CS37300 Homework	— Homework 3—	Last update: Feb 28, 2024
Due Date (Theory): March 8, 2024 (11:59 pm) — Due Date (Programming): March 20, 2024 (11:59 pm)		

Name: Derrick Khoo PUID: 0037028290

Instructions and Policy: You are allowed to study with others and use online resources for reference, however, the work you turn in must be your own. This means do not copy/paste from Stack Exchange (or from another student.) If you have worked closely with other students, provide their name(s) and a brief (at most one paragraph) description of the interaction; if we feel this oversteps the bounds, we will discuss it with you. Each student should write up their own solutions independently. The requirements below are supposed to be followed in this and further homework assignments.

- For your theoretical submission:
  - YOU MUST INCLUDE YOUR NAME IN THE HOMEWORK.
  - The answers MUST be submitted via Gradescope.
  - Please write clearly and concisely - clarity and brevity will be rewarded. Refer to known facts as necessary.
  - Theoretical questions MUST include the intermediate steps to the final answer.
- For your programming answers:
  - The Python scripts will be submitted separately via Gradescope
  - Zero points will be given in any question where the Python code answer doesn't match the answer on Gradescope.
  - If the answer is/includes a plot, it should be added to your theoretical submission unless otherwise specified.

Your code is REQUIRED to run on Python 3.11 in an environment detailed in Homework 0 under the Python Installation section. While your code may run in other environments, any points lost due to environmental issues will not be regraded.

Please make sure you don't use any libraries that are not imported for you. If such a library is used, you will get 0 pt for the coding part of the assignment.

If your code doesn't run on Gradescope, then even if it compiles on another computer, it will still be considered not running and the respective part of the assignment will receive 0 pt.

Theoretical Questions (8+20+14+16+18=76 pts)

Please submit your answers on Gradescope.

Q1 (8 pts): True or False questions

Answer the following as True or False with a justification or example. Points are uniformly distributed wihtin the questions.

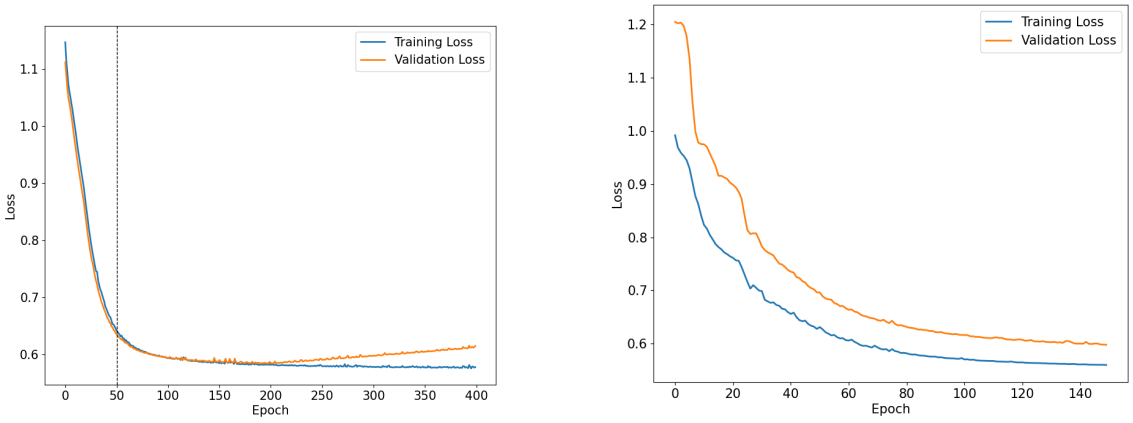
(a) (**4 pts**) High bias can result in the model learning random noise in the training data.

False. A high bias would result in a model that underfits the the test data and is likely to be a simple model. Hence, this could mean that the model would be rather generalised and factors specific types of solutions and thus random noises would not be able to learn random noise in the training data.

(b) (**4 pts**) In k-fold cross-validation, each sample is used to train the model k times.

False. This is because for every iteration i, we train the model on $S_1 \cup S_2 \cup .. S_{i-1} \cup S_{i+1} \cup ... \cup S_k$. Thus, when the model is trained k times, each sample is used for a total of k-1 times, and not k times.

Q2 (20 pts): Cross-validation



(a) Training and validation loss of a model M_1

(b) Training and validation loss of a model M_2

Figure 1: Example of training curves showing training and validation loss for two hypothetical models M_1 and M_2.

1. **(6 pts)** Review the training and validation loss curve in Figure 1a (model M_1). The y-axis shows a loss (lower is better) for the training and validation data. The x-axis shows the iterations to update the model parameters (e.g. update steps on Perceptron algorithm or steps of gradient descent for logistic regression).
   Hint: Note that these are training curves, not learning curves.

   (a) **(3 pts)** What are the best parameters for model M_1? The parameters at 150 iterations or at 400 iterations? Please justify your answer.

   The parameters at 150 iterations. This is because at 150 iterations, the training loss and validation loss is the lowest, at ≈ 0.6 for both, as compared to at 400 iterations where the validation loss is higher at ≈ 0.66, despite having a lower training loss. This indicates that at 400 iterations, the model has overfitted to the training data.

   (b) **(3 pts)** What is the disadvantage of choosing the model with parameters obtained after 50 iterations?

The disadvantage is that the model has not fully converged and may be underfitting to the training data. This is because the model may not have learned enough from the training data to capture the underlying pattern of the data and thus, resulting in higher training and validation losses.

2. **(4 pts)** Review the training and validation loss curve in Figure 1b of model M_2.

   Which of the following explanations and decisions are appropriate for the observed behavior in 1b?
   Choose all answers that are correct. You don't need to justify your answer.

