CS 135: Intro to Machine Learning

Midterm Exam

Thurs. Oct 26, 2023

Your Name:

Your Student ID:

Date:

DO NOT BEGIN UNTIL INSTRUCTED.

You have 60 minutes to complete this exam.

Remember to **skim through all questions** first and **prioritize the order** you answer the questions.

Write complete yet concise answers *legibly!* in the space after each prompt. Please show your work! Calculation steps may earn partial credit.

Good luck!

Useful Facts and Formulas can be found on the *last sheet* of this packet.

1. **True/False with justification.**

   For each claim, circle True or False. Provide a 1-2 sentence justification.

   (a) When using K-Nearest Neighbors regression, the training step takes longer to compute than the prediction step

   True False Justification: ___________

   (b) In gradient descent, we need to ensure the step size is always between 0.0 and 1.0 otherwise the loss after each step will always get worse.

   True False Justification: ___________

   (c) When training logistic regression, one reason we might prefer L1 (sum of absolute value) regularization over L2 (sum of squares) regularization on the weights is that L1 penalties can better identify which features to ignore.

   True False Justification: ___________

1. True/False (continued).

(d) Suppose we fit two MLP binary classifiers, each with one hidden layer of 100 units. One gets a training set of size 5 and the other gets a training set of size 5000, with both sets drawn from the same distribution. **Claim:** We should generally expect the MLP trained on less data to have higher training error rate.

True False Justification: ___________

(e) Consider a grid search for the L2 penalty strength $\alpha > 0$ of logistic regression on a dataset with 100 examples. You use 5-fold cross-validation across a grid of 3 possible α values. **Claim:** The grid search requires 15 separate calls to 'fit' (each with a train set of size 80), and then building the final model (with chosen α) requires one additional call to 'fit' on the full size 100 dataset.

True False Justification: ___________

2. **Linear Regression for Elephants**

   You are working with biologists to predict an adult elephant's weight y_i (in kilograms) given measurements $x_i \in \mathbb{R}^F$ of the animal's physical size.

   You have a small training set of $N = 50$ examples and $F = 3$ numerical features, each representing a numerical length of some body part.

   In your training set, elephant weights y_i range between 2200 - 6400 kg, with a mean of 4123 kg.

   You care about minimizing squared error when predicting y. Consider two variants of L2-penalized linear regression:

$$\min_{w \in \mathbb{R}^F, \quad b \in \mathbb{R}} \quad \alpha(b^2 + \textstyle\sum_{f=1}^F w_f^2) + \sum_{n=1}^N (y_n - (w^T x_n + b))^2 \tag{1}$$

$$\min_{w \in \mathbb{R}^F, \quad b \in \mathbb{R}} \quad \alpha(\textstyle\sum_{f=1}^F w_f^2) + \sum_{n=1}^N (y_n - (w^T x_n + b))^2 \tag{2}$$

   (a) Linear regression with a penalty on the bias coefficient b, like in Eq. (1), would as $\alpha \to \infty$ always predict $\hat{y}(x_i) =$ _______

   Explain:

   (b) Linear regression with NO penalty on the bias coefficient b, like in Eq. (2), would as $\alpha \to \infty$ always predict $\hat{y}(x_i) =$ _______

   Explain:

2. Linear Regression for Elephants (continued).

(c) For this elephant task, would recommend penalizing the bias, or not? Why?

(d) Describe how you would select a specific α value using grid search. Provide a range of reasonable candidates, and explain how you'd select among them using only the provided $N = 50$ dataset. Be sure to specify and justify your strategy for estimating generalization error on heldout data.

3. **Evaluating binary classifiers**

Suppose we are building classifiers for a disease screening task (1 means does have disease, 0 means does not).

We show the confusion matrix of three different thresholds (A, B, C) for the same classifier. Recall from HW2 that using different thresholds can yield different confusion matrices, even for the same classifier.

