HOMEWORK 7

>>NAME HERE<<
>>ID HERE<<

Instructions: Use this latex file as a template to develop your homework. Please submit a single pdf to Canvas. Late submissions may not be accepted. You can choose any programming language (i.e. python, R, or MATLAB). Please check Piazza for updates about the homework.

1 Kernel SVM [15 pts]

Consider the following kernel function defined over $z, z' \in Z$:

$$k(z, z') = \begin{cases} 1 & \text{if } z = z', \\ 0 & \text{otherwise.} \end{cases}$$

1. (5 pts) Prove that for any integer $m > 0$, any $z_1, \ldots, z_m \in Z$, the $m \times m$ kernel matrix $K = [K_{ij}]$ is positive semi-definite, where $K_{ij} = k(z_i, z_j)$ for $i, j = \{1 \ldots m\}$. (Let us assume that for $i \neq j$, we have $z_i \neq z_j$.)
To prove that the kernel matrix K is positive semi-definite, we need to show that for any non-zero vector $f \in \mathbb{R}^m$, $f^T K f \geq 0$. We can compute the product $f^T K f$ as follows:

$$f^T K f = \sum_{i=1}^{m} \sum_{j=1}^{m} f_i K_{ij} f_j = \sum_{i=1}^{m} \sum_{j=1}^{m} f_i k(z_i, z_j) f_j.$$

If $i = j$, then $k(z_i, z_j) = 1$ because $z_i = z_j$. If $i \neq j$, then $k(z_i, z_j) = 0$ because $z_i \neq z_j$. We have:

$$\begin{aligned} f^T K f &= \sum_{i=1}^{m} \sum_{j=1}^{m} f_i k(z_i, z_j) f_j \\ &= \sum_{i=1}^{m} f_i k(z_i, z_i) f_i + \sum_{i=1}^{m} \sum_{j=1, j \neq i}^{m} f_i k(z_i, z_j) f_j \\ &= \sum_{i=1}^{m} f_i f_i + 0 \\ &= \sum_{i=1}^{m} f_i^2 \geq 0 \end{aligned}$$

2. (5 pts) Given a training set $(z_1, y_1), \ldots, (z_n, y_n)$ with binary labels, the dual SVM problem with the above kernel k will have parameters $\alpha_1, \ldots, \alpha_n, b \in \mathbb{R}$. (Assume that for $i \neq j$, we have $z_i \neq z_j$.) The predictor for input z takes the form

$$f(z) = \sum_{i=1}^{n} \alpha_i y_i k(z_i, z) + b.$$

Recall the label prediction is $\mathrm{sgn}(f(z))$. Prove that there exists $\alpha_1, \ldots, \alpha_n, b$ such that f correctly separates the training set. In other words, k induces a feature space rich enough such that in it any training set can be linearly separated.
First, we have to create a feature space that is rich enough for n training samples, so we compute the n-dimensional space. We have a mapping function $\Phi : z \to \Phi(z)$, where $\Phi(z_i) \in \mathbb{R}^n$ is an unit vector with 1 at the i^{th} and 0 at other positions. Therefore $\Phi(z_i)\Phi(z_j) = 1$ if i = j, otherwise is 0.

The kernel $k(z_i, z)$ can be defined as a dot product of $\Phi(z_i)$ and $\Phi(z)$. Using this, we have the predictor for input z

$$f(z) = \sum_{i=1}^{n} \alpha_i y_i k(z_i, z) + b = \sum_{i=1}^{n} \alpha_i y_i \Phi(z_i) \cdot \Phi(z) + b$$

To correctly separate the training set, we define $\alpha_1, \ldots, \alpha_n, b \in \mathbb{R}$ such that

$$f(z_j) = \sum_{i=1}^{n} \alpha_i y_i k(z_i, z_j) + b = \sum_{i=1}^{n} \alpha_i y_i \Phi(z_i) \cdot \Phi(z_j) + b = \alpha_j y_j + b$$

since $\Phi(z_i)\Phi(z_j) = 1$ if i = j, otherwise is 0. Set $\alpha_j = 1, b = 0$, we have

$$f(z_j) = y_j$$

$$\text{sgn}(f(z_j)) = \text{sgn}(y_j) = y_j.$$

This proves f correctly separates the training set for some $\alpha_1, \ldots, \alpha_n, b \in \mathbb{R}$.

3. (5 pts) How does that f predict input z that is not in the training set?
Since z is not in the training set, there is no z_i such that z_i = z. Therefore, we have:

$$f(z) = \sum_{i=1}^{n} \alpha_i y_i k(z_i, z) + b = b$$

.

$$\text{sgn}(f(z)) = \text{sgn}(b)$$

This implies that the prediction of input z is solely based on b. Therefore, as long as we can compute $k(z, z')$, kernel SVM will work to predict z.

Comment: One useful property of kernel functions is that the input space Z does not need to be a vector space; in other words, z does not need to be a feature vector. For all we know, Z can be turkeys in the world. As long as we can compute $k(z, z')$, kernel SVM works on turkeys.

