99% accuracy but will never detect a Bigfoot (0% recall).
Correct answer: **B**

vi. [1 points] So finally, using all the machine learning skills from CS229, you debug all the problems, and have a model that detects Bigfoots really well. The NPS deploys the model into production, and things are going great. Your algorithm alerts the park rangers automatically, and the campers are happy.
After a few weeks, the NPS calls you again frantically. This time they have discovered that Abominable Snowmen[2] have migrated from the Himalayas to Yosemite! Following your impressive work on detecting Bigfoots, they now want you to extend your model's capabilities to also detect Abominable Snowmen in the NPS camera feeds. You add another neuron in the last layer to correspond to the Abominable Snowman.

[2] `https://en.wikipedia.org/wiki/Yeti`, again not an honor code violation to read it.

Then you find out that Abominable Snowmen and Bigfoots avoid each other so much, that it is impossible to capture both of them in the same picture. What would you change the output activation function of your Neural Network to?

A. ReLU
B. Softmax
C. Sigmoid
D. Tanh

Answer: This is a multiclass classification problem: your label is a 'one-hot' encoded vector corresponding to the number of the class present on the photo. You use the softmax function as your activation function.
Correct answer: **B**

Your changes to the model work well, and the updated model is able to detect Abominable Snowmen and Bigfoots reliably from the NPS camera feeds. After several months of successful operation, thanks to the alert system based on your model, the park rangers find out that all this while it was just a group of mischievous tourists dressing up as Bigfoots and Abominable snowmen. Perhaps the real ones are still out there...

(c) [6 points] **Bias-Variance in Supervised Learning**

You need to mark exactly one answer corresponding to bias and exactly one answer corresponding to variance for each of the sub-questions below. Both need to be correct in order for you to earn any points for that sub-question.

We consider a regression task where we have m training examples and our hypothesis $h_\theta(x)$ is parameterized by θ. For each of the following scenarios, select whether we should expect bias and variance to increase or decrease.

i. [2 points] Replace the value of your last feature by a random number for every example of your dataset.

A. Bias increases.
B. Bias decreases.
C. Uncertain effect on bias.
D. Variance increases.
E. Variance decreases.
F. Uncertain effect on variance.

Answer: Correct answers :

A Bias should increase as you are replacing information that could be useful for your prediction with a meaningless feature.

D Variance should also increase as your model will try to fit this noise on the train set.

ii. [2 points] Map $x \in \mathbb{R}^n$ to $\phi(x) \in \mathbb{R}^d$ where $d > n$, $\phi(x)_i = x_i$ for $1 \leq i \leq n$, and $\phi(x)_i = f_i(x)$ for $n < i \leq d$ where each f_i is some scalar valued, deterministic function over the initial set of features. None of the f_i's are just constant values that ignore the input.

A. Bias increases.
B. Bias decreases.
C. Uncertain effect on bias.
D. Variance increases.
E. Variance decreases.
F. Uncertain effect on variance.

Answer: Correct answers :

B Bias should decrease since we are adding new features to the examples: the hypothesis space of the model grows.

D Variance should increase since we are adding new features to the examples: the hypothesis space of the model grows.

iii. [2 points] Stop your optimization algorithm early.

A. Bias increases.
B. Bias decreases.
C. Uncertain effect on bias.
D. Variance increases.

E. Variance decreases.

F. Uncertain effect on variance.

Answer: Correct answers :

A Bias should increase since your optimization algorithm has less iterations to minimize the cost function.

E Variance should decrease since you prevent your algorithm from tweaking the parameter θ to minimize your cost function as much as possible, hence avoiding overfitting.

(d) **[6 points] Bias-Variance in Unsupervised Learning**

You need to mark exactly one answer corresponding to bias and exactly one answer corresponding to variance for each of the sub-questions below. Both need to be correct in order for you to earn any points for that sub-question.

In a Gaussian Mixture Model (GMM), the data is modeled by specifying its joint distribution over data points $x^{(i)}$ and latent cluster identities $z^{(i)}$,

$$p(x^{(i)}, z^{(i)}) = p(x^{(i)}|z^{(i)})p(z^{(i)}),$$

where,

$$z^{(i)} \sim \text{Multinomial}(\phi), \left(\phi_j \geq 0, \sum_{j=1}^{k} \phi_j = 1\right)$$

and

$$(x^{(i)}|z^{(i)} = j) \sim \mathcal{N}(\mu_j, \Sigma_j).$$

Let k denote the number of values that the $z^{(i)}$'s can take on, m denote the number of data points or the number of values i can take and d to be the dimension of the data point $x^{(i)}$.

Select how Bias and Variance are each affected by *increasing* the value of the following variables, keeping all else fixed.

i. [2 points] Increase d.

A. Bias increases.
B. Bias decreases.
C. Uncertain effect on bias.
D. Variance increases.
E. Variance decreases.
F. Uncertain effect on variance.

Answer: Correct answers :

B Bias should decrease since we are increasing the dimension of our features: the hypothesis space of the model grows.

D Variance should increase since we are increasing the dimension of our features: from the curse of dimensionality, as dimensions increase we are more likely to 'overfit'.

ii. [2 points] Increase k.

A. Bias increases.
B. Bias decreases.
C. Uncertain effect on bias.
D. Variance increases.
E. Variance decreases.
F. Uncertain effect on variance.

Answer: Correct answers : (As intuition, first set $k = 1$ and then $k = m$)

B Bias should decrease since using many components in the mixture lets us fit many distributions very accurately.

D Variance should increase since we have more parameters, more degrees of freedom and so we can't fit any one of them as precisely.

iii. [2 points] Increase m.

A. Bias increases.

B. Bias decreases.

C. Uncertain effect on bias.

D. Variance increases.

E. Variance decreases.

F. Uncertain effect on variance.

Answer: Correct answers :

C Bias uncertain : the effect of adding new data has an uncertain effect on bias.

E Variance should decrease since there is lesser variability fitting of in the parameters. Your model see more diverse examples, hence avoiding overfitting.

(e) [8 points] **True or False**

To discourage random guessing, 1 point will be deducted for each wrong answer.

Provide a short explanation (one short sentance, or two max) along with each True/False answer.

A. Generalized Linear Models (GLMs) *always* have a unique well defined optimum parameter.

B. For decision trees, when the number of leaves increases, both bias and variance decreases.

C. All two leaved classification trees (or decision stubs) are linear classifiers.

D. The classification boundaries for a support vector machine with hard boundaries depends only on a small subset of the data.

E. If we perform Bagging multiple logistic regression models instead of decision trees, the resulting model still has a linear decision boundary, but one that cannot be recovered by directly training as a single model.

F. A 7 layered neural network classifier with identity activation function for the hidden layers (i.e, $f(x) = x$), and a single output layer with sigmoid activation, has the same representational power as a linear classifier.