A

Predicted True	0	1
0	140	20
1	10	20

B

Predicted True	0	1
0	160	0
1	20	10

C

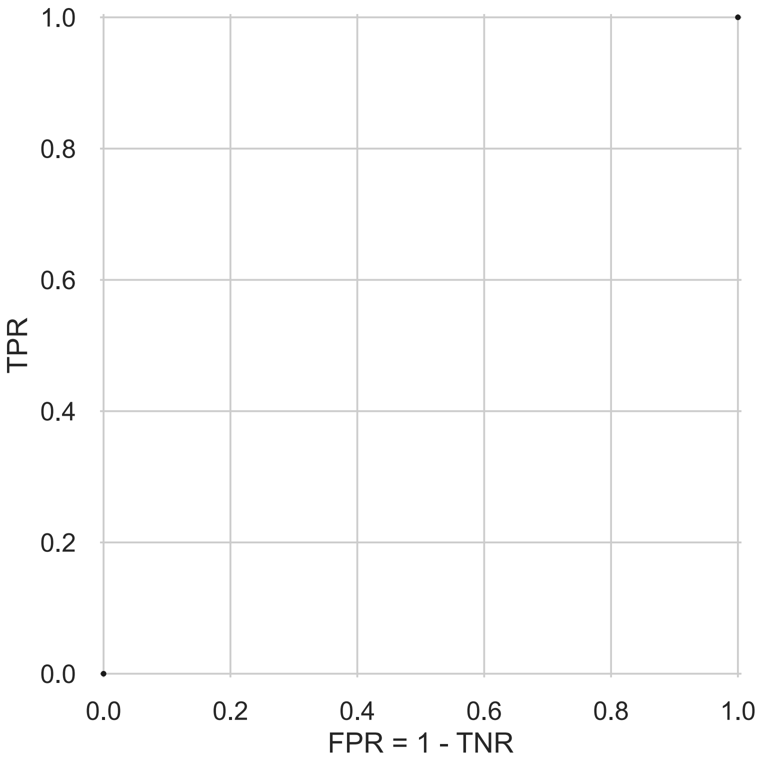
Predicted True	0	1
0	60	100
1	0	30

(a) Compute the true positive rate (TPR) of each classifier A, B, and C.

(b) Compute the true negative rate (TNR) of each classifier A, B, and C

3. Evaluating binary classifiers (continued).

(c) Draw the 3 points corresponding to these classifiers in the ROC plot, and label them as 'A', 'B', 'C'.



(d) For a new classifier, you compute an area-under-ROC of $v = 0.80$. But there's a bug (see diagram)! You accidentally multiplied each predicted probability by 0.5 before computing the ROC curve. Explain how the correct area-under-ROC u is related to the area v that uses buggy probabilities. Is u definitely larger than v? Smaller? The same? Or impossible to tell?

n	y_n	$\hat{p}_n^{\text{bug}}$	$\hat{p}_n$
1	0	0.02	0.04
2	1	0.49	0.98
3	1	0.23	0.46
$\vdots$			
N	0	0.35	0.70

4. **Softmax without NaNs.**

Given a K-length vector of real-valued scores $z = [z_1, \dots z_K]$, recall that the softmax function is a mapping from z to a vector of the same length with non-negative probability values that sum to one:

$$\text{softmax}([z_1, \dots z_K]) = \left[\frac{e^{z_1}}{\sum_{k=1}^K e^{z_k}} \ \frac{e^{z_2}}{\sum_{k=1}^K e^{z_k}} \ \dots \ \frac{e^{z_K}}{\sum_{k=1}^K e^{z_k}} \right]$$

We could also ask for the log-of-softmax, which returns a K-length vector that is the element-wise log of the above

$$\text{log-softmax}([z_1, \dots z_K]) = \left[\log(\frac{e^{z_1}}{\sum_{k=1}^K e^{z_k}}) \ \dots \ \log(\frac{e^{z_K}}{\sum_{k=1}^K e^{z_k}}) \right]$$

Log-of-softmax is useful when computing the cross-entropy loss of a multi-class neural net.