2 Game of Classifiers [50 pts]

2.1 Implementation

Implement the following models in choice of your programming language. Include slack variables in SVM implementation if needed. You can use autograd features of pytorch, tensorflow etc. or derive gradients on your own. (But don't use inbuilt models for SVM, Kernel SVM and Logistic Regression from libraries)

- Implement Linear SVM (without kernels).
- Implement Kernel SVM, with options for linear, rbf and polynomial kernels. You should keep the kernel parameters tunable (e.g. don't fix the degree of polynomial kernels but keep it as a variable and play with different values of it. Is Linear SVM a special case of Kernel SVMs?
- Implement Logistic Regression with and without kernels (use same kernel as Question 1).

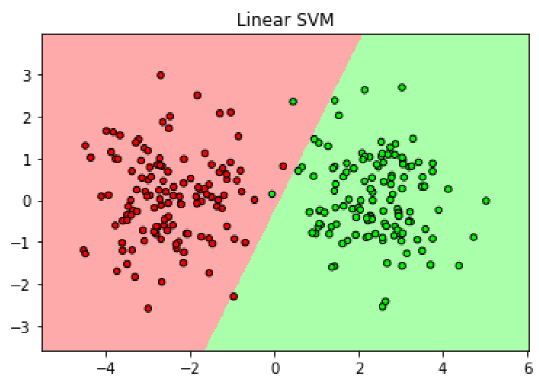
2.2 Evaluation on Synthetic Dataset

2.2.1 Synthetic Dataset-1 (20 pts)

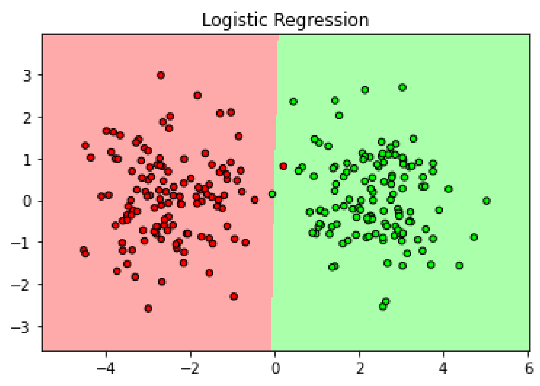
Generate 2-D dataset as following,
Let $\mu = 2.5$ and I_2 be the 2×2 identity matrix. Generate points for the positive and negative classes respectively from $\mathcal{N}([\mu, 0], I_2)$, and $\mathcal{N}([-\mu, 0], I_2)$. For each class generate 750 points, (1500 in total). Randomly create train, validation and test splits of size 1000, 250, 250 points respectively. Do the following with this dataset:

- (5 pts) Train your Linear SVM, Logistic Regression models and report decision boundaries, test accuracies. **The code for this exercise is in 'hw7.ipynb'**

	Test Accuracy
Linear SVM	0.988
Logistic Regression	0.992



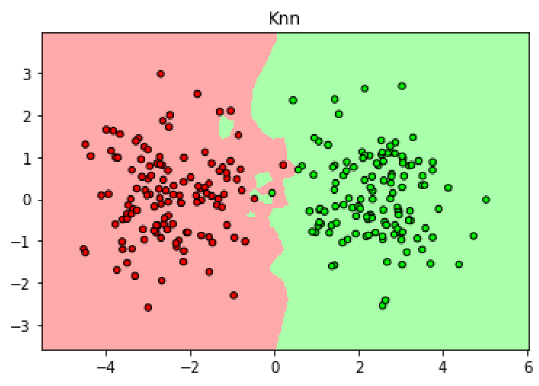
(a) epoch 1

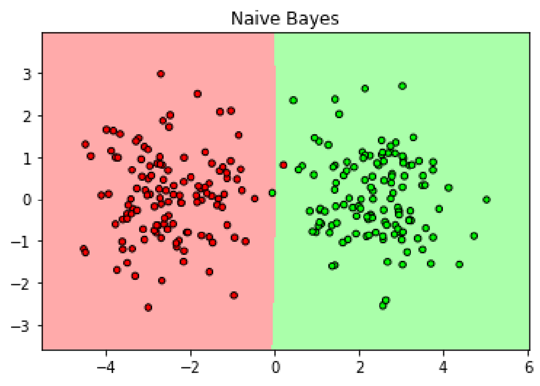


(b) epoch 50

- (5 pts) Show decision boundaries with K-NN and Naive Bayes Classifiers. (You can use library implementations or implement from scratch. Figure out the hyper-parameters using the validation set.)