   (A) The validation data is scarce and not very representative.

   (B) The training data does not provide sufficient information to learn the problem relative to the validation data used to evaluate it.

   (C) Getting more validation data will not reduce the training loss.

   (D) The training loss has converged, so obtaining more and better training data will not reduce the training loss further.

C

3. **(4 pts)** Which statement about k-fold cross-validation is incorrect?
   Find out the false statements.

   (A) When using k-fold cross-validation, using a smaller k means it will take more time to run the evaluation.

   (B) With larger k, the training set gets larger, while the test set gets smaller. X

   (C) When splitting data into folds, it is desirable to minimize the variance across folds. X

   (D) When $k = N$, where N is the size of the data set, k-fold cross-validation is not the same as leave-one-out cross-validation.

A, D

4. **(6 pts)** Given the supervised learning dataset with 3 training examples $(X_1, Y_1) = (-2, -1)$, $(X_2, Y_2) = (-1, -1)$, $(X_3, Y_3) = (0, 1)$, use the Perceptron algorithm to learn a classifier. If needed, use $\text{sign}(0) = 1$. The Perceptron classifier will have two parameters $w = (w_0, w_1)$ (the first parameter, w_0 acts as the bias). What is the average 0-1 loss of the Perceptron algorithm evaluated by 3-fold cross-validation? (Please also describe, for each fold/iteration of the cross-validation, what is the training data used and the loss obtained).

Using Perceptron algorithm on these 3 training examples:
We initialise $w_0 = 0, w_1 = 0$ first for each iteration. Then, we run the algorithm:
On first iteration, we use (x_1, y_1), (x_2, y_2) as training data and (x_3, y_3) as validation set:

$\text{sign}(w_1^T x_1 + w_0) = \text{sign}(0) = 1$

$\therefore w_1 = 0 + (-1)(-2) = 2$

$w_0 = 0 + (-1) = -1$

$\text{sign}(w_1^T x_2 + w_0) = \text{sign}(2(-1) + (-1)) = \text{sign}(-3) = -1$

$\text{sign}(w_1^T x_1 + w_0) = \text{sign}(2(-2) + (-1) = \text{sign}(-5) = -1$

On validation set, $\text{sign}(w_1^T x_3 + w_0) = \text{sign}(2(0) + 1) = \text{sign}(1) = 1$

$\therefore$ 0-1 loss obtained $= 0$ as it is correctly classified.

On second iteration, we use (x_1, y_1) and (x_3, y_3) as training data and (x_2, y_2) as validation set and $w_0 = w_1 = 0$ initially.

$\text{sign}(w_1^T x_1 + w_0) = \text{sign}(0) = 1$

$\therefore w_1 = 0 + (-2)(-1) = 2$

$w_0 = 0 + (-1) = -1$

$\text{sign}(w_1^T x_2 + w_0) = \text{sign}(2(0) - 1) = \text{sign}(-1) = -1$

$\therefore w_1 = 2 + (1)(0) = 2$

$w_0 = -1 + (1) = 0$

$\text{sign}(w_1^T x_1 + w_0) = \text{sign}(2(-2) + 0) = \text{sign}(-4) = -1$

$\text{sign}(w_1^T x_3 + w_0) = \text{sign}(2(0) + 0) = \text{sign}(0) = 1$

Q2.4 On validation set, $\text{sign}(w_1^T x_2 + w_0) = \text{sign}(2(-1) + 0) = \text{sign}(-2) = -1$

(continued) $\therefore$ 0-1 loss obtained $= 0$ as it is correctly classified.

On third iteration, we use (x_2, y_2) and (x_3, y_3) as training data and (x_1, y_1) as validation set, and $w_1 = w_0 = 0$ initially.

$\text{sign}(w_1^T x_2 + w_0) = \text{sign}(0) = 1$

$\therefore w_1 = 0 + (-1)(-1)$

$= 1$

$w_0 = 0 + (-1)$

$= -1$

$\text{sign}(w_1^T x_3 + w_0) = \text{sign}(1(0) + (-1)) = \text{sign}(-1) = -1$

$\therefore w_1 = 1 + (0)(1)$

$= 1$

$w_0 = -1 + 1$

$= 0$

$\text{sign}(w_1^T x_2 + w_0) = \text{sign}(1(-1) + 0) = \text{sign}(-1) = -1$

$\text{sign}(w_1^T x_3 + w_0) = \text{sign}(1(0) + 0) = \text{sign}(0) = 1$

On validation set, $\text{sign}(w_1^T x_1 + w_0) = \text{sign}(1(-2) + 0) = \text{sign}(-2) = -1$

$\therefore$ 0-1 loss obtained $= 0$ as it is correctly classified.

$\therefore$ Average 0-1 loss of the Perceptron algorithm $= \frac{1}{3}\sum_{i=1}^{3} E_{0-1} = \frac{0+0+0}{3}$

$= 0$ //

Q3 (14 pts): Ensemble methods (AdaBoost)

Consider the dataset in the following table:

Index (i)	**Feature 1** (X_{i1})	**Feature 2** (X_{i2})	**Class label** (y_i)
1	-1	1	-1
2	1	1	+1
3	1	-1	+1
4	-1	-1	+1

For this question, you have to show the first few steps of the AdaBoost algorithm. The weak learner can be one of the four possible decision stumps illustrated in Fig. 2 (i.e., at each step you must choose one of the classifiers in the figure, to minimize the weighted empirical error rate, defined as $\varepsilon_t = \sum_{i=1}^{n} D_t(i)\mathbb{I}[h_t(x_i) \neq y_i]$, where t means the number of iterations and $D_t(i)$ means the reweighting of data at round t, breaking any ties by choosing the classifier with lower number). You may refer to this link `https://drive.google.com/file/d/1OWABdO_gvkOcDgt3WsiNx_b-hoOUOvXM/view?usp=sharing` for details of AdaBoost.

