6.036: Midterm, Fall 2019

Solutions

- This is a closed book exam. One page (8 1/2 in. by 11 in.) of notes, front and back, is permitted. Calculators are not permitted.
- The total exam time is 2 hours.
- The problems are not necessarily in any order of difficulty.
- Record all your answers in the places provided. If you run out of room for an answer, continue on a blank page and mark it clearly.
- If a question seems vague or under-specified to you, make an assumption, write it down, and solve the problem given your assumption.
- If you absolutely *have* to ask a question, come to the front.
- **Write your name on every page.**

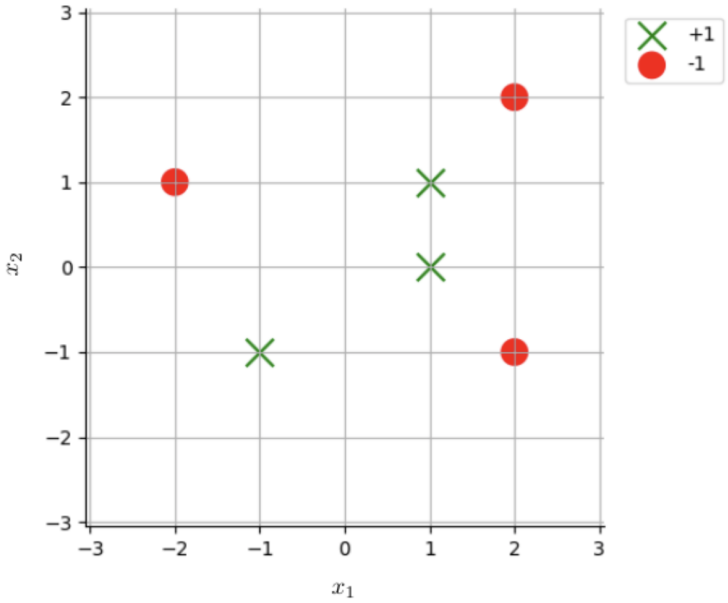
Name: _____ MIT Email: _____

Question	Points	Score
1	10	
2	15	
3	15	
4	15	
5	15	
6	15	
7	15	
Total:	100	

Name: _____

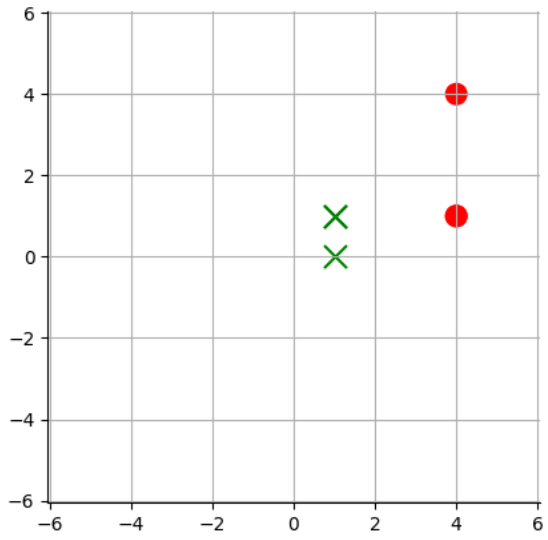
1 Linear classifier

1. (10 points) Consider the data in the plot below. The circles are negative and the X's are positive.



(a) Is the data linearly separable? ◯ Yes √ **No**

(b) Now, consider the feature transformation $\phi(x) = [x_1^2, x_2^2]$. Plot the points with feature transformation $\phi(x)$. Use circles for negative points and X's for positive ones.



(c) Is the data under feature transformation $\phi(x)$ linearly separable? √ **Yes** ◯ No

