CSE 512 Machine Learning: Homework 2

Department of Computer Science
Stony Brook University

- There are 4 questions on this assignment. The first question involves coding. Do *not* attach your code to the writeup. Instead, zip and submit your code electronically on Blackboard (Bb). Name your .zip file as **[your_SBU_name].zip**, e.g. vivek.zip
- The assignment is due at 5:30 PM (beginning of class) on **Tuesday Mar 24, 2015**.
- Do not forget to put both your name and SBU ID on *each* page of your submission.
- If you have any questions, please direct your question first to the TA, then the instructor.
- You may *discuss* the questions with fellow students, *however* you must always write your own solutions and must acknowledge whom you discussed the question with. Do not copy from other sources, share your work with others, or search for solutions on the web.

1 Boosting [50 points]

The details of Adaboost are in *Robert E. Schapire. The boosting approach to machine learning: An overview.* In Nonlinear Estimation and Classification. Springer, 2003. `https://www.cs.princeton.edu/picasso/mats/schapire02boosting_schapire.pdf`

The algorithm details of Schapire's tutorial differ slightly from those in the textbook. Both will yield the same results, but the internal values of weights will differ. The proofs of 1.1 follow Schapire's tutorial. Please use Schapire's algorithm for Problem 1.1 and 1.2. You may implement either and should obtain identical results for Problem 1.3.

1.1 [20 Points] Analyzing the training error of boosting

Consider the AdaBoost algorithm you saw in class. In this question we will try to analyze the training error of boosting.

1. *(4 points)* Given a set of m examples, (x_i, y_i) (where $y_i \in \{-1, +1\}$ is the class label of x_i), $i = 1, \ldots, m$, let $h_t(x)$ be the weak classifier obtained at step t, and let $\alpha_t \in \mathbf{R}$ be its weight. Recall that the final classifier can be written as:

$$H(x) = \text{sign}(f(x)), \text{ where } f(x) = \sum_{t=1}^{T} \alpha_t h_t(x).$$

   where T is the number of base learners.

   Show that the training error of the final classifier can be bounded from above by an exponential loss function (namely):

$$\frac{1}{m} \sum_{i=1}^{m} I(H(x_i) \neq y_i) \leq \frac{1}{m} \sum_{i=1}^{m} \exp(-f(x_i) y_i),$$

   where $I(a = b)$ is the indicator function which is equal to 1 if $a = b$, and 0 otherwise.

   Hint: $e^{-x} \geq 1 \Leftrightarrow x \leq 0$.

2. *(4 points)*Recall that the weight for example i at step $t+1$ is recursively defined as

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

with $D_1(i) = \frac{1}{m}$ being the base case. Z_t is a normalization factor so that D_{t+1} is a distribution. Using this recursive definition prove the following:

$$\frac{1}{m}\sum_{i=1}^{m}\exp(-f(x_i)y_i) = \prod_{t=1}^{T} Z_t, \tag{1}$$

where Z_t, the normalization factor to make D_{t+1} a distribution is defined below:

$$Z_t = \sum_{i=1}^{m} D_t(i)\exp(-\alpha_t y_i h_t(x_i)). \tag{2}$$

Hint: remember that $e^{\sum_i g_i} = \prod_i e^{g_i}$,$D_1(i) = \frac{1}{m}$, *and that* $\sum_{i=1}^{m} D_t(i) = 1$ *for all* t.

3. *(4 points)* Equation 1 suggests that the training error $\epsilon_{training}$ can be reduced rapidly by greedily optimizing Z_t at each step. You have shown that the error is bounded from above:

$$\epsilon_{training} \leq \prod_{t=1}^{T} Z_t.$$

Observe that $Z_1, \ldots, Z_{t-1}$ are determined by the first $(t-1)$ rounds of boosting, and we cannot change them on round t. A greedy step we can take to minimize the training error bound on round t is to minimize Z_t.