G. Any trained Kernelized SVM model can be re-expressed as a Kernelized Perceptron.

H. The probability distribution having PDF

$$p(x;\theta) = \frac{\pi}{\theta}\left(\frac{x}{\theta}\right)^{\pi-1} \exp\left\{-\left(\frac{x}{\theta}\right)^{k}\right\}$$

belongs to the exponential family.

Answer:

A. **False:** A positive semi definite hessian does not mean there should be a well defined minima. Gradient can be asymptotically zero like in the case of linearly separable logistic regression. In such cases, minimas dont exist.

B. **False:** For trees when leaf size decreases, variance increases and bias decreases. As the leaf size decreases we could have a variety of valid trees with different decision boundaries trained on the data.

C. **True:** Two leafed classification trees are linear models since they have only one decision node and a decision node for vanilla trees is a linear boundary.

D. **True:** The classification boundaries for a support vector machine with hard boundaries depends only on the support vectors which are closest to the decision boundary.

E. **False:** Consider logistic regression as a counter example. For any bagged linear classifier model you have:

$$\sum_{i=1}^{n} \alpha_i f(\theta_i^\top X) = c$$

This is not a linear decision boundary.

F. **True:** When there are no non linear activations in the hidden layers. Since the composite of linear functions is a linear function, $A \times Bx$ is a linear function for matrices A and B. The neural network classification can be expressed as:

$$\sigma(W^\top X + b)$$

This is the same function as the one for logistic regression and is a linear classifier.

G. **True:** The prediction functions of Kernelized SVM and Kernelized Perceptron are identical (with a minor change in $+1$/-1 vs $0/1$ convention).

H. **True:** It is a Weibull distribution with fixed shape parameter π.

(f) **[10 points] Lower Bounds**

To discourage random guessing, 1 point will be deducted for each wrong answer.

In EM, we maximize $p(x;\theta)$ indirectly by maximizing it's lower bound in the M-step. Which of the following are other valid lower bounds of $p(x;\theta)$ (though not all of them might be *useful*)?

A. $\log p(x;\theta)$.
B. $p(x,z;\theta)$ for any given value of $z = \hat{z}$.
C. $p(x|z;\theta)$ for any given value of $z = \hat{z}$.
D. $p(z|x;\theta)$ for any given value of $z = \hat{z}$.
E. $\log p(x|z;\theta)$ for any given value of $z = \hat{z}$.
F. $\log p(z|x;\theta)$ for any given value of $z = \hat{z}$.
G. $\mathbb{E}_{z\sim p(z|x;\theta)}\left[\log \frac{p(x,z;\theta)}{p(z|x;\theta)}\right]$.
H. $\mathbb{E}_{z\sim p(z;\theta)}\left[\log \frac{p(x,z;\theta)}{p(z;\theta)}\right]$.
I. $\mathbb{E}_{z\sim p(z|x;\theta)}\left[\log p(x,z;\theta)\right]$.
J. $\mathbb{E}_{z\sim p(z;\theta)}\left[\log p(x,z;\theta)\right]$.

Answer:

A. $\log p(x;\theta)$
Answer: YES. $\log(z) < z$ for all $z > 0$.

B. $p(x,z;\theta)$ for any given value of $z = \hat{z}$.
Answer: YES.

$$\begin{aligned} p(x) &= \sum_z p(x,z) \\ &= p(x,\hat{z}) + \sum_{z\neq\hat{z}} p(x,z) \\ &\geq p(x,\hat{z}). \end{aligned}$$

C. $p(x|z;\theta)$ for any given value of $z = \hat{z}$.
Answer: NO.

D. $p(z|x;\theta)$ for any given value of $z = \hat{z}$.
Answer: NO.

E. $\log p(x|z;\theta)$ for any given value of $z = \hat{z}$
Answer: YES. $p(\cdot) \geq 0$, and $\log p(\cdot) \leq 0$ always.

F. $\log p(z|x;\theta)$ for any given value of $z = \hat{z}$
Answer: YES. $p(\cdot) \geq 0$, and $\log p(\cdot) \leq 0$ always.

G. $\mathbb{E}_{z\sim p(z|x;\theta)}\left[\log \frac{p(x,z;\theta)}{p(z|x;\theta)}\right]$
Answer: YES.

$$\begin{aligned} p(x) &> \log p(x) \\ &\geq \mathbb{E}_{z\sim p(z|x;\theta)}\left[\log \frac{p(x,z;\theta)}{p(z|x;\theta)}\right] \quad \text{(EM notes)} \end{aligned}$$

H. $\mathbb{E}_{z\sim p(z;\theta)}\left[\log \frac{p(x,z;\theta)}{p(z;\theta)}\right]$
Answer: YES.

$$\begin{aligned} p(x) &> \log p(x) \\ &\geq \mathbb{E}_{z\sim p(z;\theta)}\left[\log \frac{p(x,z;\theta)}{p(z;\theta)}\right] \quad \text{(EM notes)} \end{aligned}$$

I. $\mathbb{E}_{z\sim p(z|x;\theta)}\left[\log p(x,z;\theta)\right]$
Answer: YES.

$$\begin{aligned} p(x) &> \log p(x) \\ &\geq \mathbb{E}_{z\sim p(z|x;\theta)}\left[\log \frac{p(x,z;\theta)}{p(z|x;\theta)}\right] \\ &= \mathbb{E}_{z\sim p(z|x;\theta)}\left[\log p(x,z;\theta)\right] - \mathbb{E}_{z\sim p(z|x;\theta)}\left[\log p(z|x;\theta)\right] \\ &= \mathbb{E}_{z\sim p(z|x;\theta)}\left[\log p(x,z;\theta)\right] + C \\ &\geq \mathbb{E}_{z\sim p(z|x;\theta)}\left[\log p(x,z;\theta)\right] \end{aligned}$$

J. $\mathbb{E}_{z\sim p(z;\theta)}\left[\log p(x,z;\theta)\right]$
Answer: YES.

$$\begin{aligned} p(x) &> \log p(x) \\ &\geq \mathbb{E}_{z\sim p(z;\theta)}\left[\log \frac{p(x,z;\theta)}{p(z;\theta)}\right] \\ &= \mathbb{E}_{z\sim p(z;\theta)}\left[\log p(x,z;\theta)\right] - \mathbb{E}_{z\sim p(z;\theta)}\left[\log p(z;\theta)\right] \\ &= \mathbb{E}_{z\sim p(z;\theta)}\left[\log p(x,z;\theta)\right] + C \\ &\geq \mathbb{E}_{z\sim p(z;\theta)}\left[\log p(x,z;\theta)\right] \end{aligned}$$

2. **[19 points] First steps with convolutional neural networks**

   Convolutional neural networks, also known as CNNs, are a specialized kind of neural network for processing data that has a known, grid-like topology. In this problem, we will construct a small one-dimensional CNN. One-dimensional CNNs are useful for dealing with one-dimensional continuous times series data and can be trained to perform tasks such as anomaly detection.

