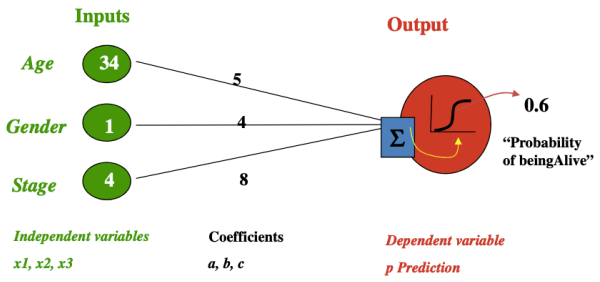
1. **Neural networks.** (20 points)

(a) (10 points) Consider a neural networks for a binary classification using sigmoid function for each unit. If the network has no hidden layer, explain why the model is equivalent to logistic regression.

Answer: The logistic regression model has the following structure, same as the structure of a neural networks without any hidden layer. Independent variable, dependent variable, coefficients in the logistic regression are equivalent to input variable, output variable, weights respectively in the neural network without any hidden layer.



Besides, the neural network using sigmoid function has the same P(Y|X) as the one of logistic regression.

$$P(Y|X) = \frac{1}{1 + \exp(-w^T x)}$$

(b) (10 points) Consider a simple two-layer network in the lecture slides. Given the cost function used to training the neural networks

$$\ell(w, \alpha, \beta) = \sum_{i=1}^{m} (y^i - \sigma(w^T z^i))^2$$

where $\sigma(x) = 1/(1 + e^{-x})$ is the sigmoid function. Show the that the gradient is given by

$$\frac{\partial \ell(w, \alpha, \beta)}{\partial w} = -\sum_{i=1}^{m} 2(y^i - \sigma(u^i))\sigma(u^i)(1 - \sigma(u^i))z^i.$$

where $z_1^i = \sigma(\alpha^T x^i)$, $z_2^i = \sigma(\beta^T x^i)$. Also find the gradient of ℓ with respect to α and β.

Answer: (in the following page)

This neural network has only one hidden ~~layers~~ layer, and this hidden layer has two neurons.

We will use $\sigma(x)' = \sigma(x)(1-\sigma(x))$ (eq. 1) when $\sigma(x) = 1/(1+e^{-x})$

Here is the proof: $\sigma(x)' = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{(1+e^{-x})}\frac{e^{-x}}{(1+e^{-x})} = \sigma(x)(1-\sigma(x))$

let $u^i = w^T z^i$

$$\frac{\partial L(w,\alpha,\beta)}{\partial w} = \frac{\partial \sum_{i=1}^{m}(y^i - \sigma(u^i))^2}{\partial w} = \frac{\partial \sum_{i=1}^{m}(y^i - \sigma(u^i))^2}{\partial u^i}\frac{\partial u^i}{\partial w}$$

$$= \sum_{i=1}^{m} 2(y^i - \sigma(u^i))(-1)\sigma(u^i)'\frac{\partial w^T z^i}{\partial w}$$

(equa. 1)

$$= -\sum_{i=1}^{m} 2(y^i - \sigma(u^i))\left[\sigma(u^i)(1-\sigma(u^i))\right]\left[z^i\right] \quad \text{Done.}$$

let $z_1^i = \sigma(\alpha^T x^i)$, $v^i = \alpha^T x^i$

$$\frac{\partial L(w,\alpha,\beta)}{\partial \alpha} = \frac{\partial L(w,\alpha,\beta)}{\partial u^i}\cdot\frac{\partial u^i}{\partial z_1^i}\frac{\partial z_1^i}{\partial \alpha}$$

$$= -\sum_{i=1}^{m} 2(y^i - \sigma(u^i))\,\sigma(u^i)(1-\sigma(u^i))\left[\frac{\partial w^T z^i}{\partial z_1^i}\right]\left[\frac{\partial \sigma(v^i)}{\partial \alpha}\right]$$

$$= -\sum_{i=1}^{m} 2(y^i - \sigma(u^i))\,\sigma(u^i)(1-\sigma(u^i))\left[\frac{\partial w^T(z_1^i + z_2^i)}{\partial z_1^i}\right]\left[\frac{\partial \sigma(v^i)}{\partial v^i}\cdot\frac{\partial v^i}{\partial \alpha}\right]$$

$$= -\sum_{i=1}^{m} 2(y^i - \sigma(u^i))\,\sigma(u^i)(1-\sigma(u^i))\left[w\right]\left[\sigma(v^i)(1-\sigma(v^i))\,x^i\right] \quad (\text{eq. 1})$$

For gradient L respect to β, it has ~~the~~ a similar form, with only one difference that $v^i = \beta^T x^i$.