In this question, you will prove that for binary weak classifiers, Z_t from Equation 2 is minimized by picking α_t as:

$$\alpha_t^* = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right), \tag{3}$$

where ϵ_t is the training error of weak classifier h_t for the weighted dataset, namely:

$$\epsilon_t = \sum_{i=1}^{m} D_t(i) I(h_t(x_i) \neq y_i).$$

where I is the indicator function. For this proof, only consider the simplest case of binary classifiers, i.e. the output of $h_t(x)$ is binary, $\{-1, +1\}$.

For this special class of classifiers, first show that the normalizer Z_t can be written as:

$$Z_t = (1-\epsilon_t)\exp(-\alpha_t) + \epsilon_t \exp(\alpha_t).$$

Hint: consider the sums over correctly and incorrectly classified examples separately.

Now, prove that the value of α_t that minimizes this definition of Z_t is given by Equation 3.

4. *(4 points)* Prove that for the above value of α_t

$$Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$$

Furthermore, let $\epsilon_t = \frac{1}{2} - \gamma_t$, prove that

$$Z_t \leq \exp(-2\gamma_t^2)$$

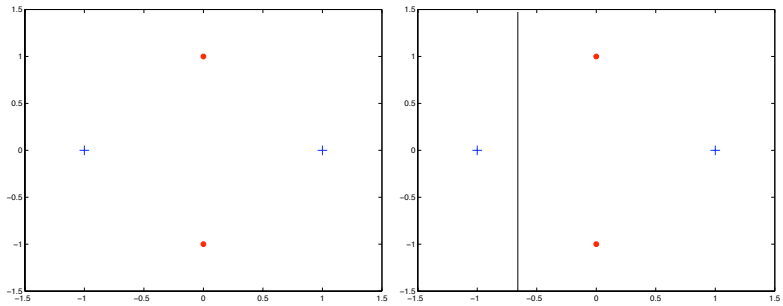

Figure 1: a) Toy data in Question 1. b) h_1 in Question 1

where $\gamma_t \leq 0.5$

Hint: $\log(1-x) \leq -x$ *for* $0 < x \leq 1$.

Therefore

$$\epsilon_{training} \leq \prod_t Z_t \leq \exp(-2\sum_t \gamma_t^2)$$

Finally prove that, if each weak classifier is slightly better than random, so that for each t, $\gamma_t \geq \gamma$, for some $\gamma > 0$, then the training error drops exponentially fast in T, i.e.

$$\epsilon_{training} \leq \exp(-2T\gamma^2)$$

5. *(4 points)* Show that in each round of boosting, there always exists a weak classifier h_t such that its training error on the weighted dataset $\epsilon_t \leq 0.5$. Also show that for $\epsilon_t = 0.5$ the training error can get "stuck" above zero.

   Hint: $D_t(i)$*s do not change over* t.

1.2 [5 Points] Adaboost on a toy dataset

Now we will apply Adaboost to classify a toy dataset. Consider the following dataset in Figure 1a). The dataset consists of 4 points, $(X_1 : 0, -1, -)$, $(X_2 : 1, 0, +)$, $(X_3 : -1, 0, +)$ and $(X_4 : 0, 1, -)$.

1. Use simple decision stumps as weak classifiers. (For description of decision stumps, refer to Problem 1.3) Now for $T = 4$, show how Adaboost works for this dataset. For each timestep remember to compute the following numbers:

   $$\epsilon_t, \alpha_t, Z_t, D_t(i)\ \forall i,$$

   Also for each timestep draw your weak classifier. For example h_1 can be as shown in 1b).

2. What is the training error of Adaboost?

3. Is the above dataset linearly separable? Explain why Adaboost does better than a decision stump in the above dataset.

1.3 [25 Points] Implementation

Implement the AdaBoost algorithm (page 658 in the Bishop's book) using a decision stump as the weak classifier.