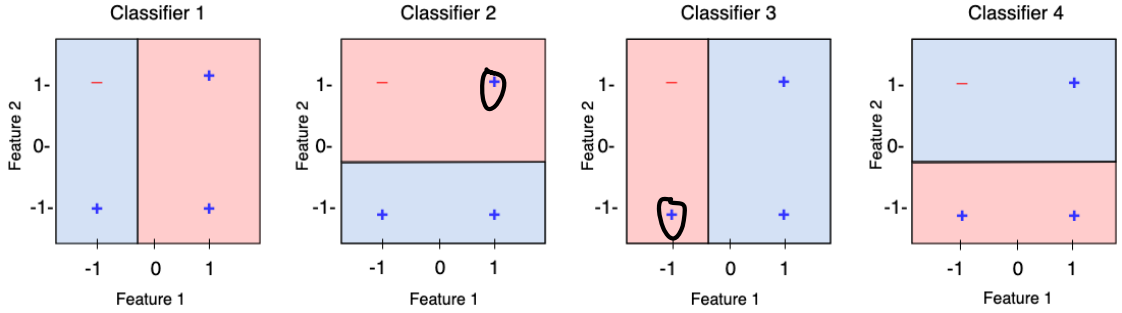


Figure 2: Choices of classifiers for the dataset. The solid red areas are assigned label -1 and the hatched blue areas are assigned label $+1$

1. **(6 pts)** Complete the following table, with the correct values computed during the execution of the AdaBoost algorithm, for $t = 1, \ldots, 5$. Please refer to lecture slides for details of AdaBoost.

$\varepsilon_t = \sum_{i=1}^{n} D_t(i)\mathbb{I}[h_t(x_i) \neq y_i]$

t	ε_t	α_t	$D_t(i)$ $i=1$	$i=2$	$i=3$	$i=4$	h_t Classifier	$\mathbb{I}(h_t(x_i) \neq y_i)$ $i=1$	$i=2$	$i=3$	$i=4$
1	0.25	0.549	0.25	0.25	0.25	0.25	2	0	1	0	0
2	0.166	0.809	0.166	0.501	0.166	0.166	3	0	0	0	1
3	0.301	0.424	0.0996	0.301	0.0996	0.500	2	0	1	0	0
4	0.359	0.298	0.0713	0.500	0.0713	0.357	3	0	0	0	1
5	0.406	0.190	0.0578	0.373	0.0578	0.440	2	0	1	0	0

2. **(4 pts)** Complete the following table, with the predictions computed by the AdaBoost algorithm for the training data, at the end of training (i.e. after $T = 5$ rounds).

	$\hat{h}_t(x_i)$ $i=1$	$i=2$	$i=3$	$i=4$
$\sum_{t=1}^{T} \alpha_t h(x_i)$	-2.261	-0.059	2.261	0.059
$\hat{h}(x_i)$	-1	-1	+1	+1

Q3.1

(Calculations)

$t=1$: $q_t = 0.25(1)$

$= 0.25$

$\alpha_t = \frac{1}{2}\ln\left(\frac{1-0.25}{0.25}\right)$

$= 0.549$ (3sf)

$t=2$: $D_t(1) = D_t(3) = D_t(4)$

$= 0.25 \cdot e^{-0.549}$

$= 0.1444$ (3sf)

$D_t(2) = 0.25\, e^{0.549}$

$= 0.433$ (3sf)

Renormalizing:

$D_t(1) = D_t(3) = D_t(4)$

$= \frac{0.144}{0.144(3)+0.433}$

$= 0.166$ (3sf)

$D_t(2) = \frac{0.433}{0.144(3)+0.433}$

$= 0.501$ (3sf)

$q_t = 0.166(1)$

$= 0.166$

$\alpha_t = \frac{1}{2}\ln\left(\frac{1-0.166}{0.166}\right)$

$= 0.807$ (3sf)

$t=3$:

$D_t(1) = D_t(3)$

$= 0.166\, e^{-0.807}$

$= 0.0741$ (3sf)

$D_t(2) = 0.501\, e^{-0.807}$

$= 0.224$ (3sf)

$D_t(4) = 0.166\, e^{0.807}$

$= 0.372$ (3sf)

Renormalizing:

$D_t(1) = \frac{0.0741}{(2)0.0741+0.224+0.372}$

$= 0.0996 = D_t(3)$

$D_t(2) = \frac{0.224}{(2)0.0741+0.224+0.372}$

$= 0.301$

$D_t(4) = \frac{0.372}{(2)0.0741+0.224+0.372}$

$= 0.500$ (3sf)

$q_t = 0.301\,(1) = 0.301$

$\alpha_t = \frac{1}{2}\ln\left(\frac{1-0.301}{0.301}\right)$

$= 0.421$

$t=4$:

$D_t(1) = D_t(3)$

$= 0.0996 e^{-0.421}$

$= 0.0659$ (3St)

$D_t(2) = 0.301\, e^{0.421}$

$= 0.459$ (3St)

$D_t(4) = 0.500 e^{-0.421}$

$= 0.328$ (3St)

Renormalizing:

$D_t(1) = \frac{0.0659}{2(0.0659) + 0.459 + 0.328}$

$= 0.073 = D_t(3)$

$D_t(2) = \frac{0.459}{2(0.0659) + 0.459 + 0.328}$

$= 0.500$ (3St)

$D_t(4) = \frac{0.328}{2(0.0659) + 0.459 + 0.328}$

$= 0.357$ (3St)