2. **Comparing SVM and simple neural networks.** (40 points)

This question is to implement and compare **SVM and simple neural networks** for the same datasets we tried for the last homework. We suggest to use Scikit-learn, which is a commonly-used and powerful Python library with various machine learning tools. But you can also use other similar libraries in other programming languages of your choice to perform the tasks.

You may use a neural networks function sklearn.neural_network with hidden_layer_sizes=(5, 2). Tune the step size so you have reasonable results. You may use svc and tune the penalty term C to get reasonable results.

Part One (Divorce classification/prediction). (20 points)

We will compare using the same dataset as the last homework, which is about participants who completed the personal information form and a divorce predictors scale.

The data is a modified version of the publicly available at `https://archive.ics.uci.edu/ml/datasets/Divorce+Predictors+data+set` (by injecting noise so you will not replicate the results on uci website). There are 170 participants and 54 attributes (or predictor variables) that are all real-valued. The dataset **q3.csv**. The last column of the CSV file is label y (1 means "divorce", 0 means "no divorce"). Each column is for one feature (predictor variable), and each row is a sample (participant). A detailed explanation for each feature (predictor variable) can be found at the website link above. Our goal is to build a classifier using training data, such that given a test sample, we can classify (or essentially predict) whether its label is 0 ("no divorce") or 1 ("divorce").

Build two classifiers using SVM and a simple neural networks. First random shuffle the data set. Then use the first 80% data for training and the remaining 20% for testing. If you use scikit-learn you can use train_test_split to split the dataset.

(a) (15 points) Report testing accuracy for each of the two classifiers. Comment on their performance: which performs better and make a guess why it performs better in this setting.

Answer:

Classifier Name	Tuned Parameter	Does random seed impact result?	Testing Accuracy
SVM	C = 0.1	No	94.1%
ANN	Step size = 0.01	Yes, if seed = 3	94.1%
ANN	Step size = 0.01	Yes, if seed = 54	44.1%

For SVM tuning, I did a grid search of different C values (0.001, 0.01, 0.1, 1, 10, 100, 1000), and found C equal to 0.1 to be the best one.

For ANN tuning, I did a grid search of different step sizes (0.00001, 0.0001, 0.001, 0.01, 0.1), and found step size equal to 0.01 to be the best one.

For SVM performance, it showed a high accuracy (94.1%), 32 correct assignments out 34 testing data points.

For ANN performance, different random weight initializations would lead to different testing accuracy. After proper random seeding, ANN showed a high accuracy (94.1%), same as the performance of SVM. Without a proper random seeding, ANN showed a low accuracy (44.1%).

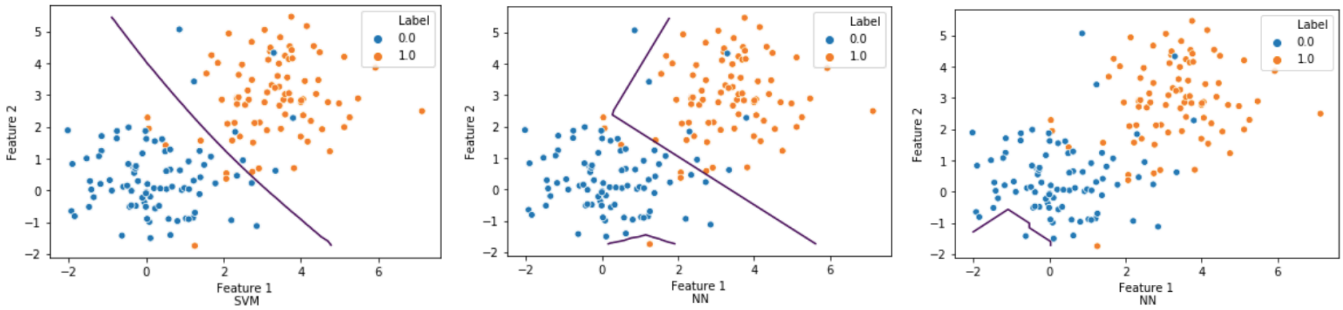
In conclusion, (linear) SVM and ANN showed a same performance of accuracy (94.1%). Because of the simplicity of dataset and independent features in this data set, linear SVM and

simple-structure ANN both performed well. Given more data, both SVM and ANN would likely perform better.

(Note: The q3.csv contains scaled data and ANN requires a scaled data input. I did another min-max scaling of q3.csv data, and obtained a low accuracy of 67.6% using ANN.)

(b) (15 points) Use the first two features to train two new classifiers. Plot the data points and decision boundary of each classifier. Comment on the difference between the decision boundary for the two classifiers. Please clearly represent the data points with different labels using different colors.