AdaBoost trains a sequence of classifiers. Each classifier is trained on the same set of training data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, m$, but where each example $\{\mathbf{x}_i, y_i\}$ is weighed differently. The weight for $(\mathbf{x}_i, y_i)$ at iteration t is $D_t(i)$. At iteration t, a classifier, $h_t(\mathbf{x}) \to \{-1, 1\}$, is trained to minimize the weighted classification error, $\sum_{i=1}^{m} D_t(i) \cdot I(h_t(\mathbf{x}_i) \neq y_i)$, where I is the indicator function (0 if the predicted and actual labels match, and 1 otherwise). The overall prediction of the AdaBoost algorithm is a linear combination of these classifiers, $H_T(\mathbf{x}) = sign(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}))$. *Note:* The textbook uses $w_i \equiv D_t(i)$.

A decision stump is a decision tree with a single node. It corresponds to a single threshold in one of the features, and predicts the class for examples falling above and below the threshold respectively, $h_t(\mathbf{x}) = C_1 I(x^j \geq c) + C_2 I(x^j < c)$, where x^j is the j^{th} component of the feature vector $\mathbf{x}$. Unlike in class, where we split on Information Gain, for this algorithm split the data based on the weighted classification accuracy described above, and find the class assignments $C_1, C_2 \in \{-1, 1\}$, threshold c, and feature choice j that maximize this accuracy.

1. *(15 points)* Evaluate your AdaBoost implementation on the Bupa Liver Disorder dataset that is available for download from `http://www.cs.stonybrook.edu/~leman/courses/15CSE512/hws/hw2-data.tar.gz`. The classification problem is to predict whether an individual has a liver disorder (indicated by the selector feature) based on the results of a number of blood tests and levels of alcohol consumption. Use 90% of the dataset for training and 10% for testing. Average your results over 50 random splits of the data into training sets and test sets. Limit the number of boosting iterations to 100. In a single plot show:
   - average training error after each boosting iteration
   - average test error after each boosting iteration
2. *(5 points)* Using all of the data for training, display the selected feature component j, threshold c, and class label C_1 of the decision stump $h_t(\mathbf{x})$ used in each of the first 10 boosting iterations ($t = 1, 2, ..., 10$)
3. *(5 points)* Using all of the data for training, in a single plot, show the empirical cumulative distribution functions of the margins $y_i f_T(\mathbf{x}_i)$ after T=10, 50 and 100 iterations respectively, where $f_T(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$. Notice that in this problem, before calculating $f_T(\mathbf{x})$, you should normalize the α_ts so that $\sum_{t=1}^{T} \alpha_t = 1$. This is to ensure that the margins are between -1 and 1.

   Hint: the empirical cumulative distribution function of a random variable X at x is the proportion of times $X \leq x$.

2 Model Selection and Cross-Validation [20 points]

2.1 [13 points] Bias-Variance Trade-off

We define a term *True Risk* as the expected squared error between the predicted model f and the true model f^* as:

$$R(f) = E_X[(f(X) - f^*(X))^2]$$

If we assume zero noise variance, it is shown that this risk can be expressed in terms of Bias-Variance Trade-off, i.e:

$$R(f) = E_X[(f(X) - Y)^2] = Variance + Bias^2$$

where $Y = f^*(X)$.

Analogously we define the *parameter risk* of our estimated parameter $\hat{\theta}$ (obtained using MLE or MAP or density estimator, etc) and the true parameter θ as:

$$R(\theta, \hat{\theta}) = E_\theta[(\hat{\theta} - \theta)^2] = Var_\theta(\hat{\theta}) + bias^2$$

where $bias = E_\theta[\hat{\theta}] - \theta$. Here it is important to note that E_θ denotes the expectation over the randomness of data where each data sample x is drawn from the conditional distribution specified by $P(X|\theta)$.

Let $X_1, \ldots, X_n \sim Bernoulli(p)$, represent a sequence of random variables where each X_i represents the outcome of an independent coin flip, $X_i = 1$ indicates flipping a *Head*, and $X_i = 0$ indicates flipping a *Tail*, with p being the probability of getting a *Head*.

Consider two estimators for p: $\hat{p}_1 = \frac{1}{n}\sum_i X_i$ (the MLE estimate) and $\hat{p}_2 = \frac{\sum_i X_i + \alpha}{\alpha + \beta + n}$ (the mean of the posterior Beta distribution $P(p|D)$, where D is the observed data and we use $Beta(\alpha, \beta)$ as a prior for p).