   Let's first consider a one-dimensional input vector $x = (x_1, ..., x_n)$ of time series values.

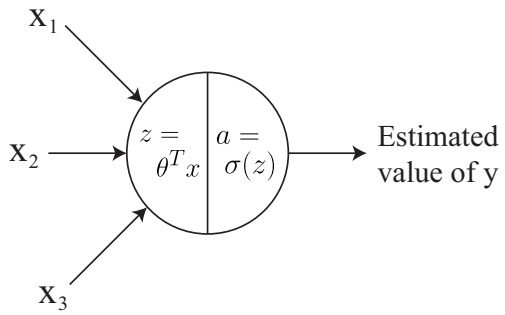
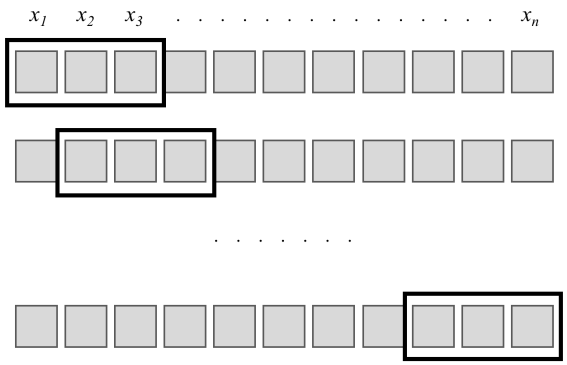


Figure 1: Logistic regression as a single neuron.

   Recall logistic regression. It can be represented as a neural network, as shown in Figure 1. The parameter vector $\theta = (\theta_1, ..., \theta_n)$ must have the same number of elements as the input vector $x = (x_1, ..., x_n)$. For an anomaly detection problem, this means θ_1 always looks at the first sample x_1 of the time series no matter what. However, we know that an anomaly might appear at any location within the input time series vector. It is possible that θ_1 was never trained on an anomaly at the first sample of the input time series vector. As a result, during test time, if an anomaly appears at the first time sample of the input vector, the logistic regression will likely predict *no anomaly*. This is a problem.

   This leads us to convolutional neural networks. Suppose θ is no longer a vector of the size of the input but instead a vector of the size of a small detection window. For our anomaly detection example, let's suppose $\theta = (\theta_1, ..., \theta_d)$ with $d \ll n$.



We now take our small vector of parameters θ and slide it over the input vector x. This is shown above by the thick rectangle (Note that in the diagram, $d = 3$). To compute the first activation z_1, we compute the dot product between θ and the portion of the vector x that overlaps with the position of the detection window θ. Formally:

$$z_1 = \sum_{i=1}^{d} \theta_i x_i \tag{1}$$

We then move this window slightly to the right in the input vector and repeat this process. The number of time samples by which we move the sliding window is called *stride*, which we will denote by s with $s \ll n$. (Note that in the diagram, $s = 1$). This allows us to compute the second activation z_2:

$$z_2 = \sum_{i=1}^{d} \theta_i x_{i+s} \tag{2}$$

We continue sliding the window until we reach the end of the input vector. The computed activations are stored into a vector z. Once we have reached the end of the input vector, the parameters θ have "seen" all times samples within x: θ_1 is no longer related to only the first time sample. As a result, whether the anomaly appears in the beginning or the end of the input vector, the neural network will successfully detect the anomaly.

(a) We just described a one-dimensional convolutional layer with input x, output z with parameters θ.

i. [1 points] Note that the vector z will be of a shorter length than x. Let m be the length of the vector z. Express m in terms of n, d, s. For simplicity, we will assume no padding to the input vector (that is to say, we are not adding any elements to the input vector to fit additional sliding windows), and that the stride is such that the last window lands exactly on the last elements of the input vector.
Answer: $\frac{n-d}{s} + 1$

ii. [2 points] For any convolutional layer, there is actually a fully-connected layer that implements the same forward function. Give an expression of the weight matrix W of this fully-connected layer, such that $z = Wx$.
Answer: $W \in \mathbb{R}^{m \times n}$ contains mostly zeros. Its elements are defined as:

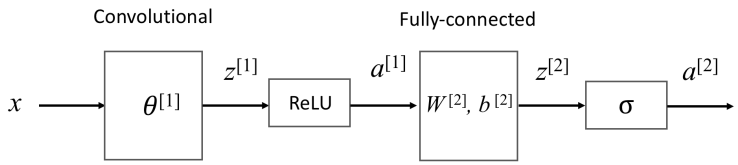
$$W_{ij} = \begin{cases} 0, & \text{for } j \leq (i-1)s \\ \theta_{j-(i-1)s}, & \text{for } (i-1)s < j \leq (i-1)s + d \\ 0, & \text{for } (i-1)s + d < j \end{cases}$$

Alternatively, we can define W as a matrix of m rows, where each row is of length n. Each row $(w_i)^T$ contains only zeros, except for the portion between indexes $(i-1)s + 1$ to $(i-1)s + d$ included, where it is equal to θ^T.

iii. [2 points] How many parameters does the convolutional layer approach have versus its fully-connected layer equivalent? What does this entail in terms of computational requirements for training a network with convolutional layers versus fully-connected layers? We will assume that we cannot store sparse matrices in a compressed format and that the matrix W would have to be stored in full.

Answer: The convolutional layer would have d parameters while the fully-connected layer equivalent would have $m \times n = \left(\frac{n-d}{s} + 1\right) n$ parameters. Since we have $d << n$, we see that the parameter sharing in the convolutional layer results in significantly less parameters, so less memory requirements and less computing requirements, since there are less operations to perform during the forward pass and less parameters to update during the backpropagation.

(b) We now incorporate this convolutional layer into a small neural network:



The first layer of the network is a convolutional layer as we have defined previously. We then use a Rectified Linear Unit (ReLU) activation function:

$$R(z) = \max(0, z)$$

The second layer of the network is a fully-connected layer:

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]},$$

followed by a sigmoid activation function.

After a single forward pass through the neural network, the output will be a predicted scalar value $\hat{y} = a^{[2]} \in (0, 1)$. The ground truth label $y \in \{0, 1\}$ indicates whether the input vector x contains an anomaly or not. We then compute the logistic loss:

$$\mathcal{L}(\hat{y}, y) = -\Big[(1 - y)\log(1 - \hat{y}) + y \log \hat{y}\Big]$$

i. [2 points] Specify the dimensions of each of the parameters of this neural network. Your answer should be expressed in terms of any subset of n, d, s or m.

Answer: $\theta^{[1]} \in \mathbb{R}^d$, $W^{[2]} \in \mathbb{R}^{1 \times m}$, $b^{[2]} \in \mathbb{R}$

ii. [6 points] Derive the gradient descent update rule for each of the parameters of this neural network assuming we use a learning rate of α.