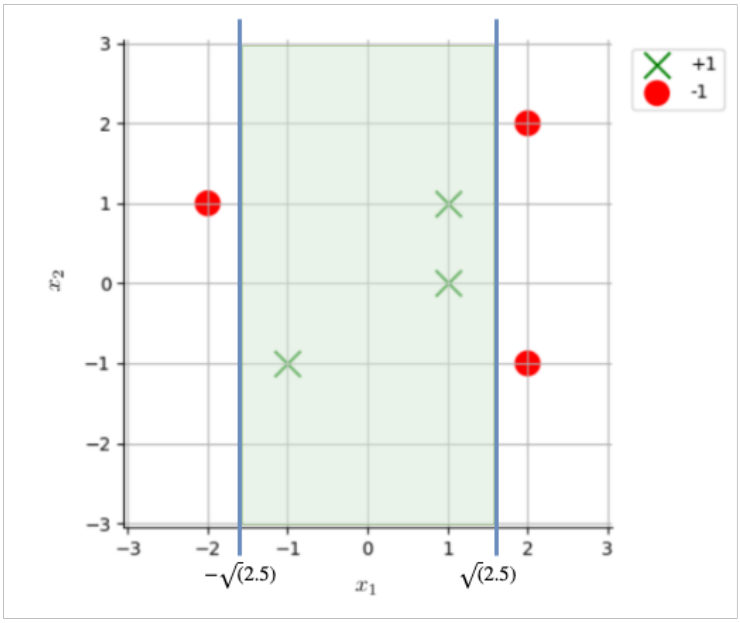
(d) Give θ and θ_0 values that define a linear separator of the *transformed* data.

- θ: $(-1, 0)$

- θ_0: 2.5

Note that there are many possible solutions to this problem.

(e) The linear separator that you just found above in the feature space corresponds to a non-linear separator in the original space. On the plot below (same as the first one), shade in the area *of the original space* that would be considered positive by the maximum-margin separator that you found in the feature space.



We checked for correctness relative to your answer to the previous part.

Name: _____

2 Perceptron

2. (15 points) We will explore several aspects of the perceptron algorithm.

   (a) Run the perceptron algorithm (with offset) on the following dataset where points are given in the form $(x^{(i)}, y^{(i)})$, showing the first iteration through the whole data set. Initialize θ to $(0, 0)$ and θ_0 to 0. Iterate through the points in the order given. Show each step of the algorithm.

   $$((-1,\ 2),\ -1),\ ((0,\ 3),\ -1),\ ((2,\ 4),\ 1),\ ((4,\ 4),\ 1)$$

datapoint x	datapoint y	mistake?	new θ	new θ_0
(-1, 2)	-1	yes	(1, -2)	-1
(0, 3)	-1	no		
(2, 4)	+1	yes	(3, 2)	0
(4, 4)	+1	no		

   (b) If the data set it is trained on is **NOT** linearly separable, the perceptron algorithm will (mark all that apply):

   ◯ Sometimes, but not always, stop updating the coefficients of the linear separator before reaching the maximum number of allowed iterations over the data, provided as an input parameter.

   ◯ Always stop updating the coefficients of the linear separator before reaching the maximum number of allowed iterations over the data, provided as an input parameter.

   √ **Sometimes, but not always, give a linear separator that misclassifies the smallest number of points achievable by a linear classifier.**

   ◯ Always give a linear separator that misclassifies the smallest number of points achievable by a linear classifier.

   (c) We are given a dataset $\mathcal{D}_n$ of points in 2D: $x \in \mathbb{R}^2$ and their corresponding labels $y \in \{+1, -1\}$. Consider the following two scenarios:

   1. We run the perceptron algorithm (with offset) on x, with θ initialized to $(0, 0)$ and θ_0 initialized to 0.
   2. We construct a 3D dataset x' from x by concatenating a 1 to each x: $x' = [x\ 1]^T$. We run the perceptron algorithm (without offset) on x' with the parameters for this model, θ', initialized to $(0, 0, 0)$.

   Assuming we iterate through the points in the same order, when we actually run these algorithms, will they make the same number of mistakes?

   √ **Yes** ◯ No

   Explain your answer.

   Solution: Scenario 1 update rule:

   $$[\theta_1, \theta_2] = [\theta_1, \theta_2] + y * [x_1, x_2]$$

   $$\theta_0 = \theta_0 + y$$

Scenario 1 update rule:

$$[\theta_1, \theta_2, 1] = [\theta_1, \theta_2, \theta_0] + y * [x_1, x_2, 1]$$

We can see that the results of the update rule is the same for each scenario; similarly, computing the signed distance of a point from the separator yields the same result. Thus, the two algorithms will have the same behavior.

(d) We are given a linearly separable dataset $\mathcal{D}_n$ of points $x^{(i)}$ in 2D that are contained with a circle of radius 1 centered at origin, and their corresponding labels $y^{(i)}$. If we shift our dataset so that the points are now centered around (5, 5), the theoretical bound for the number of mistakes of the perceptron algorithm should:

√ **Increase** ◯ Stay the same ◯ Decrease

Explain your answer.