1. (1 point) Compute the risk of $\hat{p}_1$, i.e. $R(p, \hat{p}_1)$'
2. (4 points) Compute the risk of $\hat{p}_2$, i.e. $R(p, \hat{p}_2)$
3. (2 points) Which estimator $\hat{p}_1$ or $\hat{p}_2$ would you prefer when there is less data and which would you prefer, when there is more data? (*Hint*: Consider bias-variance tradeoff)
4. (3 points) Given a particular n, find values of α and β that will make the risk of $\hat{p}_2$ constant (independent of p).
5. (3 points) Using Hoeffding's inequality, and knowing that $0 \leq X_i \leq 1$ (each X_i is bounded), find an upper bound of $|\hat{p}_1 - p|$ with a probability of at least $1 - \gamma$.

2.2 [7 points] Model Selection

Let $X \in \{0, 1\}$ be a random variable that denotes the result of a coin toss ($X = 0$ for *Tails*, $X = 1$ for *Heads*). The coin is potentially biased, so that *Heads* occurs with probability θ_1. Suppose that someone else observes the coin flip and reports to you the outcome which is a realization of the random variable Y. But this person is unreliable and only reports the result correctly with probability θ_2; i.e., $P(Y|X; \theta_2)$ is given by

	$Y = 0$	$Y = 1$
$X = 0$	θ_2	$1 - \theta_2$
$X = 1$	$1 - \theta_2$	θ_2

Assume that θ_2 is independent of θ_1 and X.

1. (1 point) Write down the joint probability distribution $P(X, Y|\theta)$ as a 2×2 table, in terms of $\theta = (\theta_1, \theta_2)$. *Hint: write down how the likelihood function $P(X, Y|\theta)$ factorizes.*
2. (1 point) Suppose we have the following dataset: $\mathcal{D} = [(1,1), (1,0), (0,0), (1,0), (1,1), (0,0), (0,1), (1,1), (1,0), (0,0)]$ where the first element of each tuple represents the outcome of the coin toss and the second element of each tuple represents the corresponding value reported by the person.

   What are the MLEs for θ_1 and θ_2? Justify your answer. What is $P(\mathcal{D}|\hat{\theta}, M_2)$ where M_2 denotes this 2-parameter model?

3. (1 point) Now consider a model with 4 parameters, $\theta = (\theta_1, \theta_2, \theta_3, \theta_4)$, representing $P(X, Y|\theta)$ such that $P(X = x, Y = y) = \theta_{x,y}$. (Only 3 of these parameters are free to vary, since they must sum to one.)

X	Y	θ
0	0	$\theta_{0,0} \equiv \theta_1$
0	1	$\theta_{0,1} \equiv \theta_2$
1	0	$\theta_{1,0} \equiv \theta_3$
1	1	$\theta_{1,1} \equiv \theta_4$

What is the MLE of θ? What is $P(\mathcal{D}|\hat{\theta}, M_4)$ where M_4 denotes this 4-parameter model?

4. (4 points) Suppose we are not sure which model is correct. We compute the *Leave-One-Out Cross Validation* (LOOCV) log likelihood of the 2-parameter model M_2 and the 4-parameter model M_4 as follows:

$$L(M) = \sum_{i=1}^{m} \log P(X = x_i, Y = y_i | M, \hat{\theta}(D_{-i}))$$

and $\hat{\theta}(D_{-i})$ denotes the MLE computed on $\mathcal{D}$ excluding row i and there are m examples. Which model will LOOCV pick and why? *Hint: Notice how the table of counts changes when you omit each training case one at a time.*