Answer: We first compute the gradients:

$$\frac{\partial \mathcal{L}}{\partial b^{[2]}} = (a^{[2]} - y)$$
$$\frac{\partial \mathcal{L}}{\partial W_i^{[2]}} = (a^{[2]} - y) a_i^{[1]} \qquad \text{for } i = 1, \ldots, m$$
$$\frac{\partial \mathcal{L}}{\partial \theta_j^{[1]}} = (a^{[2]} - y) \left[\sum_{i=1}^{m} W_i^{[2]} \mathbb{1}\{z_i^{[1]} > 0\} x_{j+(i-1)s} \right] \qquad \text{for } j = 1, \ldots, d$$

which give us the following update rules for a single input example:

$$b^{[2]} := b^{[2]} - \alpha \frac{\partial \mathcal{L}}{\partial b^{[2]}}$$
$$W_i^{[2]} := W_i^{[2]} - \alpha \frac{\partial \mathcal{L}}{\partial W_i^{[2]}} \qquad \text{for } i = 1, \ldots, m$$
$$\theta_j^{[1]} := \theta_j^{[1]} - \alpha \frac{\partial \mathcal{L}}{\partial \theta_j^{[1]}} \qquad \text{for } j = 1, \ldots, d$$

We can also write the gradients with respect to $W^{[2]}$ and $\theta^{[1]}$ in matrix notation:

$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = (a^{[2]} - y) a^{[1]T}$$
$$\frac{\partial \mathcal{L}}{\partial \theta^{[1]}} = (a^{[2]} - y)\ J^T\ (W^{[2]T} \circ S) = (a^{[2]} - y)\ J^T\ \text{diag}(S)\ W^{[2]T},$$

where $\circ$ denotes element-wise multiplication and the elements of the Jacobian matrix $J \in \mathbb{R}^{m \times d}$ and the indicator vector $S \in \mathbb{R}^m$ are

$$J_{ij} = \frac{\partial z_i^{[1]}}{\partial \theta_j^{[1]}} = x_{j+(i-1)s} \qquad S_i = \mathbb{1}\{z_i^{[1]} > 0\},$$

which gives us the following update rules for a single input example:

$$W^{[2]} := W^{[2]} - \alpha \frac{\partial \mathcal{L}}{\partial W^{[2]}} \qquad \theta^{[1]} := \theta^{[1]} - \alpha \frac{\partial \mathcal{L}}{\partial \theta^{[1]}}$$

Alternatively, we also accepted gradient descent update rules for a batch of K input examples:

$$\frac{\partial \mathcal{L}}{\partial b^{[2]}}^{(k)} = (a^{[2](k)} - y^{(k)})$$
$$\frac{\partial \mathcal{L}}{\partial W^{[2]}}^{(k)} = (a^{[2](k)} - y^{(k)}) a^{[1](k)T}$$
$$\frac{\partial \mathcal{L}}{\partial \theta^{[1]}}^{(k)} = (a^{[2](k)} - y^{(k)}) J^{(k)T}\ (W^{[2]T} \circ S^{(k)})$$
$$= (a^{[2](k)} - y^{(k)}) J^{(k)T}\ \text{diag}(S^{(k)})\ W^{[2]T},$$

where $\circ$ denotes element-wise multiplication and the elements of the Jacobian matrix $J^{(k)} \in \mathbb{R}^{m \times d}$ and the indicator vector $S^{(k)} \in \mathbb{R}^m$ are

$$J_{ij}^{(k)} = \frac{\partial z_i^{[1](k)}}{\partial \theta_j^{[1]}} = x_{j+(i-1)s}^{(k)} \qquad S_i^{(k)} = \mathbb{1}\{z_i^{[1](k)} > 0\},$$

which gives us the following update rules:

$$b^{[2]} := b^{[2]} - \frac{\alpha}{K} \sum_{k=1}^{K} \frac{\partial \mathcal{L}}{\partial b^{[2]}}^{(k)}$$

$$W^{[2]} := W^{[2]} - \frac{\alpha}{K} \sum_{k=1}^{K} \frac{\partial \mathcal{L}}{\partial W^{[2]}}^{(k)}$$

$$\theta^{[1]} := \theta^{[1]} - \frac{\alpha}{K} \sum_{k=1}^{K} \frac{\partial \mathcal{L}}{\partial \theta^{[1]}}^{(k)}$$

(c) [6 points] We then decide to use a different loss function for our neural network. We now use the squared loss instead of the logistic loss:

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2.$$

Derive the gradient descent update rule for each of the parameters of this neural network with the new loss function assuming we use a learning rate of α.

Answer: We can reuse most of the calculations from the previous question and only change the partial derivative of $\frac{\partial \mathcal{L}}{\partial a^{[2]}}$. This gives us:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial b^{[2]}} &= (a^{[2]} - y)(1 - a^{[2]})a^{[2]} \\
\frac{\partial \mathcal{L}}{\partial W_i^{[2]}} &= (a^{[2]} - y)(1 - a^{[2]})a^{[2]}a_i^{[1]} && \text{for } i = 1, \cdots, m \\
\frac{\partial \mathcal{L}}{\partial \theta_j^{[1]}} &= (a^{[2]} - y)(1 - a^{[2]})a^{[2]} \left[\sum_{i=1}^{m} W_i^{[2]} \mathbb{1}\{z_i^{[1]} > 0\} x_{j+(i-1)s} \right] && \text{for } j = 1, \cdots, d
\end{aligned}$$

which gives us the following update rules for a single input example:

$$\begin{aligned}
b^{[2]} &:= b^{[2]} - \alpha \frac{\partial \mathcal{L}}{\partial b^{[2]}} \\
W_i^{[2]} &:= W_i^{[2]} - \alpha \frac{\partial \mathcal{L}}{\partial W_i^{[2]}} && \text{for } i = 1, \cdots, m \\
\theta_j^{[1]} &:= \theta_j^{[1]} - \alpha \frac{\partial \mathcal{L}}{\partial \theta_j^{[1]}} && \text{for } j = 1, \cdots, d
\end{aligned}$$

We can also write the gradients with respect to $W^{[2]}$ and $\theta^{[1]}$ in matrix notation:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial W^{[2]}} &= (a^{[2]} - y)(1 - a^{[2]})a^{[2]}a^{[1]T} \\
\frac{\partial \mathcal{L}}{\partial \theta^{[1]}} &= (a^{[2]} - y)(1 - a^{[2]})a^{[2]}J^T \left(W^{[2]T} \circ S\right) \\
&= (a^{[2]} - y)(1 - a^{[2]})a^{[2]}J^T \, \text{diag}(S) \, W^{[2]T}
\end{aligned}$$

where $\circ$ denotes element-wise multiplication Jacobian matrix $J \in \mathbb{R}^{m \times d}$ and the indicator vector $S \in \mathbb{R}^m$ are given in the answer to Problem 2(b) ii.