$q_t = 0.357$ (1) $= 0.357$

$\alpha_t = \frac{1}{2}\ln\left(\frac{1-0.357}{0.357}\right)$

$= 0.294$

$t=5$:

$D_t(1) = D_t(3)$

$= 0.0913 e^{-0.294}$

$= 0.0531$ (3St)

$D_t(2) = 0.500 e^{-0.294}$

$= 0.373$ (3St)

$D_t(4) = 0.328 e^{0.294}$

$= 0.440$ (3St)

Renormalizing:

$D_t(1) = \frac{0.0531}{2(0.0531) + 0.373 + 0.440}$

$= 0.0578 = D_t(3)$

$D_t(2) = \frac{0.373}{2(0.0531) + 0.373 + 0.440}$

$= 0.406$ (3St)

$D_t(4) = \frac{0.440}{2(0.0531) + 0.373 + 0.440}$

$= 0.479$ (3St)

$q_t = 0.406$ (1) $= 0.406$

$\alpha_t = \frac{1}{2}\ln\left(\frac{1-0.406}{0.406}\right)$

$= 0.190$ (3St)

Q3.2

(calculations)

	$\hat{h}_t(x_i)$			
	$i=1$	$i=2$	$i=3$	$i=4$
$\sum_{t=1}^{T} \alpha_t h(x_i)$				
$\hat{h}(x_i)$				

$i=1$: $\sum_{t=1} \alpha_t h(x_i) = 0.549(-1) + 0.807(-1) + 0.921(-1) + 0.294(-1) + 0.190(-1)$
$= -2.261$

$\hat{h}(x_i) = \text{sign}(-2.261)$
$= -1$

$i=2$: $\sum_{t=1} \alpha_t h(x_i) = 0.549(-1) + 0.807(1) + 0.921(-1) + 0.294(1) + 0.190(-1)$
$= -0.059$

$\hat{h}(x_i) = \text{sign}(-0.059)$
$= -1$

$i=3$: $\sum_{t=1} \alpha_t h(x_i) = 0.549(1) + 0.807(1) + 0.921(1) + 0.294(1) + 0.621(1)$
$= 2.261$

$\hat{h}(x_i) = \text{sign}(2.261)$
$= 1$

$i=4$: $\sum_{t=1} \alpha_t h(x_i) = 0.549(1) + 0.807(-1) + 0.921 + 0.294(-1) + 0.621(1)$
$= 0.059$

$\hat{h}(x_i) = \text{sign}(0.059)$
$= 1$

3. **(4 pts)** Assuming we stopped the algorithm after finishing the step $t = 4$, what is the predicted label for a new sample $\underline{x} = (0, 1)$? If the true label of this new test example is $+1$, what is the 0-1 loss for this example?

Predicted label = +1

∵ True label = +1, and predicted label = +1,

0-1 loss for this example = 0.

Q4 (20 pts): SVM

Consider an SVM that obtains a decision boundary through the maximization optimization in Equation (1), which is equivalent to the minimization optimization in Equation (2), where β and β_0 denote the parameters, x_i the i-th training example, $y_i \in \{-1, 1\}$ the class label for the i-th training example, M the margin, N the number of instances.

$$\begin{aligned} &\max_{\beta,\beta_0,\|\beta\|_2=1} M \\ &\text{subject to } \; y_i \left(x_i^T \beta + \beta_0\right) \geq M, \; i = 1, \ldots, N. \end{aligned} \tag{1}$$

which is equivalent to

$$\begin{aligned} &\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|_2^2 \\ &\text{subject to } \; y_i \left(x_i^T \beta + \beta_0\right) \geq 1, \; i = 1, \ldots, N. \end{aligned} \tag{2}$$

Mark true or false for the following assertions (Q4.1, Q4.2, Q4.3) and justify your answer. Questions with no justification will receive 0 points.

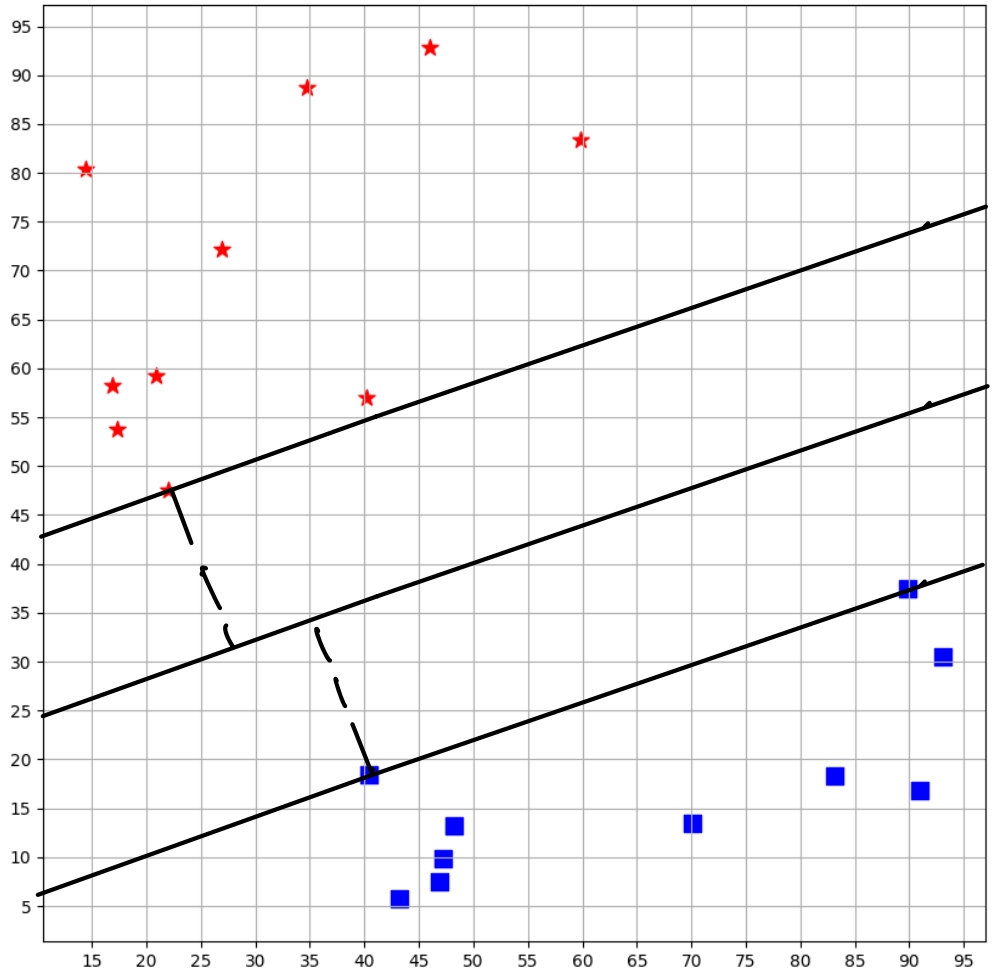
1. **(4 pts)** This SVM objective works for both linearly separable and non-linearly-separable data.