3 Neural Networks [15 points]

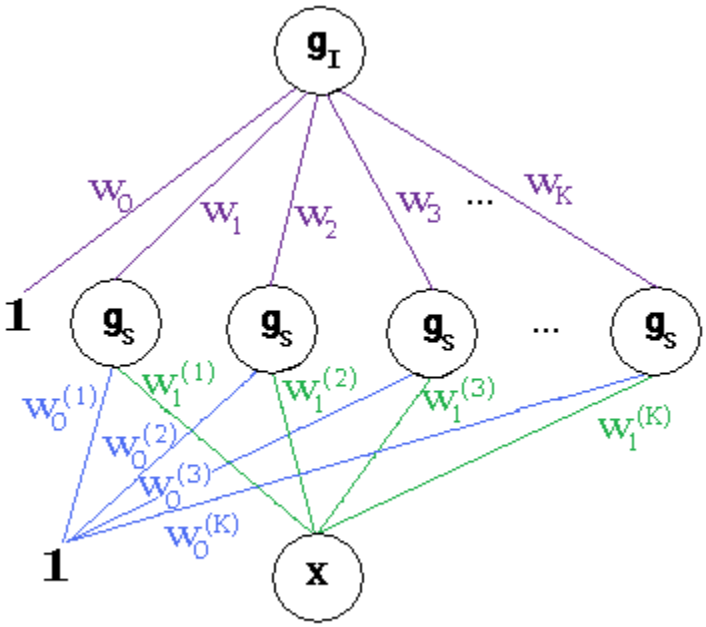

Figure 2: Neural net drawing in Question 3.

In this question, you will prove that a neural network with a single hidden layer can provide an arbitrarily close approximation to any 1-dimensional bounded smooth function.

Suppose that you have two types of activation functions at hand:

- identity
$$g_{\mathbf{I}}(x) = x,$$
- step function
$$g_{\mathbf{S}}(x) = \begin{cases} 1 & \text{if } x \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

So, for example, the output of a neural network with one input x, a single hidden layer with K units having step function activations, and a single output with identity activation can be written as $out(x) = g_{\mathbf{I}}(w_0 + \sum_{k=1}^{K} w_k g_{\mathbf{S}}(w_0^{(k)} + w_1^{(k)} x))$, and can be drawn as in Figure 2.

1. *(3 points)* Consider the step function $u(x)$ in Figure 3. Construct a neural network with one input x and one hidden layer whose response is $u(x)$. That is, if $x < x_0$, the output of your network should be a, whereas if $x \geq x_0$, the output should be b. Draw the structure of the neural network, specify the activation function for each unit (either $g_{\mathbf{I}}$ or $g_{\mathbf{S}}$), and specify the values for all weights (in terms of a, b and x_0).

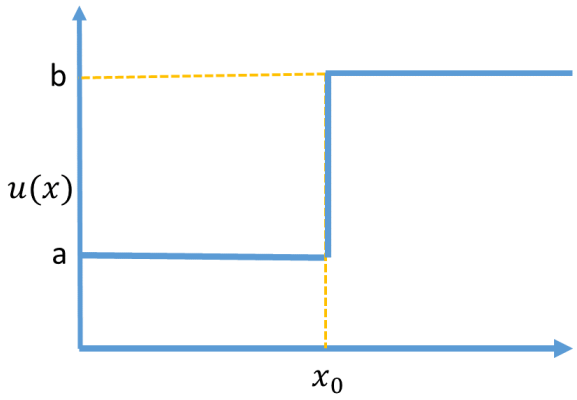

Figure 3: Step function in Question 3.

2. *(3 points)* Now consider the indicator function $I_{[a,b)}(x)$:
$$I_{[a,b)}(x) = \begin{cases} 1, & \text{if } x \in [a, b); \\ 0, & \text{otherwise.} \end{cases}$$
Construct a neural network with one input x and one hidden layer whose response is $yI_{[a,b)}(x)$, for given real values y, a, and b; that is, its output is y if $x \in [a, b)$, and 0 otherwise. Draw the structure of the neural network, specify the activation function for each unit (either $g_{\mathbf{I}}$ or $g_{\mathbf{S}}$), and specify the values for all weights (in terms of a, b, and y).