Solution: The margin γ remains unchanged. R increases since each point moves away from the origin $(x_1', x_2') = (x_1 + 5, x_2 + 5)$. The theoretical bound $(\frac{R}{\gamma})^2$ on mistakes made by perceptron through origin increases.

It would have been clearer to specify in the problem that the perceptron goes through the origin, it is less clear what happens if it doesn't. If we add a dimension with 1 values and do perceptron through origin, then the argument is similar.

(e) We ran the perceptron algorithm, with offset, on the following dataset and recorded the number of mistakes we made for each point. Without running the algorithm, what is the final value of θ_0?

i	1	2	3	4	5
$x^{(i)}$	(-4, 3)	(-2, 2)	(-2, 0)	(1, 3)	(2, 0)
$y^{(i)}$	1	-1	-1	-1	-1
# mistakes	2	3	2	1	0

θ_0: -4

(f) Consider adding a learning rate $\eta \in (0, 1]$ to the standard perceptron algorithm **through the origin**. Our new update rule becomes

$$\theta^{(k+1)} = \theta^{(k)} + \eta x^{(i)} y^{(i)} \quad .$$

Assume we initialize $\theta^{(0)}$ to the 0 vector and we run the algorithm with separator through the origin. Does this new perceptron always make the same number of mistakes as the original perceptron ($\eta = 1$)?

√ **Yes** ◯ No

Explain why or why not.

Solution: Changing the learning rate shouldn't affect the algorithm. It changes $||\theta||$, but doesn't affect the direction θ points in and thus doesn't affect the sign of the prediction.

Name: _____

3 Regression

3. (15 points) (a) Reggie heard about standardizing features for classification and thought they'd try it for regression, too. Reggie has a one-dimensional linear regression data set (so $d = 1$) and so they decide to compute the transform

$$\begin{aligned} x_r^{(i)} &= \frac{x^{(i)} - \mu(X)}{\text{SD}(X)} \\ y_r^{(i)} &= \frac{y^{(i)} - \mu(Y)}{\text{SD}(Y)} \end{aligned}$$

where $\mu(X)$ is the mean, or average, of the data values $x^{(i)}$ and $\text{SD}(X)$ is the standard deviation. Then, they perform ordinary least squares regression using the $(x_r^{(i)}, y_r^{(i)})$ data points, and get the parameters θ and θ_0.

Now they have to perform a transformation on θ and θ_0 to obtain the θ^*, θ_0^* that solve the original problem (that is, so that it will work correctly on the original $(x^{(i)}, y^{(i)})$ data).

Write an expression for θ^* in terms of X, $\mu(X)$, $\text{SD}(X)$, Y, $\mu(Y)$, $\text{SD}(Y)$, θ and θ_0.

Solution:

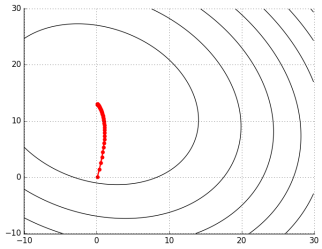
$$\theta^* = \frac{\text{SD}(Y)}{\text{SD}(X)}\theta$$

(b) Reggie ran ridge regression using several different parameter settings, but scrambled the graphs! The dimension of the data is $d = 1$, so there are two parameters, θ and θ_0, which are the axes of the graphs. The contour lines indicate the value of the overall objective J, and the connected points indicate the trajectory of the (θ, θ_0) values during the process of gradient update. It always starts near $(0, 0)$, with θ plotted on the x axis and θ_0 on the y axis.

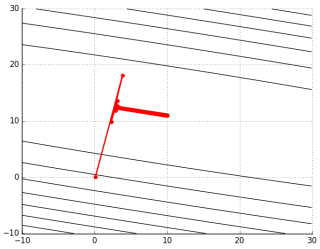
Which graph corresponds to which parameter settings?