   False. This SVM objective is for hard margin SVMs, which works only for linearly separable data and not non-linearly separable data.

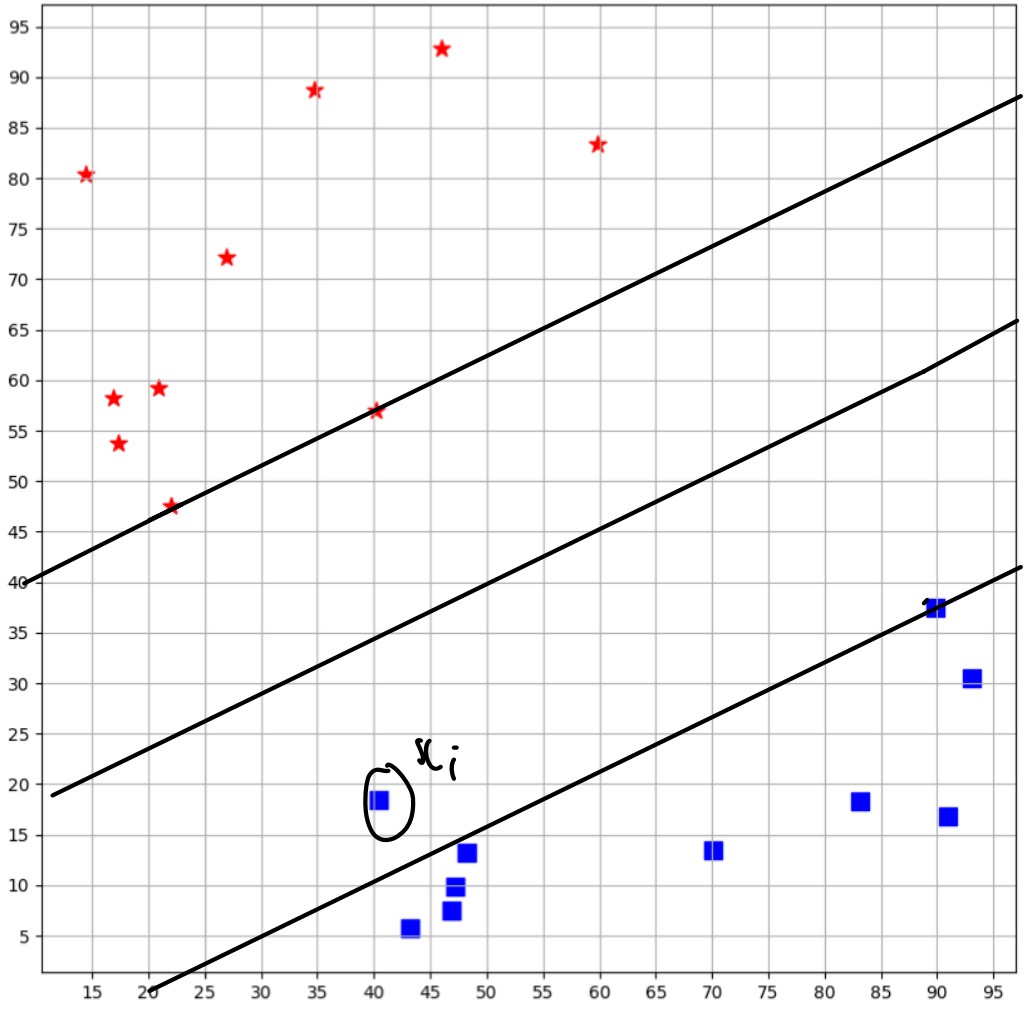
2. **(4 pts)** After obtaining the decision boundary in Equation 1, if we modify β while maintaining the restriction $\|\beta\|_2^2 = 1$, this would modify the minimum distance from the decision boundary to the origin.

   False. This is because as the norm $\|\beta\|_2^2 = 1$, this means that the distance from the decision boundary to the origin remains unchanged and thus, would not modify the minimum distance.

3. **(4 pts) SVM decision boundary.** Consider again an SVM whose decision boundary is obtained by Equations 1 and 2 like in the previous questions. Now, consider the training dataset with two training classes (signaled as squares and triangles) given by the scatter plot in the following figure. Draw the decision boundary for this dataset obtained by our SVM on it. The drawing needs to correctly separate the examples but there are multiple possible solutions. Each point of the line you draw must match the points of the true decision boundary within the error margin of 10 unit intervals along the x-axis and y-axis.