3. *(9 points)* You are now given any function $f(x)$ whose domain is $[C, D)$, for real values $C < D$. Suppose that the function is Lipschitz continuous[1]; that is,

$$\forall x, x' \in [C, D), \quad |f(x') - f(x)| \leq L|x' - x|, \tag{5}$$

for some constant $L \geq 0$ (called the Lipschitz constant of f). Observe that this implies that the first derivative(gradient) of the function is upper bounded. Use the intuition from the previous part to construct a neural network with one hidden layer that approximates this function within $\epsilon > 0$; that is, $\forall x \in [C, D), \quad |f(x) - out(x)| \leq \epsilon$, where $out(x)$ is the output of your neural network given input x. Your network should use only the activation functions $g_{\mathbf{I}}$ and $g_{\mathbf{S}}$ given above. You need to specify the number K of hidden units, the activation function for each unit, and a formula for calculating each weight w_0, w_k, $w_0^{(k)}$, and $w_1^{(k)}$, for each $k \in \{1, 2, \ldots, K\}$; these may be specified in terms of C, D, L, and ϵ, as well as the values of $f(x)$ evaluated at a finite number of x values of your choosing (please explicitly specify which x values you use). You do *not* need to draw the network or explicitly write the $out(x)$ function. Why does your neural network attain the given accuracy ϵ?

Hint: Lipschitz continuous means that points close together have similar f values. Can you find a set of points so that every $x \in [C, D)$ is close to at least one of them? How close does it need to be to guarantee the f values are within ϵ of each other?

4 Instance-based Learning [20 points]

4.1 [5 Points] Kernel Regression

In this question we are given a set of n examples, $(\mathbf{x_i}, y_i)$ (where $\mathbf{x_i}$ is the input, and y_i is output), $i = 1, \ldots, n$, and $\mathbf{x_i} \in \mathbf{R}^d$ and $y_i \in \mathbf{R}$. Here d is the number of dimensions.

Kernel regression is a widely used non-parametric regression method that applies a kernel function $K(.)$ to smooth the data y_i:

$$\hat{r}(\mathbf{x}) = \frac{\sum_{i=1}^{n} K(\mathbf{x} - \mathbf{x_i}) y_i}{\sum_{i=1}^{n} K(\mathbf{x} - \mathbf{x_i})} = \sum_{i=1}^{n} w_i(\mathbf{x}, \mathbf{x_i}) y_i$$

where $w_i(\mathbf{x}, \mathbf{x_i}) = \frac{K(\mathbf{x} - \mathbf{x_i})}{\sum_{j=1}^{n} K(\mathbf{x} - \mathbf{x_j})}$, and $\hat{r}(\mathbf{x})$ is the estimator of y (a scalar) at the point $\mathbf{x}$ (which is d-dimensional).

In the following questions we are going to look at 1-dimensional ($d = 1$) classification data with binary classes, i.e., $y_i \in \{0, 1\}$.

1. (3 points) Suppose that $x_i = i$ for all $i = 1, \ldots, n$, and to predict which class any new point $x \in [0, n]$ belongs to, we use the decision rule

$$\hat{y} = I(\hat{r}(x) > 0.5)$$

where I is the indicator function that equals to 1 if the expression within I is true, and 0 if not true. Can you think of a kernel function K such that this decision rule gives the same answer as the k-nearest neighbor algorithm? (*Hint: the function $K(x)$ should be defined at some fixed range like $[a, b)$; you can ignore the case where x is near the margins, i.e. close to 0 or n.*)

[1] Lipschitz continuity is a smoothness condition that limits how fast a function can change.

2. (2 points) In general, if training data is drawn from some marginal distribution $p(x)$, we are unable to find a nice-looking kernel to simulate k-NN. One way to solve this problem is to use "locally weighted regression (LWR)" instead of kernel regression. The LWR is very similar to kernel regression except that we only calculate the weighted sum of k points near the given new point x:

$$\hat{r}(x) = \sum_{i|x_i \in \text{k-NN of } x} w_i(x, x_i) y_i$$

Now consider the case of weighted k-nearest neighbor (WKNN). Weighted k-NN penalizes the vote of every point z within the k-NN range by $d(x, z)$, the distance from x to z, meaning that having a point in class j in the k-neighbor will increase the vote for this class by $1/d(x, z)$.