This gives us the following update rules for a single input example:

$$W^{[2]} := W^{[2]} - \alpha \frac{\partial \mathcal{L}}{\partial W^{[2]}} \qquad \theta^{[1]} := \theta^{[1]} - \alpha \frac{\partial \mathcal{L}}{\partial \theta^{[1]}}$$

Alternatively, we also accepted gradient descent update rules for a batch of K input

examples:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial b^{[2]}}^{(k)} &= (a^{[2](k)} - y^{(k)})(1 - a^{[2](k)})a^{[2](k)} \\
\frac{\partial \mathcal{L}}{\partial W^{[2]}}^{(k)} &= (a^{[2](k)} - y^{(k)})(1 - a^{[2](k)})a^{[2](k)}a^{[1](k)T} \\
\frac{\partial \mathcal{L}}{\partial \theta^{[1]}}^{(k)} &= (a^{[2](k)} - y^{(k)})(1 - a^{[2](k)})a^{[2](k)}J^{(k)T}\ (W^{[2]T} \circ S^{(k)}) \\
&= (a^{[2](k)} - y^{(k)})(1 - a^{[2](k)})a^{[2](k)}J^{(k)T}\ \text{diag}(S^{(k)})W^{[2]T}
\end{aligned}$$

where $\circ$ denotes element-wise multiplication Jacobian matrix $J^{(k)} \in \mathbb{R}^{m \times d}$ and the indicator vector $S^{(k)} \in \mathbb{R}^m$ are given in the answer to Problem 2(b) ii.

This gives us the following update rules:

$$\begin{aligned}
b^{[2]} &:= b^{[2]} - \frac{\alpha}{K}\sum_{k=1}^{K} \frac{\partial \mathcal{L}}{\partial b^{[2]}}^{(k)} \\
W^{[2]} &:= W^{[2]} - \frac{\alpha}{K}\sum_{k=1}^{K} \frac{\partial \mathcal{L}}{\partial W^{[2]}}^{(k)} \\
\theta^{[1]} &:= \theta^{[1]} - \frac{\alpha}{K}\sum_{k=1}^{K} \frac{\partial \mathcal{L}}{\partial \theta^{[1]}}^{(k)}
\end{aligned}$$

3. **[15 points] Linearity of Multinomial Naive Bayes**

Consider a Multinomial Naive Bayes model with a vocabulary of size k words, $W = \{w_1, w_2, ..., w_k\}$, to classify sentences as spam vs non-spam. Suppose a sentence with n words is denoted as $x \in W^n$, and has label $y \in \{0, 1\}$. According to the Naive Bayes assumption, we have $p(x|y) = \prod_{i=1}^{n} p(x_i|y)$ where x_i is the i^{th} word of the sentence. Let $\phi(x) \in \mathbb{R}^k$ be a feature extractor that maps a sequence x of n words, into a fixed length vector of length k (vocabulary), where $\phi(x)_i$ will be the count of the number of times the word w_i appears in x. Let the trained parameters of the model be, $\theta \in (0, 1)$ such that $\theta = p(y = 1)$, $\theta_{*|y=0} \in (0, 1)^k$ such that $\theta_{i|y=0} = p(w_i|y = 0)$, and $\theta_{*|y=1} \in (0, 1)^k$ such that $\theta_{i|y=1} = p(w_i|y = 1)$.

Now, show that the decision boundary of this model is linear in $\phi(x)$. Specifically, show that the points along some hyperplane $\phi(x)^\top m + c = 0$ correspond to having $p(y|x) = 0.5$, where $m \in \mathbb{R}^n$ and $c \in \mathbb{R}$. Clearly state what are m and c in terms of θ, $\theta_{*|y=0}$ and $\theta_{*|y=1}$.

Answer: Hint: Observe that $\prod_{i=1}^{n} p(x_i|y) = \prod_{i=1}^{k} p(w_i|y)^{\phi(x)_i}$.

Let n be the number of words in the sentence and k be the number of words in the dictionary

$$\begin{aligned} p(x|y) &= \prod_{i=1}^{n} p(x_i|y) \\ &= \prod_{i=1}^{k} p(w_i|y)^{\phi(x)_i} \\ \Rightarrow \log p(x|y) &= \phi(x)^\top \log \theta_{*|y} \text{(where } \log \theta_{*|y} \text{ indicates element-wise log)} \end{aligned}$$

According to the Bayes rule we have:

$$\begin{aligned} p(y|x) &= \frac{p(x|y)p(y)}{p(x)} \\ \Rightarrow \log p(y|x) &= \log p(x|y) + \log p(y) - \log p(x) \end{aligned}$$

At the decision boundary we have $p(y = 1|x) = p(y = 0|x)$:

$$\begin{aligned} \log p(y = 1|x) &= \log p(y = 0|x) \\ \Rightarrow \log p(x|y = 1) + \log p(y = 1) - \log p(x) &= \log p(x|y = 0) + \log p(y = 0) - \log p(x) \\ \Rightarrow \phi(x)^\top \log \theta_{*|y=1} + \log \theta &= \phi(x)^\top \log \theta_{*|y=0} + \log(1 - \theta) \\ \Rightarrow \boxed{\phi(x)^\top \log \left(\frac{\theta_{*|y=1}}{\theta_{*|y=0}} \right) + \log \left(\frac{\theta}{1 - \theta} \right) = 0} \end{aligned}$$

where $\frac{\theta_{*|y=1}}{\theta_{*|y=0}}$ indicates element-wise division.

So,

$$m = \log\left(\frac{\theta_{*|y=1}}{\theta_{*|y=0}}\right)$$
$$c = \log\left(\frac{\theta}{1-\theta}\right)$$

4. **[36 points] Kernels**

(a) [10 points] **String kernels** We saw in lecture that we can think of a kernel $K(x, z)$ as a measure of similarity between x and z. In this question, we will explore a kernel between two text documents. Specifically, we will consider two documents to be more similar if they have more substrings in common. We denote the alphabet (set of all symbols that occur in strings) by Σ, and the size of our alphabet by $|\Sigma|$.

Let each substring u of a document s be a dimension of the feature space. Note that the substrings do not have to be contiguous, i.e. '**c-a-r**' is a substring of both '**car**d' and '**c**ust**ar**d.' We will weight non-contiguous substrings using a decay factor $\lambda \in (0, 1)$. Therefore, we define a coordinate $\phi_u(s)$ of the feature mapping as

$$\phi_u(s) = \sum_{\mathbf{i}_u} \lambda^{i_{|u|} - i_1 + 1}$$

where $\mathbf{i}_u = (i_1, \ldots, i_{|u|})$, $0 \leq i_1 < i_2 < ... < i_{|u|} \leq |s|$, is a set of indices over s that form the substring u.