95
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95

4. **(4 pts) SVM decision boundary with removed example.** Now pick a single data point $\mathbf{x}_i$ that, if removed, would yield the maximum possible increase in the margin of a new SVM decision boundary over the dataset without $\mathbf{x}_i$. Indicate which point is $\mathbf{x}_i$ in the figure below (reproduction of last question). Draw the new decision boundary after $\mathbf{x}_i$'s removal on it. The drawing needs to correctly separate the remaining examples but there are multiple possible solutions. Each point of the line you draw must match the points of the true new decision boundary within the error margin of 10 unit intervals along the x-axis and y-axis.



Q5 (18 pts): Bagging and Boosting

Bagging is generally used to help stabilize classifiers with unstable learning algorithms (optimal score searching algorithms). A classifier has a stable learning algorithm if, by changing the training data, the predicted class labels in the test data don't change. For instance, the predictions of a decision tree might significantly change with a small change in the training data. This definition depends on the amount of data, of course. Classifiers that are unstable with 10^3 training examples may be stable with 10^9 examples. Bagging works by aggregating the answers of unstable classifiers trained over multiple training datasets. These multiple datasets are often not independent, generally sampled with replacement from the same training data.

Boosting works by converting weak classifier (very simple models) to strong ones (models that can describe complex relationships between the inputs and the class labels). A weak learner is a classifier whose output of an test example attributes x_i is only slightly correlated with its true class t_i. That is, the weak learner classifies the data better than random, but not much better than random. In boosting, weak learners are trained sequentially in a way that the current learner gives more emphasis to the examples that past learns made mistakes on.

1. **(8 pts)** Suppose we decide to use an SVM classifier with a small training dataset. Assume the network can perfectly fit the training data but we want to make sure it is accurate in our test data (without having access to the test data). Would you use boosting or bagging to help improve the classification accuracy? Describe what would be the problem of using the other approach.

I would use bagging to help improve the classification accuracy. This is because using boosting may lead to overfitting. This is because the network can perfectly fit the training data, there would not be misclassified points for boosting to focus on. Thus, using boosting may result in placing more emphasis on outliers in the test data and eventually, cause the model to overfit. However, by using bagging, where an aggregate of answers from unstable classifiers is used, overfitting is less likely and thus, I would use bagging over boosting to help improve the classification accuracy.

2. **(10 pts)** Read the following code and answer the following questions:

```
def fit(self, X, Y):

    if self.loss == "mse":
        loss = MSELoss()
    elif self.loss == "crossentropy":
        loss = CrossEntropyLoss()


    if self.classifier:
        Y = to_one_hot(Y.flatten())
    else:
        Y = Y.reshape(-1, 1) if len(Y.shape) == 1 else Y

    N, M = X.shape
    self.out_dims = Y.shape[1]
    self.learners = np.empty((self.n_iter, self.out_dims), dtype=object)
    self.weights = np.ones((self.n_iter, self.out_dims))
    self.weights[1:, :] *= self.learning_rate

    Y_pred = np.zeros((N, self.out_dims))
    for k in range(self.out_dims):
        t = loss.base_estimator()
        t.fit(X, Y[:, k])
        Y_pred[:, k] += t.predict(X)
        self.learners[0, k] = t

    for i in range(1, self.n_iter):
        for k in range(self.out_dims):
            y, y_pred = Y[:, k], Y_pred[:, k]
            neg_grad = -1 * loss.grad(y, y_pred)

            t = DecisionTree(
                classifier=False, max_depth=self.max_depth, criterion="mse"
            )

            t.fit(X, neg_grad)
            self.learners[i, k] = t

            step = 1.0
            h_pred = t.predict(X)
            if self.step_size == "adaptive":
                step = loss.line_search(y, y_pred, h_pred)

            self.weights[i, k] *= step
            Y_pred[:, k] += self.weights[i, k] * h_pred

def predict(self, X):
    Y_pred = np.zeros((X.shape[0], self.out_dims))
    for i in range(self.n_iter):
        for k in range(self.out_dims):
            Y_pred[:, k] += self.weights[i, k] * self.learners[i, k].predict(X)

    if self.classifier:
        Y_pred = Y_pred.argmax(axis=1)

    return Y_pred
```

(a) **(5 pts)** Which classifier is this? Why are the trees used in this classifier so shallow? Describe how the classifier works using pseudo-code.
Hint: It uses decision trees but this is not a decision tree classifier.

This is an AdaBoost classifier. The trees used in this classifier are shallow because they are decision stumps which are weak learners. This classifier works by the following pseudo-code:

AdaBoost():
  Input: Data set $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ where $y_i \in \{-1, 1\}$
    and weak learning algorithm A
  For $t = 1, 2, \ldots, T$
    Construct reweighting D_t for data
    Run $A(D_t)$ to get weak classifier h_t
    Calculate a weight $\alpha_t \in \mathbb{R}$

(b) **(5 pts)** What happens if we have mislabeled data? Why mislabeled data could be a problem? **Hint:** We are looking for an answer that uses the training weights of the examples at each iteration as a justification.

When we have mislabelled data, the model would mostly misclassify such data points, leading to an increase in the training weights of such examples on every iteration of AdaBoost. This could lead to the model overfitting to such mislabelled data. Eventually, this would lead to the model being biased to such mislabelled data, and potentially performing poorly on correctly labelled data.

Q5(a) continued

Output: classifier $\hat{h}$ defined as

$$\hat{h}(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Thus, D_t increases weight on points where h_{t-1} misclassifies, and decreases weight on points where h_{t-1} correctly classifies.