- Step size: 0.05, 0.3, 0.7
- lambda : 0.0, 1.0

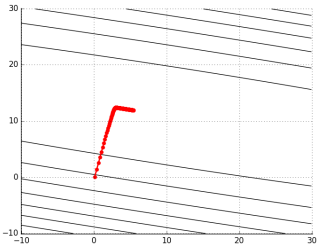
Name: _____



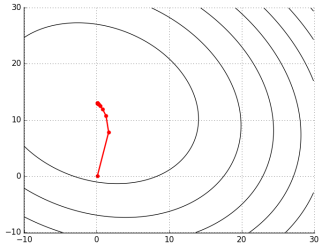
step size: **0.05** lambda: **1.0**



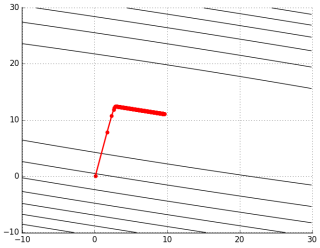
step size: **0.7** lambda: **0.0**



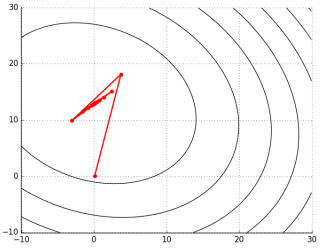
step size: **0.05** lambda: **0.0**



step size: **0.3** lambda: **1.0**



step size: **0.3** lambda: **0.0**



step size: **0.7** lambda: **1.0**

(c) We are considering formulating our machine-learning problem as an optimization problem with the following objective function

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

but we are not sure what regularizer R to use. For each of the possible choices listed below, answer the questions.

i. $R(\theta) = \sum_{j=1}^{d} \theta_j$
Is this equivalent to ridge regression? ◯ Yes √ **No**
Is this a reasonable choice for a regularizer? ◯ Yes √ **No**

ii. $R(\theta) = \sum_{j=1}^{d} |\theta_j|$
Is this equivalent to ridge regression? ◯ Yes √ **No**
Is this a reasonable choice for a regularizer? √ **Yes** ◯ No

iii. $R(\theta) = \sum_{j=1}^{d} \theta_j^2$
Is this equivalent to ridge regression? √ **Yes** ◯ No
Is this a reasonable choice for a regularizer? √ **Yes** ◯ No

iv. $R(\theta) = \sum_{j=1}^{d} \theta_j^3$
Is this equivalent to ridge regression? ◯ Yes √ **No**
Is this a reasonable choice for a regularizer? ◯ Yes √ **No**

v. $R(\theta) = \theta^T\theta$
Is this equivalent to ridge regression? √ **Yes** ◯ No
Is this a reasonable choice for a regularizer? √ **Yes** ◯ No

Name: _____

4 Deep neural networks

4. (15 points) Nori thinks about reLU units and wonders whether there's a better alternative for an activation function, and decides to explore the LUre function, defined as:

$$f_{\text{LUre}}(z) = \min(z, 0) \ .$$

(a) What is the derivative of this function, $df_{\text{LUre}}(z)/dz$?

Solution:

$$\frac{df_{\text{LUre}}(z)}{dz} = \begin{cases} 1 & \text{if } z < 0 \\ 0 & \text{otherwise} \end{cases}$$

Or any variants thereof. Note that the derivative is **never -1!**

(b) Nori's friend Ori thinks this is cool and suggests making a neural network with two activation functions per layer, and, in particular so that

$$a^l = f_{\text{LUre}}(f_{\text{reLU}}(z^l)) \ .$$

Explain what effect this will have on the network.

Solution: This will effectively make the activation function $f(z) = 0$ for all z, destorying all expressivity of the NN.

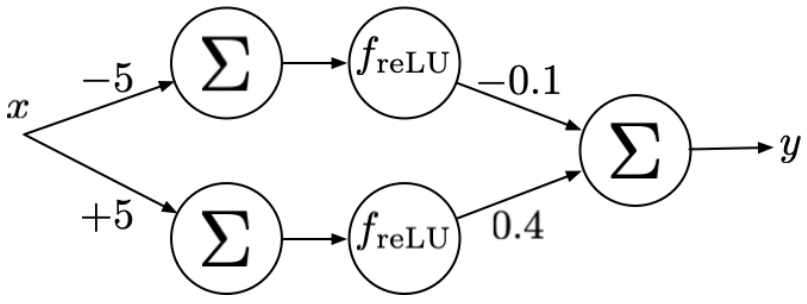
(c) Nori's other friend Dori thinks we should try this trick with two reLUs, so that

$$a^l = f_{\text{reLU}}(f_{\text{reLU}}(z^l)) \ .$$

Explain what effect this will have on the network.