To illustrate this scheme, consider the simple alphabet $\Sigma = \{a, b, c, r, t\}$, and the feature mappings for the simple documents *cat, car, catcar, bat,* and *bar* for substring length $n = 2$ (note that this is not a very interesting n for real text documents):

	c-a	c-t	a-t	b-a	...	b-t	c-r	a-r	b-r
$\phi(cat) =$	$[\lambda^2,$	$\lambda^3,$	$\lambda^2,$	$0,$	...	$0,$	$0,$	$0,$	$0]$
$\phi(catcar) =$	$[2\lambda^2 + \lambda^5,$	$\lambda^3,$	$\lambda^2,$	$0,$	...	$0,$	$\lambda^6 + \lambda^3,$	$\lambda^5 + \lambda^2,$	$0]$
$\phi(car) =$	$[\lambda^2,$	$0,$	$0,$	$0,$	...	$0,$	$\lambda^3,$	$\lambda^2,$	$0]$
$\phi(bat) =$	$[0,$	$0,$	$\lambda^2,$	$\lambda^2,$	...	$\lambda^3,$	$0,$	$0,$	$0]$
$\phi(bar) =$	$[0,$	$0,$	$0,$	$\lambda^2,$	...	$0,$	$0,$	$\lambda^2,$	$\lambda^3]$

Using this feature map results in the string kernel,

$$K_{\text{str}}(s^{(i)}, s^{(j)}) = \phi(s^{(i)})^T \phi(s^{(j)})$$

for any two strings $s^{(i)}$ and $s^{(j)}$. Now answer the following questions:

i. [2 points] What is the dimensionality of $\phi(\cdot)$? Assume that we only consider substrings of length n for features over the alphabet Σ.

ii. [2 points] What does the string kernel evaluate to when applied between *cat* and *bat*? Give an expression in terms of λ.

iii. [2 points] Define the normalized string kernel $K_{\text{norm}}(x, z) = \frac{K_{\text{str}}(x,z)}{\sqrt{K_{\text{str}}(x,x) K_{\text{str}}(z,z)}}$. Show that this is a valid kernel.

iv. [2 points] Why might we need a normalized string kernel? In particular, normalization makes the string kernel invariant to which property of the document?

v. [2 points] What is the normalizing factor (i.e denominator in the normalized string kernel) when evaluated between *cat* and *bat*?

Answer: Students were allowed to assume $n = 2$ for parts ii and v.

i. The dimensionality is $|\Sigma^n|$ where Σ^n is the set of all finite strings of length n. (We also accepted $|\Sigma^n| = 25$ for the example Σ.)

ii. λ^4

iii. If we write $K_{\text{str}}(x, z) = \phi(x)^T \phi(z)$, we then have

$$K_{norm}(x, z) = \frac{\phi(x)^T \phi(z)}{\sqrt{\phi(x)^T \phi(x) \phi(z)^T \phi(z)}} = \psi(x)^T \psi(z)$$

where

$$\psi(x) = \frac{\phi(x)}{\sqrt{\phi(x)^T \phi(x)}} = \frac{\phi(x)}{\|\phi(x)\|}$$

iv. We normalize a string kernel to make it invariant to document length.

v. $2\lambda^4 + \lambda^6$.

(b) [16 points] **The Gaussian kernel** In lecture, we stated that the Gaussian kernel is a valid kernel corresponding to an infinite dimensional feature mapping ϕ, but we did not prove it. In this question, we will prove that the Gaussian kernel is valid and write down an explicit expression for ϕ. You may cite results shown in homeworks if you find it helpful.

Assume that K is a valid kernel.

i. [3 points] Show that $K_1(x, z) = \exp(K(x, z))$ is a valid kernel.
Hint: You may use without proof that Taylor expansion of e^x is given by $e^x = \sum_{i=0}^{\infty} \frac{1}{i!} x^i$.
Answer: Rewrite the kernel as $K_1(x, z) = \sum_{i=0}^{\infty} \frac{1}{i!} K(x, z)^i$. We proved in the homework that a polynomial in $K(x, z)$ with positive coefficients is a valid kernel. So, K_1 is a valid kernel.

ii. [3 points] Show that $K_2(x, z) = e^{-\frac{\|x-z\|^2}{\sigma^2}}$ is a valid kernel, using the result shown in the previous subquestion. Assume σ^2 to be some given constant.
Hint: You may use without proof that $\|x - z\|^2 = \|x\|^2 + \|z\|^2 - 2x^T z$.
Answer: Rewrite the kernel as

$$K_2 = e^{-\frac{\|x\|^2}{\sigma^2}} e^{-\frac{\|z\|^2}{\sigma^2}} e^{\frac{2x^T z}{\sigma^2}}$$

Let $f(x) = e^{-\frac{\|x\|^2}{\sigma^2}}$. Then the first two terms together form a kernel by the result proved in the homework that $K(x, z) = f(x)f(z)$ is a valid kernel. Let $\phi(x) = x$. Then the last term can be written as $e^{K(x,z)}$ and is a valid kernel by the result showed in part (i). We proved in the homework that the product of valid kernels is also a kernel, so we are done.
Special Notes: $e^{-\frac{\|x\|^2}{\sigma^2}}$, $e^{-\frac{\|z\|^2}{\sigma^2}}$, and $e^{-\frac{\|x\|^2}{\sigma^2}} e^{-\frac{\|z\|^2}{\sigma^2}}$ are not "constants" that you can ignore in any fashion.
$\|x - z\|^2$ is not a kernel. Neither is $-\|x - z\|^2$. Neither is $\|x\|^2$, $\|z\|^2$, or $\|x\|^2 + \|z\|^2$.

iii. [10 points] For this subquestion, we make a simplifying assumption that $x, z \in \mathbb{R}$ (i.e, scalars). For a finite dimensional feature extractor $\phi(\cdot) \in \mathbb{R}^d$, we define the inner product as

$$\langle \phi(x), \phi(z) \rangle = \phi(x)^T \phi(z) = \sum_{i=1}^{d} \phi(x)_i \phi(z)_i$$

However, for an infinite dimensional feature extractor $\phi(\cdot) \in \mathbb{R}^\infty$, we will abuse notation and consider ϕ to be a function of two inputs: x and t, where x is the input, and t indexes one of the infinitely many extracted features of x. Based on this, we define the inner product as

$$\langle \phi(x), \phi(z) \rangle = \int_t \phi(x, t)\phi(z, t)dt.$$

Now, consider the (simplified) Gaussian kernel $K(x, z) = \exp(-(x - z)^2/2)$. Come up with a function $\phi(x, t)$ such that

$$K(x, z) = \int_{-\infty}^{\infty} \phi(x, t)\phi(z, t)dt.$$

Hint 1: You may want to make the substitution $r = (x+z)/2$.
Hint 2: You may use the Gaussian integral without proof, $\sqrt{\pi/a} = \int_{-\infty}^{\infty} e^{-a(t-b)^2} dt$. This identity can be very handy to "introduce" an integral into an expression when there isn't one.