Programming Part

Overview

In this assignment, you will implement a one-hot encoder, a bagging classifier, and a cross-validation class. You will use these utilities to classify the bank loan dataset.

Files

You will fill in functions in `encoder.py`, `bagging.py`, and `cv.py`. After you finish, you will submit these as well as any other files mentioned in the assignment to Gradescope.

Asides from these three files, you will also find `utils.py` in the same folder. You do not need to modify it. `utils.py` contains some helper functions.

Packages

You will need the following packages for this assignment:

- `numpy`
- `pandas`
- `scikit-learn`

If you want, you can install `tqdm` to see a progress bar when running the autograder. This is not required for this class and might not be heavily focused on in future assignments, but it is a useful package to know about if you are doing any machine learning work in Python.

These packages should be installed if you have installed Anaconda. If you are not using Anaconda, you can install them using `pip` or `conda`. For example, you can run `pip install numpy` in the terminal to install `numpy`.

Note that for this assignment, you should **not** use any other packages. You may see some other packages in the skeleton code, but they are only used for testing and grading. If you use any other packages, you may lose points. If you are not sure whether you can use a package, please ask the course staff.

Evaluation

Unless otherwise specified, your code will be evaluated by a autograder on Gradescope using the same environment as detailed below. You should make sure your code runs without errors in this environment. Otherwise, you may lose points. You can submit your code to Gradescope as many times as you want. We will only grade the latest submission.

There will be two types of test cases in the autograder: public (local) and hidden. Public test cases are visible to you, and you can see the results after you submit your code. You can also run the public test cases locally by running `python <filename>.py` in the same folder as your code. Passing all public test cases **does not** guarantee you will get full credit for the assignment.

Hidden test cases, however, are not visible to you, and you will not be able to see the results until your grade is published. The final score you get for this assignment is the sum of the scores of all public and hidden test cases.

Getting Help

If you have any questions about this assignment, please contact the course staff for help, preferably during office hours. You can also post your questions on the course forum. However, please do not post any code publicly. When asking questions, you should describe your problem in words and post the relevant code snippet. Please avoid showing a screenshot of your code to TAs and ask "why my code is not working".

1 One-Hot Encoder

In this part, you will implement a one-hot encoder in `encoder.py`. You will use this encoder in Part 2 to encode the categorical features in the bank loan dataset.

In machine learning, we often need to encode categorical features into numerical features so that we can use them in our models. Suppose we have a dataset of students, and each student has a major. The major is a categorical feature. We can encode the major into a numerical feature by assigning a number to each major. For example, we can assign 0 to "Computer Science", 1 to "Mathematics", and 2 to "Chemistry".

However, this encoding is not ideal because it implies that "Computer Science" is closer to "Mathematics" than "Chemistry" if we calculate the "distance" between the two majors. This is not true because the majors are categorical features, and there is no natural ordering between them. To avoid this problem, we can use a one-hot encoder.

A one-hot encoder encodes a categorical feature into a vector of binary features. In our previous example, we can encode the major into a vector of length 3, where the first element is 1 if the major is "Computer Science" and 0 otherwise, the second element is 1 if the major is "Mathematics" and 0 otherwise, and the third element is 1 if the major is "Chemistry" and 0 otherwise. This encoding is called a one-hot encoding because only one element in the vector is 1 and the rest are 0.

1.1 One-Hot Encoder - Constructor

Under `class OneHotEncoder`, fill in the constructor so that it initializes the attribute `self.categories`, `self.data`, and `self.encoded_data` to `None`. We will initialize them later in the `fit` method.

1.2 One-Hot Encoder - Fit

Under `class OneHotEncoder`, fill in the `fit` method so that it takes in a numpy array `data` and save it to `self.data` and initializes `self.categories` to an empty list.

`data` is a 2D numpy array, where each row is a data point and each column is a feature. For example, if we have 3 data points and 2 features, `data` will be a 3 by 2 numpy array. You can assume that `data` only contains categorical features.

`self.categories` is a list of lists (or a list of numpy arrays). You should iterate through each column in `data` and find all the unique values in that column. For example, if the first column in `data` is ["Com-

puter Science", "Mathematics", "Mathematics"], then `self.categories[0]` should be ["Computer Science", "Mathematics"]. You can use `np.unique` to find the unique values in a numpy array.

You may assume that the input data is valid and does not contain any missing values or unknown categories. You do not need to handle these cases.

1.3 One-Hot Encoder - Encode

Under `class OneHotEncoder`, fill in the `encode` method so that it takes in a numpy array `data` and returns a numpy array `encoded_data`. To encode, you should iterate through each column in `data` and create a two dimensional numpy array `encoding_array` of shape `(len(data), len(self.categories[i]))`.

For example, if the first column in `data` is [["Computer Science"], ["Mathematics"], ["Mathematics"]], then `encoding_array` should be a 3 by 2 numpy array where the first row is `[1, 0]`, the second row is `[0, 1]`, and the third row is `[0, 1]`. You should use `self.categories` to find the index of the category in each row and set the corresponding element in `encoding_array` to 1.

You may assume that the input data is valid and does not contain any missing values or unknown categories. You do not need to handle these cases.

2 Bagging

In this part, you will implement a bagging classifier in `bagging.py`. You will use this classifier in Part 3 to train a bagging classifier on the bank loan dataset.

Bagging is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting. Although it is usually applied to decision tree methods, it can be used with any type of method.

Bagging leads to "improvements for unstable procedures", which include, for example, artificial neural networks, classification and regression trees, and subset selection in linear regression. On the other hand, it can mildly degrade the performance of stable methods such as K-nearest neighbors.

In this assignment, we will use bagging to improve the performance of a decision tree classifier. We will train multiple decision tree classifiers on different subsets of the training set and combine their predictions to make the final prediction.