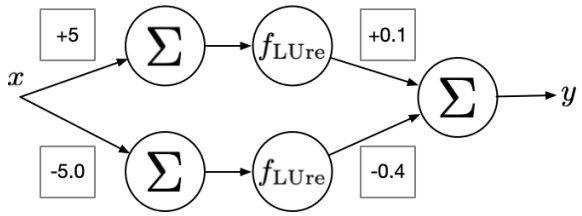
Solution: It will act exactly the same as having one reLU, this stacked activation function is the same as a single reLU.

(d) Nori finds a neural network trained by Smaug in his treasure pile that takes a single-dimensional input (so $d = 1$) and looks like this:



He sees that it computes $\hat{y} = -0.1 \cdot f_{\text{reLU}}(-5x) + 0.4 \cdot f_{\text{reLU}}(5x)$ and is very curious to see if he can replace those reLU activation units with his own LUres. Please help him find another neural network that computes exactly the same function as the one above (that is, maps any input x to the same output as the original one). Provide a set of weights that achieves this in the boxes on the diagram below.

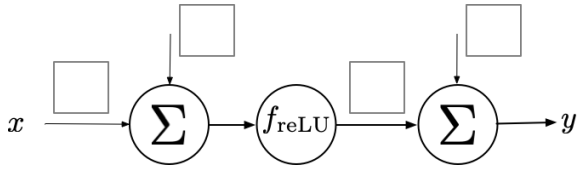
Name: _____



(e) Ori, Dori, and Nori have this data set with their heights and whether they like beer:

$$x^{(1)} = [1] \quad y^{(1)} = 1$$
$$x^{(2)} = [2] \quad y^{(2)} = 0$$
$$x^{(3)} = [3] \quad y^{(3)} = 1$$

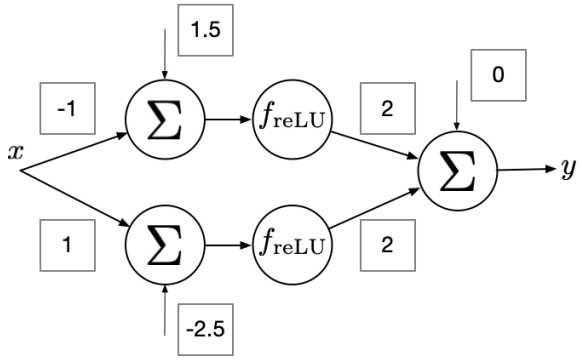
They make a two-layer neural network, as shown below:



Are there weights and biases for this network that will predict their data correctly? If so, specify them in the boxes on the network diagram. If not, argue why not.

Solution: No. We can think of each hidden unit as one separating hyperplane, and since we need at least two separating hyperplanes (one to separate $x^{(1)}$ from $x^{(2)}$, and another to eparate $x^{(2)}$ from $x^{(3)}$), we need at least two hidden units.

(f) Thorin suggests they add one more unit, so that they have this architecture.



Are there weights and biases for this network that will predict their data correctly? If so, specify them in the boxes on the network diagram. If not, argue why not.

Name: _____

5 Model Evaluation

5. (15 points) Triage is done in emergency rooms when limited medical resources must be allocated to maximize the number of survivors. Some versions of the triage system involve a color-coding scheme using, in decreasing order of treatment urgency, red (immediate), yellow (observation), green (wait), and white (dismiss) tags. In normal days at the BT Hospital (BTH), the proportion of tags given to patients is 5% red, 25% yellow, 60% green, and 10% white. You can assume that these tags are completely accurate.

   The BTH staff is training classifiers based on physiological signals and other patients' features in order to identify the level of urgency of its patients more quickly; classifications will help inform final triage decisions. Assume that BTH stores anonymized records of 10,000 patients.

   (a) For this question assume for some reason that you can only use one held-out set to evaluate a classifier, so you divide the data into two parts: part 1 has 70% of the data and part 2 has 30%.

   i. If you want to train a classifier that is more likely to work well on new data, how would you use the parts?

   √ **part 1 (70%) for training, part 2 (30%) for testing**

   ◯ part 2 (30%) for training, part 1 (70%) for testing

   ii. If you want to be more sure about your classifier's error estimate, how would you use the parts?