Answer:

$$
\begin{aligned}
K(x,z) &= \exp(-\frac{1}{2}(x-z)^2) \\
&= \exp(-\frac{1}{2}(x^2+z^2-2xz)) \\
&= \exp(-x^2-z^2)\exp(\frac{1}{2}(x+z)^2) \\
&= \exp(-x^2-z^2)\exp(2r^2) \quad \text{where } r = (x+z)/2 \\
&= \exp(-x^2-z^2+2r^2)(\pi/2)^{-1/2}\sqrt{\pi/2} \\
&= \exp(-x^2-z^2+2r^2)(\pi/2)^{-1/2}\int_{-\infty}^{\infty}\exp(-2(t-r)^2)dt \\
&= \exp(-x^2-z^2+2r^2)(\pi/2)^{-1/2}\int_{-\infty}^{\infty}\exp(-2t^2-2r^2+4tr))dt \\
&= \exp(-x^2-z^2+2r^2-2r^2)(\pi/2)^{-1/2}\int_{-\infty}^{\infty}\exp(-2t^2+4tr))dt \\
&= \exp(-x^2-z^2)(\pi/2)^{-1/2}\int_{-\infty}^{\infty}\exp(-2t^2+4tr))dt \\
&= (\pi/2)^{-1/2}\int_{-\infty}^{\infty}\exp(-2t^2-x^2-z^2+4tr))dt \\
&= (\pi/2)^{-1/2}\int_{-\infty}^{\infty}\exp(-2t^2-x^2-z^2+2t(x+z))dt \\
&= (\pi/2)^{-1/2}\int_{-\infty}^{\infty}\exp(-(x-t)^2-(z-t)^2)dt \\
&= \int_{-\infty}^{\infty}\left[(\pi/2)^{-1/4}\exp(-(x-t)^2)\right]\left[(\pi/2)^{-1/4}\exp(-(z-t)^2)\right]dt \\
&\Rightarrow \phi(x,t) = (\pi/2)^{-1/4}\exp(-(x-t)^2).
\end{aligned}
$$

(c) [10 points] **Kernelizing K-Means** We begin by briefly reviewing k-means. Given a training set of numerical values $\{x^{(1)}, \ldots, x^{(m)}\}$ to be grouped into k clusters,

1. Initialize cluster centroids $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n$.
2. Let $\{c^{(1)}, \ldots, c^{(m)}\}$ denote cluster identities for the next iteration.
3. Repeat until convergence: {

   Cluster assignment: For every i, set

   $$c^{(i)} := \arg\min_j ||x^{(i)} - \mu_j||^2.$$

   Centroids calculation: For each j, set

   $$\mu_j := \frac{\sum_{i=1}^m 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = j\}}.$$

   }

Our task in the problem is to kernelize this algorithm. Let $K(x, z)$ be a valid kernel, that has an implicit feature mapping $\phi(x)$. We want to effectively perform clustering in that higher dimensional feature space $\phi(x)$, without ever explicitly calculating $\phi(x)$ (since it could very well be infinite dimensional). Our task is to change step 3 in the above algorithm such that we replace all occurrences of $x^{(i)}$ and μ_j with calls to the kernel, such as $K(x^{(u)}, x^{(v)})$ for some u, v. Note that in our final algorithm, we specifically do not want to have isolated $\phi(x)$ terms or μ terms, since they could represent an infinite dimensional object and hence not computable. Also note that $\phi()$ can only be applied on data, and not on the parameters.

Your kernelized k-means algorithm should have the following template:

1. Randomly assign cluster identities $c^{(1)}, c^{(2)}, \ldots, c^{(m)} \in \{1, \ldots, k\}$.
2. Let $\{\tilde{c}^{(1)}, \ldots, \tilde{c}^{(m)}\}$ denote cluster identities for the next iteration.
3. Repeat until convergence: {

   Re-calculate: For every i, set

   $$\tilde{c}^{(i)} := \_\_\_\_\_.$$

   Assign: For each i, set

   $$c^{(i)} := \tilde{c}^{(i)}$$

   }

Please give the expression for $\tilde{c}^{(i)}$.

Hint 1: We make a similar observation as in the case of kernelizing the perceptron: for all j, μ_j is a linear combination of the $x^{(i)}$'s.

Hint 2: $(a - b)^2 = a^2 + b^2 - 2ab$.

Answer:

1. Randomly assign cluster identities $c^{(1)}, c^{(2)}, \ldots, c^{(m)} \in \{1, \ldots, k\}$.
2. Let $\{\tilde{c}^{(1)}, \ldots, \tilde{c}^{(m)}\}$ denote cluster identities for the next iteration.
3. Repeat until convergence: $\{$

   Re-calculate: For every i, set

$$\begin{aligned}
\tilde{c}^{(i)} &:= \arg\min_j \|\phi(x^{(i)}) - \mu_j\|^2 \text{ (abusing notation)} \\
&= \arg\min_j \phi(x^{(i)})^T \phi(x^{(i)}) + \mu_j^T \mu_j - 2\phi(x^{(i)})^T \mu_j \\
&= \arg\min_j \phi(x^{(i)})^T \phi(x^{(i)}) - 2\phi(x^{(i)})^T \left(\frac{\sum_{l \in S_j} \phi(x^{(l)})}{|S_j|} \right) \\
&\quad + \left(\frac{\sum_{l \in S_j} \phi(x^{(l)})}{|S_j|} \right)^T \left(\frac{\sum_{l \in S_j} \phi(x^{(l)})}{|S_j|} \right) \text{ (where } S_j = \{l : c^{(l)} = j\}) \\
&= \arg\min_j \left(K(x^{(i)}, x^{(i)}) - 2\frac{1}{|S_j|} \sum_{l \in S_j} K(x^{(i)}, x^{(l)}) + \frac{1}{|S_j|^2} \sum_{(l,p) \in S_j^2} K(x^{(l)}, x^{(p)}) \right) \\
&= \arg\min_j \left(-2\frac{1}{|S_j|} \sum_{l \in S_j} K(x^{(i)}, x^{(l)}) + \frac{1}{|S_j|^2} \sum_{(l,p) \in S_j^2} K(x^{(l)}, x^{(p)}) \right)
\end{aligned}$$

   Assign: For each i, set

$$c^{(i)} := \tilde{c}^{(i)}$$

   $\}$

5. **[26 points] Trees and Random Forests**

   (a) **[17 points] Tree Loss** When growing a decision tree, we split the input space in a greedy, top-down, recursive manner. Given a parent region R_p, we can choose a split $s_p(j,t)$ which yields two child regions $R_1 = \{X \mid x_j < t, X \in R_p\}$ and $R_2 = \{X \mid x_j \geq t, X \in R_p\}$. Assuming we have defined a per region loss $L(R)$, at each branch we select the split that minimizes the weighted loss of the children:

$$\min_{j,t} \frac{|R_1|L(R_1) + |R_2|L(R_2)}{|R_1| + |R_2|}$$

   When performing classification, a commonly used loss is the Gini loss, defined for the K-class classification problem as:

$$G(R_m) = G(\vec{p}_m) = \sum_{k=1}^{K} p_{mk}(1 - p_{mk})$$

   Where $\vec{p}_m = \begin{bmatrix} p_{m1} & p_{m2} & \dots & p_{mK} \end{bmatrix}$ and p_{mk} is the proportion of examples of class k that are are present in region R_m. However, we are oftentimes more interested in optimizing the final misclassification loss:

$$M(R_m) = M(\vec{p}_m) = 1 - \max_k p_{mk}$$

   For the problems below, assume we are dealing with binary classification ($k = 2$) and that there are no degenerate cases where positive and negative datapoints overlap in the feature space.

   i. [7 points] Show that for any given split, the weighted Gini loss of the children can not exceed that of the parent. (Hint: first show that the Gini loss is strongly concave).