2.1 Bagging - Constructor

Under `class BaggingClassifier`, fill in the constructor so that it takes in an integer `n_learners`, a string representing the encoding type for the decision tree, and a random seed `random_seed`. It should initialize the attribute `self.encoder` to an encoder object specified by the encoding type and initialize the attribute `self.random_seed` to the random seed. It should also create a list of `DecisionTreeClassifier` objects with `random_state` set to `self.random_seed` and save the list to `self.learners`.

2.2 Bagging - Preparing Samples

Under `class BaggingClassifier`, fill in the `_prepare_samples` method so that it takes in a numpy array `X` and returns a list of numpy arrays containing the indices of the samples randomly selected with replacement. You should use `self.random_seed` to initialize the random state before sampling. You may find `np.random.choice` useful.

2.3 Bagging - Fit

Under `class BaggingClassifier`, fill in the `fit` method so that it takes in a numpy array `X` and a numpy array `y` and trains `self.n_learners` decision tree classifiers on different subsets of the training set. You should use `self.encoder` to first fit and encode the training data. For each learner, you should use the `_prepare_samples` method to get the indices of the samples and use the `fit` method of the `DecisionTreeClassifier` class to train each decision tree classifier.

2.4 Bagging - Predict

Under `class BaggingClassifier`, fill in the `predict` method so that it takes in a numpy array X and returns a numpy array `y_pred` containing the predictions of the bagging classifier. You should use `self.encoder` to encode the test data first. For each learner, you should use the `predict` method of the `DecisionTreeClassifier` class to get the predictions of each decision tree classifier. You should then combine the predictions of all the decision tree classifiers to get the final prediction of the bagging classifier.You may find `np.bincount` and `np.argmax` useful.

3 Cross Validation

In this part, you will implement cross validation in `cv.py`. You will use this function to find the best hyperparameters for the bagging classifier you implemented in Part 2.

Cross validation is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice.

In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (testing dataset). The goal of cross validation is to define a dataset to "test" the model in the training phase (i.e., the validation dataset), in order to limit problems like overfitting, give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem), etc.

One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). To reduce variability, multiple rounds of cross-validation are typically performed using different partitions, and the validation results are averaged over the rounds.

The same kind of machine learning model can require different constraints, weights or learning rates to generalize different data patterns. These measures are called hyperparameters and have to be tuned so that the model can optimally solve the machine learning problem. Hyperparameters are set before the learning process begins while the values of other parameters are derived via training. The optimization or the tuning of hyperparameters is the problem of choosing a set of optimal hyperparameters for a learning algorithm.

3.1 Cross Validation - Constructor

Under `class CrossValidation`, fill in the constructor so that it takes in an integer `k` and a random seed `seed`. It should initialize the attribute `self.k` to `k` and initialize the attribute `self.random_seed` to the random seed.

3.2 Cross Validation - Split

Under `class CrossValidation`, fill in the `split` method so that it takes in a numpy array `X` and a numpy array `y` and splits the data into `self.k` folds. It should return a list of numpy arrays containing the samples in each fold. You should use `self.random_seed` to initialize the random state before splitting. Use `np.random.permutation` to randomly generate the indices of the samples and use `np.array_split` to split the indices into `self.k` folds.

3.3 Cross Validation - Combine Folds

Under `class CrossValidation`, fill in the `_combine_folds` method so that it takes in two lists of numpy arrays `X_folds` and `y_folds` and an integer `i`. It should combine all the folds except the `i`th fold into a training set and return the training set and the `i`th fold as a validation set. You should use `np.concatenate` to combine the folds.

3.4 Cross Validation - Score

Under `class CrossValidation`, fill in the `score` method so that it takes in two lists of numpy arrays `X` and `y`, the features and labels, and our Bagging `model`. It should use the model to predict the labels of the features and return the accuracy of the predictions.

3.5 Cross Validation - Cross Validate

Under `class CrossValidation`, fill in the `cross_validate` method so that it takes in two lists of numpy arrays `X` and `y`, the features and labels, and our Bagging `model`. It should use the `split` method to split the data into `self.k` folds and use the `_combine_folds` method to combine the folds into training and validation sets for each fold. It should then fit the model on the training set and use the `score` method to get the accuracy of the model on the validation set. Return the average accuracy of the model on the validation sets and the list of accuracies on the validation sets.

3.6 Cross Validation - Get Best Model

Under `class CrossValidation`, fill in the `get_best_model` method so that it takes in two lists of numpy arrays `X` and `y`, the features and labels, and a list of hyperparameters `params`. For each hyperparameter in `params`, it should create a Bagging model with that hyperparameter and use the `cross_validate` method to get the average accuracy of the model on the validation sets. It should then return the model with the best hyperparameter, the best hyperparameter, and the list of accuracies on the validation sets.

3.7 Cross Validation - Visualize

Under `class CrossValidation`, fill in the `plot_learning_curve` method so that it creates a plot of the learning curve of the model. The x-axis should be the hyperparameters and the y-axis should be the average accuracy of the model on the validation sets using each hyperparameter. You should use `plt.plot` to plot the learning curve. You should also use `plt.xlabel` and `plt.ylabel` to label the x-axis and y-axis. You should use `plt.title` to give the plot a proper title. You should use `plt.savefig` to save the plot as `learning_curve.png`. Return the plot.

Attach the plot to the Gradescope submission entry for plots. If plots are submitted as a part of the code submission and not submitted through the gradescope link for plots, then they won't be graded..

Submission

You will submit the following files to Gradescope:

- `encoder.py`
- `bagging.py`
- `cv.py`
- `learning_curve.png`