   ◯ part 1 (70%) for training, part 2 (30%) for testing

   √ **part 2 (30%) for training, part 1 (70%) for testing**

   This is an obscure way Of saying "test error estimate" — a guess of what the test error would be

   iii. If we want to train a good classifier *and* predict the model's performance highly accurately, would it be a good idea to instead use all data for both training and testing? Why or why not?

   Solution: No – this overfits and under-estimates the error.

   (b) Aiming to predict its model's performance for future patients, BTH uses accuracy score to evaluate classifiers. We define `accuracy score`, where N is the number of predictions, as

$$\frac{\sum_{i=1}^{N} \text{equals}(y_i, \hat{y}_i)}{N}$$

   where $\text{equals}(y_i, \hat{y}_i) := 1$ when y_i (actual tag) equals $\hat{y}_i$ (predicted tag), and 0 otherwise.

   i. If the model assigns random tags so that each tag is equally probable, what would the average accuracy score be? ◯ 5% √ **25%** ◯ 33.3% ◯ 50% ◯ 60%

   ii. If the model always predicts the most likely tag, what would the average accuracy score be? ◯ 5% ◯ 25% ◯ 33.3% ◯ 50% √ **60%**

   (c) Notice that a model could reach a 95% accuracy score while mis-identifying all of the patients who need help most urgently (the red color tags). How can we change our evaluation metric so that it treats the prediction accuracy on each class equally? Define a new scoring formula with this property.

Solution:

$$\frac{1}{4}\sum_{j=\{1,2,3,4\}}\frac{1}{N_j}\sum_{\{i|y^{(i)}=j\}}\text{equals}(y_i,\hat{y}_i)$$

where N_j is the number of data points with class j.

(d) Accuracy score also assumes that all classification errors have equal cost. But, for example, it is very costly to predict *white* when the answer should have been *red*. Assume you have a function $C(g,a)$ where g is your predicted tag color and a is the actual one, which quantifies how bad it is to predict g when the correct answer is a.

Provide a scoring formula in terms of C that takes these error costs into account.

Solution:

$$\frac{1}{n}\sum_{i=1}^{N}C(\hat{y}^{(i)},y^{(i)})$$

We accepted the negation of this, as well, since the question did not make clear whether we were looking for an "accuracy" function, for which higher values are better, or a loss function, for which lower values are better.

(e) Evelyn thinks the C function approach is too complicated and suggests simply assigning integer values *red* $= 4$, *yellow* $= 3$, *green* $= 2$, *white* $= 1$ and letting $C(g,a) = (g-a)^2$. Either (1) explain why Evelyn's approach is as expressive as using a general C function, or (2) give a concrete example in this domain of a preference that Evelyn's method cannot express.

Solution: It is not as expressive. For example, we cannot say that it is worse to predict 3 when the target is 4 than it is to predict 4 when the target is 3.

Name: _____

6 CTO Logistic Regression

6. (15 points) You have a training dataset $\mathcal{D}$ with each input $x^{(i)}$ consisting of d binary features $x_1^{(i)}, \ldots, x_d^{(i)}$, where all $x_j^{(i)} \in \{0, 1\}$ and a binary label $y^i \in \{0, 1\}$. After hours of struggling with *underfitting* (high training-set error) when using logistic regression to make predictions, you decide to call your friend, the CTO of the famous start-up *ClosedAI*, who gives you a few ideas to get a lower training error. For each of them, specify whether: 1) it generally reduces underfitting; 2) it will not make a difference; or 3) it generally worsens underfitting. **Note that we are only talking about training loss, not test loss. Justify each answer, either by describing when underfitting will be improved/worsened or why there will be no difference.**
Notation:

- $[a, b, \ldots, z]$ concatenation
- $\widehat{y^{(i)}}$: prediction for example i.
- $\sigma(z) = \frac{1}{1+e^{-z}}$, the sigmoid function.
- $\widehat{y} = \sigma(\sum \theta_j x_j + \theta_0) = 1/(1 + e^{-(\sum \theta_j x_j + \theta_0)})$, the regular (base-e) logistic regression which has to be compared against all other versions. Note that it contains an offset!
- $\widehat{y} = \sigma_2(\sum \theta_j x_j + \theta_0) = 1/(1 + 2^{-(\sum \theta_j x_j + \theta_0)})$, base-2 logistic regression.