Answer:



Picture: Data points and decision boundary of each classifier
From left to right: SVM, ANN (Accuracy 94%), ANN (Accuracy 44%)

For liner SVM, the decision boundary is a straight line. For ANN model (accuracy 94%), decision boundary is more complex with two parts (two polylines). For ANN model (accuracy 44%), decision boundary is also complex with one polyline with multiple vertices.

Part Two (Handwritten digits classification). (20 points) Repeat the above part (a) using the **MNIST Data** in our previous homework. Here, give "digit" 6 label $y = 1$, and give "digit" 2 label $y = 0$. All the pixels in each image will be the feature (predictor variables) for that sample (i.e., image). Our goal is to build classifiers such that given a new testing sample, we can tell it is a 2 or a 6. Using the first 80% of the samples for training and remaining 20% for testing. Report the classification accuracy on testing data, for each of the two classifiers. Comment on their performance: which performs better and make a guess why they perform better in this setting.

Answer:

Classifier Name	Tuned Parameter	Does random seed impact result?	Testing Accuracy
SVM	C = 10	No	99.5%
ANN	Step size = 0.1	Yes, if seed = 3	99.0%
ANN	Step size = 0.1	Yes, if seed = 41	48.2%

For SVM tuning, I did a grid search of different C values (0.001, 0.01, 0.1, 1, 10, 100, 1000), and found C equal to 10 to be the best one.

For ANN tuning, I did a grid search of different step sizes (0.00001, 0.0001, 0.001, 0.01, 0.1), and found step size equal to 0.1 to be the best one.

For SVM performance, it showed a high accuracy (99.5%), 396 correct assignments out 398 testing data points.

For ANN performance, different random weight initializations would lead to different testing accuracy. After proper random seeding, ANN showed a high accuracy (99.0%), slightly lower than the performance of SVM. Without a proper random seeding, ANN showed a low accuracy (48.2%).

In conclusion, (linear) SVM showed a slightly better performance than ANN (accuracy 99.5% vs 99.0%). SVM showed a good classification job of separating samples in high dimension, though features were correlated in the handwritten digit context. The simple structure of ANN increased the test error considering the large feature number (784).

3. **AdaBoost.** (40 points)

Consider the following dataset, plotting in Figure 1. The first two coordinates represent the value of two features, and the last coordinate is the binary label of the data.

$$X_1 = (-1, 0, +), X_2 = (-0.5, 0.5, +), X_3 = (0, 1, -), X_4 = (0.5, 1, -),$$
$$X_5 = (1, 0, +), X_6 = (1, -1, +), X_7 = (0, -1, -), X_8 = (0, 0, -).$$

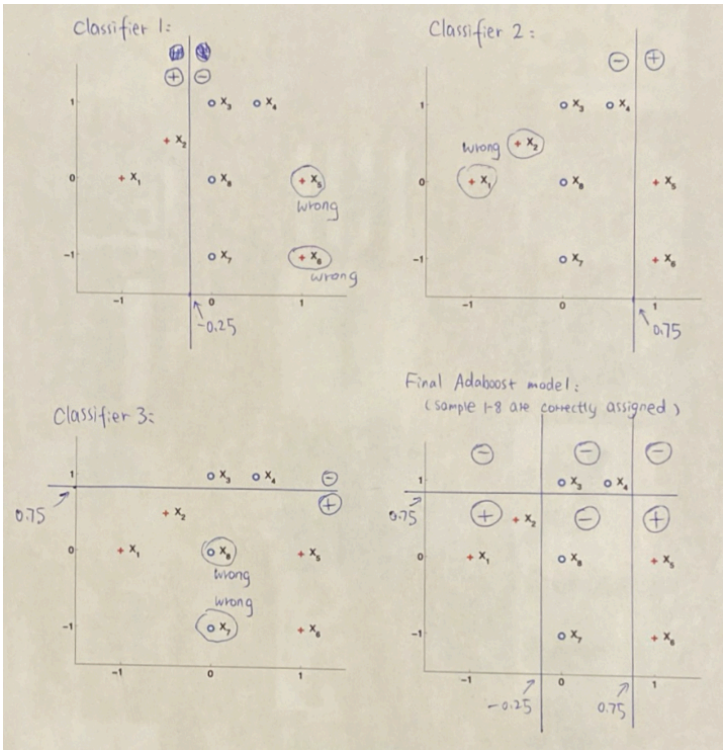
In this problem, you will run through $T = 3$ iterations of AdaBoost with decision stumps (axis-aligned half planes) as weak learners.