What should we put in for $w_i(x, x_i)$ so that the decision rule $\hat{y} = I(\hat{r}(x) > 0.5)$ gives the same answer as (unweighed) k-NN?.

4.2 [15 Points] k-NN Classifier and Decision Surfaces

1. (**10 points**) **Decision Surfaces**

Suppose, you have the following classifiers: (a) Logistic regression, (b) Gaussian Naive Bayes, (c) 1-Nearest-Neighbor, and (d) 10-Nearest-Neighbor. Now, if you apply each of these classifiers on the following two datasets displayed in Figure 4, what would be the possible decision surface/boundaries. A hand-drawn decision boundary would be enough for each of the cases. But make sure that you briefly describe the important features of the boundary, e.g., (i) what made you choose that boundary, (ii) how good of a classifier it is, (iii) and anything else noteworthy.

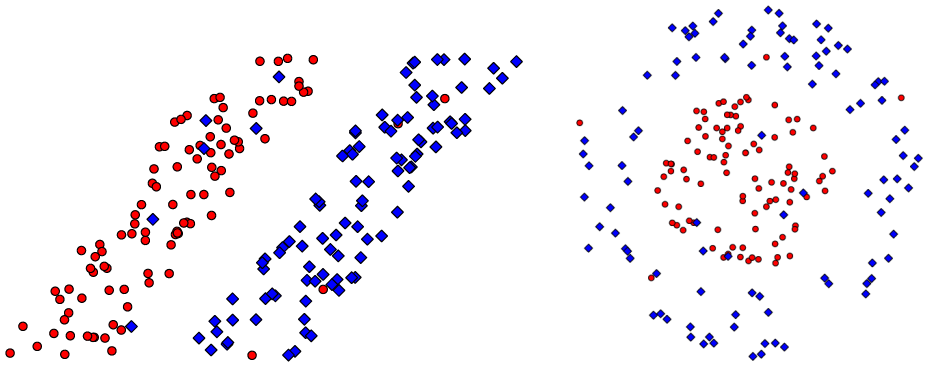

Figure 4: a) Toy data for Question 4.2. 1. b) Toy data for Question 4.2. 1

Each image contains 200 points (2-dimensional), 100 from each of the two well-separated clusters; however, a few labels have been flipped due to noisy observation. For your convenience, these two images are made available in the document section of the Blackboard.

2. **[5 points] Limitation of k-NN** Consider n sample points $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ independently and uniformly drawn from a p-dimensional zero-centered unit ball $B := \{\mathbf{x} \mid \sqrt{\mathbf{x}^\top \mathbf{x}} \leq 1, \mathbf{x} \in \mathbf{R}^p\}$. In this problem you will study the size of the 1-nearest neighborhood of the origin $\mathbf{0}$ and how it changes with

respect to the dimension p, thereby gaining intuition about the downside of k-NN in high dimension. More precisely, consider the distance from $\mathbf{0}$ to its nearest neighbor in the sample:

$$d^* := \min_{1 \le i \le n} \sqrt{\mathbf{x}_i^\top \mathbf{x}_i},$$

which is a random variable since the sample is random.

(a) (1.5 point) In the special case $p = 1$, what is the cumulative distribution function (cdf) of d^*, i.e., $P(d^* \le t)$ for $0 \le t \le 1$?

(b) (1.5 point) In the general case $p \in \{1, 2, 3, \ldots\}$, what is the cdf of d^*? (*Hint*: You may find the following fact useful: the volume of a p-dimensional ball with radius r is $\frac{(r\sqrt{\pi})^p}{\Gamma(p/2+1)}$, where $\Gamma(\cdot)$ is the Gamma function.)

(c) (2 point) With the cdf you derived in Problem 2.2b, answer the following question: How large should the sample size n be such that with probability at least 0.9, the distance d^* from $\mathbf{0}$ to its nearest neighbor is less than $1/2$, i.e., half way from $\mathbf{0}$ to the boundary of the ball? Your answer should be a function of p. From this function, what can you infer? Can you identify the downside of k-NN in terms of n and p?