(a) *"Consider the other point of view":* For each data-point, augment the dimensions by adding the *complement* of each feature before trying to fit the dataset; i.e. $[x_1^{(i)}, \ldots, x_d^{(i)}] \rightarrow [x_1^{(i)}, \ldots, x_d^{(i)}, 1 - x_1^{(i)}, \ldots, 1 - x_d^{(i)}]$

◯ generally reduces underfitting √ **no change** ◯ generally increases underfitting

Solution: Any solution to this new logistic regression can be parametrized as $\widehat{y^i} = \sigma\left(\sum_{j=1}^{f} \alpha_j x_j^i + \sum_{j=1}^{f} \beta_j (1 - x_j^i) + \gamma\right)$, then using the distributive property:

$$\widehat{y^i} = \sigma\left(\sum_{j=1}^{f} \alpha_j x_j^i + \sum_{j=1}^{f} \beta_j (-x_j^i) + \sum_{j=1}^{f} \beta_j \cdot 1 + \gamma\right)$$

$$\widehat{y^i} = \sigma\left(\sum_{j=1}^{f} (\alpha_j - \beta_j) x_j^i + \left(\sum_{j=1}^{f} \beta_j + \gamma\right)\right)$$

which has the form of the original logistic regression.
It was enough to say that because the new features are a linear function of the original ones, they do not add expressive power.

(b) *"Two is bigger than one":* Use two sigmoids instead of one; i.e., instead of parametrizing the solution as $\widehat{y^{(i)}} = \sigma(\sum \theta_j x_j^{(i)})$, parametrize it as: $\widehat{y^{(i)}} = \sigma(\sigma(\sum \theta_j x_j^{(i)}))$.
◯ generally reduces underfitting ◯ no change √ **generally increases underfitting**

> **Solution:** The output of $\sigma(z)$ is between 0 and 1 for all z. Therefore the output of $\sigma(\sigma(z))$ is between $\sigma(0) = 0.5$ and $\sigma(1) \approx 0.73$, which severely limits its range and in particular cannot predict $y^i = False$.

(c) *"Ensembling is always good":* Fit a regular logistic regression using the e-based σ sigmoid and a second logistic regression using a base-2 σ_2 sigmoid function and return the most confident result (furthest away from 0.5).
◯ generally reduces underfitting √ **no change** ◯ generally increases underfitting

> **Solution:**
>
> $$\sigma_2(\sum_{j=1}^{f} \alpha_j x_j^i + \gamma) = \frac{1}{1 + 2^{-(\sum \alpha_j x_j^i + \gamma)}} = \frac{1}{1 + e^{-\ln 2(\sum x_j^i + \gamma)}} = \sigma(\sum_{j=1}^{f} \alpha_j x_j^i \ln 2 + \gamma \ln 2)$$
>
> Therefore, σ_2 logistic regression and σ logistic regression are equally expressive and training them will yield equal predictions for all datapoints. Therefore ensembling them in the proposed manner will not change results.

(d) *"Random distillation":* Before fitting the dataset, for every data-point i, flip a coin and add the result of that flip as a feature; i.e., $[x_1^{(i)}, \ldots, x_d^{(i)}] \rightarrow [x_1^{(i)}, \ldots, x_d^{(i)}, \text{coin}(i)]$.
√ **generally reduces underfitting** ◯ no change ◯ generally increases underfitting

> **Solution:** First, it is clear that this function class is at least as expressive because we can always set the coefficient multiplying $coin(i)$ to 0. Then, with high probability, the random feature will have non-0 correlation with the labels and will thus contain some information about them that can be exploited to reduce underfitting. (More concretely, for n datapoints the probability of this happening is $\frac{\binom{n}{n/2}}{2^n} \rightarrow 0$ and 0 for n odd, this level of detail was not needed for a satisfactory answer.) Another way of seeing it is that $coin(i)$ is a function that is not a linear combination of the current feature set and will thus increase the capacity of our classifier.
>
> Note that this would be a bad idea because it would lead to poor generalization, but it would still help with underfitting.

(e) *"Not quite a 2-layer neural network":* Fit a regular logistic regression and obtain a prediction $\widehat{y^{(i)}}$ for each element $x^{(i)}$. Add $\widehat{y^{(i)}}$ as a feature $([x_1^{(i)}, \ldots, x_d^{(i)}, \widehat{y^{(i)}}])$ and fit another logistic regression to the new dataset.
√ **generally reduces underfitting** ◯ no change ◯ generally increases underfitting

Solution: $\widehat{y_i}$ is a non-linear function combination of the features and therefore increases the capacity of the logistic regresssion. Moreover, $\widehat{y_i}$ contains information on the labels through the parameters trained during the first logistic regression, which makes it a very informative feature.