(a) (20 points) For each iteration $t = 1, 2, 3$, compute ϵ_t, α_t, Z_t, D_t by hand (i.e., show all the calculation steps) and draw the decision stumps on Figure 1. Recall that Z_t is the normalization factor to ensure that the weights D_t sum to one. (*Hint: At each iteration, you may specify any reasonable decision rule h_t as you would like.*)

Answer:

Here is the summary table (see calculation details in the following page) and drawings:

t	Epsilon_t	Alpha_t	Z_t	Dt(1)	Dt(2)	Dt(3)	Dt(4)	Dt(5)	Dt(6)	Dt(7)	Dt(8)
1	0.2500	0.5493	0.8661	0.1250	0.1250	0.1250	0.1250	0.1250	0.1250	0.1250	0.1250
2	0.1666	0.8050	0.7454	0.0833	0.0833	0.0833	0.0833	0.2500	0.2500	0.0833	0.0833
3	0.1000	1.0990	0.5999	0.2501	0.2501	0.0500	0.0500	0.1499	0.1499	0.0500	0.0500



Round 1: D1(i) = 0.125 for i = 1, 2, …, 8

First classifier, minus if x >= -0.25: only data 5 and 6 are wrongly assigned

Epsilon_1 = D1(5) + D1(6) = 0.125 + 0.125 = 0.25

Alpha_1 (amount of say) = 0.5 * ln((1-epsilon_1)/epsilon_1) = 0.5493

For wrongly assigned sample, D2(i)_before_/Z1 = D1(i) * e^alpha

For correctly assigned sample, D2(i)_before_/Z1 = D1(i) * e^(-alpha)

See the following table for the D2 calculation

Round 2: **Second classifier, plus if x >= 0.75: only data 1 and 2 are wrongly assigned**

Epsilon_2 = D2(1) + D2(2) = 0.0833 + 0.0833 = 0.1666

Alpha_2 (amount of say) = 0.5 * ln((1-epsilon_2)/epsilon_2) = 0.8050

See the following table for the D3 calculation

Round 3: **Third classifier, minus if y >= 0.75: only data 7 and 8 are wrongly assigned**

Epsilon_3 = D3(7) + D3(8) = 0.05+ 0.05 = 0.1

Alpha_3 (amount of say) = 0.5 * ln((1-epsilon_3)/epsilon_3) = 1.099

See the following table for the Z3 calculation

Final label:

Final Label = the sign of [Alpha_1 * label_assignment_1 + Alpha_2* label_assignment_2 + Alpha_3* label_assignment_3]

Sample	D1	Label Assignment	Value of e^alpha or e^(-alpha)	D2_before_/Z1	Z1	D2	Label Assignment	Value of e^alpha or e^(-alpha)	D3_before_/Z2	Z2	D3	Label Assignment	Value of e^alpha or e^(-alpha)	D4_before_/Z3	Z3	Final Label Calculation	True Label	Training error
1	0.125	1	0.5774	0.0722		0.0833	-1	2.2370	0.1864		0.2501	1	0.3332	0.0833		0.8433	1	
2	0.125	1	0.5774	0.0722		0.0833	-1	2.2370	0.1864		0.2501	1	0.3332	0.0833		0.8433	1	
3	0.125	-1	0.5774	0.0722		0.0833	-1	0.4471	0.0373		0.0500	-1	0.3332	0.0167		-2.4533	-1	
4	0.125	-1	0.5774	0.0722	0.8661	0.0833	-1	0.4471	0.0373	0.7454	0.0500	-1	0.3332	0.0167	0.5999	-2.4533	-1	0
5	0.125	-1	1.732	0.2165		0.2500	1	0.4471	0.1118		0.1499	1	0.3332	0.0500		1.3547	1	
6	0.125	-1	1.732	0.2165		0.2500	1	0.4471	0.1118		0.1499	1	0.3332	0.0500		1.3547	1	
7	0.125	-1	0.5774	0.0722		0.0833	-1	0.4471	0.0373		0.0500	1	3.001	0.1500		-0.2553	-1	
8	0.125	-1	0.5774	0.0722		0.0833	-1	0.4471	0.0373		0.0500	1	3.001	0.1500		-0.2553	-1	
															Note:	Correct Label	Wrong Label	

(b) (20 points) What is the training error of AdaBoost? Give a one-sentence reason for why AdaBoost outperforms a single decision stump.

Answer:

The AdaBoost training error is 0 because all 8 samples are correctly assigned. Please see the previous page for the calculation process. Here is the sample 8 final label assignment example:

Final Label of 8

= the sign of [Alpha_1 * label_assignment_1 + Alpha_2* label_assignment_2 + Alpha_3* label_assignment_3]

= the sign of [0.5493 * -1 + 0.8050 * -1 + 1.0990 * 1]

= the sign of [-0.2553]

= minus

= True label

One-sentence reason:

A decision stump is a weak learner based on just one single input feature, and AdaBoost is an ensemble method to give much better results by assigning weights to weak leaners and combining weak learners' results.