Possible lines. Use these lines of code in your solutions to (a) and (b) on the next page.

```
1       return np.exp(z_K) / denom
2       return np.exp(log_numer_K) / denom
3       return log_numer_K - log_denom
4       denom = np.sum(np.exp(z_K))
5       denom = np.sum(np.exp(log_numer_K))
6       log_denom = logsumexp(np.hstack([z_K, 0]))
7       log_denom = logsumexp(log_numer_K)
8       m = np.max(z_K)
9       m = np.min(z_K)
10      log_numer_K = z_K
11      log_numer_K = z_K - m
```

You can assume all logarithms use base e.

4. Softmax without NaNs (continued).

Define a function as *numerically stable* if it will never return nan or inf given finite inputs, regardless of how big or small those inputs are.

(a) Softmax

Using only a *subset* of the lines of code 1-11 in the listing above, provide a implementation of softmax that is *correct* and *numerically stable*.

Please copy each full line, including line number and code.

```
def softmax(z_K):

    ----

    ----

    ----

    ----
```

(b) Log of Softmax

Using only a *subset* of the lines of code 1-11 in the listing above, provide a implementation of the element-wise log of softmax that is *correct* and *numerically stable*.

Please copy each full line, including line number and code.

```
def log_of_softmax(z_K):

    ----

    ----

    ----

    ----
```

Blank page for extra work

Useful Facts and Formulas (page 1/2)

Binary classifier formulas:

Given a dataset with n_+ positive labeled examples and n_- negative labeled examples in the true labels y, we can count all four possible (label, prediction) paired outcomes:

n_{TP}	number of true positives
n_{FP}	number of false positives
n_{FN}	number of false negatives
n_{TN}	number of true negatives

True Positive Rate:

$$\text{TPR} = \frac{n_{\text{TP}}}{n_{\text{TP}} + n_{\text{FN}}} = \frac{n_{\text{TP}}}{n_+}$$

True Negative Rate:

$$\text{TNR} = \frac{n_{\text{TN}}}{n_{\text{TN}} + n_{\text{FP}}}$$

False Positive Rate (FPR) = 1 - TNR

Least-squares formulas:

Let our training data consist of response vector y with N rows, one per training example, and feature matrix X with N rows and F columns (features). The least-squares or "ordinary least-squares" update to find optimal weight parameters $w^* \in \mathbb{R}$ for linear regression is:

$$w^* = (X^T X)^{-1} X^T y.$$

This solves the optimization problem: $w^* = \arg\min_w \sum_n (y_n - x_n^T w)^2$

Log loss, also called Binary Cross Entropy (uses a base-2 logarithm):

$\text{log\_loss}(y_i, \hat{p}_i) = -y_i \log_2 \hat{p}_i - (1 - y_i) \log_2 (1 - \hat{p}_i)$

where $y_i \in \{0, 1\}$ is the true binary label of example i

and $\hat{p}_i \in (0, 1)$ is the predicted probability that i should be labeled 1

Useful Facts and Formulas (page 2/2)

Logistic sigmoid:

$$\text{logistic\_sigmoid}(z) = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{1+e^z}$$

where $z \in \mathbb{R}$ is a real-valued score

Values of this function, in tabular form:

z	$\sigma(z)$	$\log_2(\sigma(z))$
-3.000	0.047	-4.398
-2.500	0.076	-3.721
-2.000	0.119	-3.069
-1.500	0.182	-2.455
-1.000	0.269	-1.895
-0.500	0.378	-1.405
0.000	0.500	-1.000
0.500	0.622	-0.684
1.000	0.731	-0.452
1.500	0.818	-0.291
2.000	0.881	-0.183
2.500	0.924	-0.114
3.000	0.953	-0.070

Log-sum-exp:

This function maps a vector of size L to a scalar real.

$$\text{logsumexp}([a_1, a_2, \ldots a_L]) = \log\left(e^{a_1} + e^{a_2} + \ldots e^{a_L}\right) \tag{5}$$

You can assume a Python implementation is numerically stable (will not overflow or return NaN for any finite input)