Name: _____

7 Traffic prediction

7. (15 points) You were just hired by Ways, and they ask you to use machine-learning methods to predict the traffic at intersections in a city.

   (a) The first thing you have to decide is how to encode the inputs. The first city you analyze data for is designed on a grid, where all streets either run due north/south or due east/west. You know the names of all the streets in the city in advance. You have measurements of the traffic at many intersections at many times of day, and you have to encode each measurement using a vector of feature values.

   For each of the features below, explain how best to encode it. Your encoding should be able to take into account both the fact that traffic at different points on the same road tends to be correlated and the fact that some whole parts of a city might be busier than others.

   i. Name of street that runs north/south

   ii. Name of street that runs east/west

   > **Solution:** Use one-hot encoding for each street. These features are important because the street name can help generalization (there is lots of traffic on Mass Ave, but not on the alleyway next door).

   iii. Latitude (positive real)

   iv. Longitude (positive real)

   > **Solution:** Treat each as a positive real value. Technically longitude is a cyclic quantity, so you'd have trouble if your city were Greenwich, UK, but we did not expect you to handle that here.

   v. Day of week (an integer from 1 to 7)

   > **Solution:** One choice might be a binary feature for weekend or not. One-hot is also good. Integers are not good, because the implied ordering doesn't make sense (Monday is not "closer" to Sunday than to Wednesday.)

   vi. Time of day (a real number between 0 and 24)

   > **Solution:** Full credit for noting that there is a wrap-around problem even if not correctly handled. Ideal would be to represent it in 2D by mapping into range $0 - -2\pi$ and then use the sin and cos as features. Another reasonable answer is to discretize into bins.

   (b) You do such a good job in that city, you are assigned to work on a new city that is very old, and although there are still only two streets involved in an intersection, the streets wind all around and don't run in a consistent direction, so it's not clear which order to put them in. So, a training example for an intersection might be arbitrarily described as one of

   ```
   ("1st street", "park street", 34.4, 54.2, 02139, 22.35, Friday)
   ("park street", "1st street", 34.4, 54.2, 02139, 22.35, Friday)
   ```

You can assume that for any two streets, there is only a single intersection between them. How would you change your encoding of the first two features to deal with this, or would you leave it as is? Explain your answer.

> **Solution:** We create a bag-of-words-like encoding. Namely, for each of the n street in the city, we assign an index, and the intersection of streets indexed i and j is represented as an n-dimensional vector of zeros with ones at indices i and j.
>
> It is important that your representation still be able to efficiently represent "there is a lot of traffic on Mass Ave at 5PM on weekdays". If you do one-hot encoding of intersections, you can't generalize well.

(c) Your goal is to predict how much traffic there is at a given intersection at a given time, so now you have to think about how to encode the output. We'll just consider a single direction of a single road. The raw data that you have contains either a positive number or `None`. If it's a number, that is the average speed in miles per hour of the cars that went through this road in the past 10 minutes. If it is `None`, then no cars went through in the last 10 minutes.

Design an output encoding for this data. Specify how many dimensions it has, and precisely what transformation you would do on the raw output values in order to compute your encoding.

> **Solution:** When we wrote this question, we assumed that the only thing that would cause the case that "no cars went through in the last 10 minutes" was that there were no cars on the road at all (so the effective speed was infinite, or the speed limit. However, some students pointed out that this condition could also happen when the traffic was at a standstill, in which case the speed is 0.
>
> We gave up and gave everyone full credit.
>
> In the case that None means traffic is at a standstill, you can just convert None to 0 and encode as a numeric feature. If it means that there are no cars at all, then we need to do something special. One good solution is to use two features: one for whether this value is None, and one encoding the speed if not. Another good one is to convert None to the speed limit.

(d) You are using a neural network to predict traffic using the encoding you selected above. What activation function(s) would you use on your output units, and what loss function would you use? Explain your choices.

> **Solution:** Depends on choice above. If one-hot bins, then softmax/NLLM. If positive numeric values then relu/squared loss. If general (positive or negative values) then the identity function and squared loss.