   Answer:

   First note that we can re-write the Gini loss in the binary class in terms of a single proportion p_m, which we define as the proportion of positive examples in the region R_m:

$$\begin{aligned} G(\vec{p}_m) = G(p_m) &= p_m(1 - p_m) + (1 - p_m)(1 - (1 - p_m)) \\ &= 2p_m - 2p_m^2 \end{aligned}$$

   Thus, $\frac{d^2G}{dp_m^2}(p_m) = -4$ and the Gini loss is strongly concave.

   Defining $t = \frac{|R_1|}{|R_1|+|R_2|}$, we can write the parent region R_p proportion p_p as:

$$p_p = t * p_1 + (1 - t) * p_2$$

   Thus, we can use the definition of concavity to show that:

$$\begin{aligned} G(R_p) &= G(p_p) \\ &= G(tp_1 + (1-t)p_2) \\ &\geq tG(p_1) + (1-t)G(p_2) \\ &= \frac{|R_1|G(R_1) + |R_2|G(R_2)}{|R_1| + |R_2|} \end{aligned}$$

ii. [4 points] List out the cases where Gini loss will stay the same after a split. Show why these do not violate the strong concavity of the Gini loss. Briefly explain why these cases do not prevent a fully grown tree from achieving zero Gini loss.

Answer:

First recall the definition of strong concavity. G is strictly concave if:

$$\forall p_1 \neq p_2, \forall t \in (0,1) : G(tp_1 + (1-t)p_2) > tG(p_1) + (1-t)G(p_2)$$

There are two cases where the loss of the children will not be lower than the loss of the parent. The first is when one of the two children are empty. Then either $t = 0$ or $t = 1$, which is outside the range where the inequality is forced to be strict. The second case is when the proportion of positive and negative examples in each child remains the same as that of the parent. Then we have that $p_p = p_1 = p_2$, which again allows the inequality to not be strict.

These cases do not prevent a tree from fully fitting the data, as the lack of degenerate examples will always allow us to pick splits that have non-zero cardinality children, and the constant diminishing of the cardinality of the children will eventually force an uneven split where $p_1 \neq p_2$.

iii. [6 points] If instead we use misclassification loss, what additional case causes the loss to stay the same after a split? Show why this is (hint: you may find it useful to define $N_m = |R_m|$ and N_{mk} as the number of examples of class k present in R_m).

Answer:

If the majority class of the two children match that of the parent, the misclassification loss will remain the same. To show this, first define class c as the majority class prediction. We then have that:

$$\begin{aligned} \frac{|R_1|M(R_1) + |R_2|M(R_2)}{|R_1| + |R_2|} &= \frac{N_1(1-p_{1c}) + N_2(1-p_{2c})}{N_1 + N_2} \\ &= \frac{N_1(1-\frac{N_{1c}}{N_1}) + N_2(1-\frac{N_{2c}}{N_2})}{N_1 + N_2} \\ &= 1 - \frac{N_{1c} + N_{2c}}{N_1 + N_2} \\ &= 1 - \frac{N_{pc}}{N_p} = M(R_p) \end{aligned}$$

(b) **[9 points] Random Forests** We start this section of the problem with *bagging*. To set this up, recall bootstrap sampling, where we draw random samples Z of size N from the training data. After B iterations we obtain $Z_1, Z_2, ..., Z_B$, i.e. we generate B different bootstrapped data sets of the training data.

Now, we perform *bagging* with regression trees. We obtain tree T_i when trained with the bootstrapped training set Z_i. Observe that all the T_i's are random variables, since they are the output of a deterministic function (i.e the Decision Tree algorithm) over Z_i's (which are random). The final bagging model averages the predictions from each tree as follows:

$$\hat{T}(x) = \frac{1}{B}\sum_{i=1}^{B} T_i(x).$$

Note that $\hat{T}$ is also a random variable. We assume all the trees T_i have the same variance σ^2. Now, had all the trees $T_1, ..., T_B$ been independent of each other, the variance of $\hat{T}$ would have just been σ^2/B. However, in practice, the bagged trees can be correlated. So, let us assume $T_1, ..., T_B$ still have the same variance σ^2, but also have a positive pair-wise correlation ρ (recall that the correlation between two variables is just $\text{Cor}(X, Y) = \text{Cov}(X, Y)/\sqrt{\text{Var}(X)\text{Var}(Y)}$).

Thus, we have $\rho = \text{Cor}(T_i, T_j), \quad i \neq j$.

i. [6 points] Show that in this case, the variance of $\hat{T}$ is given by:

$$\text{Var}(\hat{T}) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

Answer:

$$\begin{aligned}
\text{Var}\left(\frac{1}{B}\sum_{i=1}^{B} T_i(x)\right) &= \frac{1}{B^2}\sum_{i=1}^{B}\sum_{j=1}^{B}\text{Cov}(T_i(x), T_j(x)) \\
&= \frac{1}{B^2}\sum_{i=1}^{B}\left(\text{Cov}(T_i(x), T_i(x)) + \sum_{j\neq i}^{B}\text{Cov}(T_i(x), T_j(x))\right) \\
&= \frac{1}{B^2}\sum_{i=1}^{B}(\sigma^2 + (B-1)\sigma^2\rho) \\
&= \frac{B(\sigma^2 + (B-1)\sigma^2\rho)}{B^2} \\
&= \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2
\end{aligned}$$

ii. [3 points] Explain how $\hat{T}$'s obtained with naive bagging (as considered above) have higher variance than the corresponding predictors obtained via the Random Forest algorithm.

Answer: As we increase the number of sample trees B, the second term goes down and the variance of the average of all the classifier becomes

Var$(\frac{1}{B}\sum_{i=1}^{B} T_i(x)) \approx \rho\sigma^2$. Random forests restrict the features at each split, resulting in classifiers with lower pairwise correlation than in standard bagging, which in turn will result in a low variance classifier.

That's all! Congratulations on completing the midterm exam!