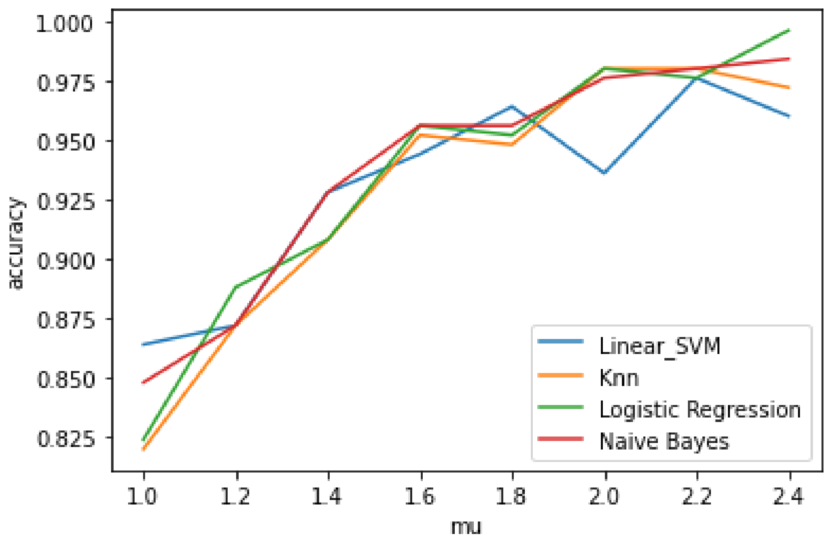
	Test Accuracy
K-NN	0.992
Naive Bayes	0.992





- (5 pts) Repeat the process by varying μ from 1.0 to 2.4 with step size of 0.2, for each value of μ obtain test accuracies of the models and plot (μ on x-axis and test accuracy on y-axis). (You will have a curve for each of the 4-classifiers mentioned above)

Classifiers/mu	1	1.2	1.4	1.6	1.8	2.0	2.2	2.4
Linear SVM	0.864	0.872	0.928	0.944	0.964	0.936	0.976	0.96
K-NN	0.82	0.872	0.908	0.952	0.948	0.98	0.98	0.972
Logistic Regression	0.824	0.888	0.908	0.956	0.952	0.98	0.976	0.996
Naive Bayes	0.848	0.872	0.928	0.956	0.956	0.976	0.98	0.984



- (5 pts) What are your conclusions from this exercise?
  From this exercise, we can draw several conclusions:

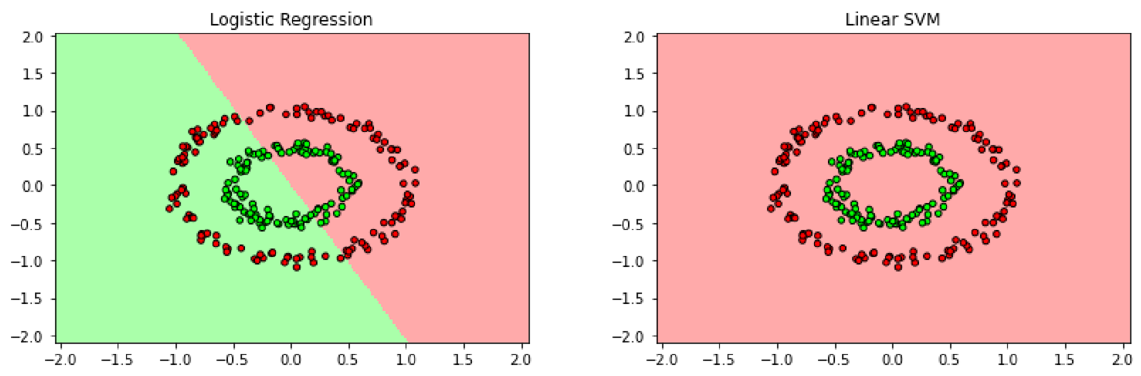
  1. The performance of Linear SVM, Logistic Regression, K-NN, and Naive Bayes classifiers is relatively similar, with test accuracies mostly ranging from 0.8 to 1.0. This indicates that all four classifiers can be effective for solving this particular 2-D synthetic dataset classification problem.
  2. As the value of μ increases, the test accuracies of all four classifiers generally improve. This is because as μ increases, the separation between the two Gaussian distributions for positive and negative classes becomes more prominent, making the linearly classification task easier.
  3. Logistic Regression and Naive Bayes classifiers tend to achieve slightly higher test accuracies compared to Linear SVM and K-NN classifiers, especially for larger values of μ.

2.2.2 Synthetic Dataset-2 (20 pts)

Generate 1500 data points from the 2-D circles dataset of sklearn (`sklearn.datasets.make_circles`). Randomly create train, validation and test splits of size 1000, 250, 250 points respectively. Evaluate the above classifiers on this setting.

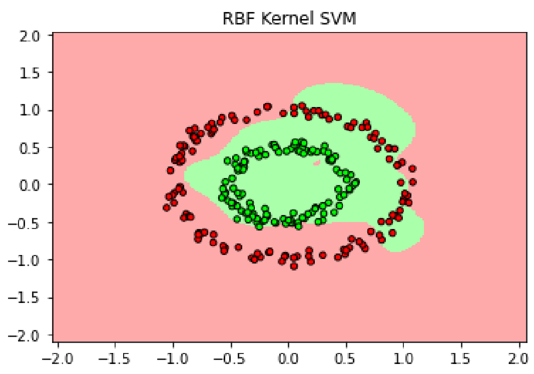
- (5 pts) Show decision boundaries for Linear SVM and Logistic Regression classifiers.

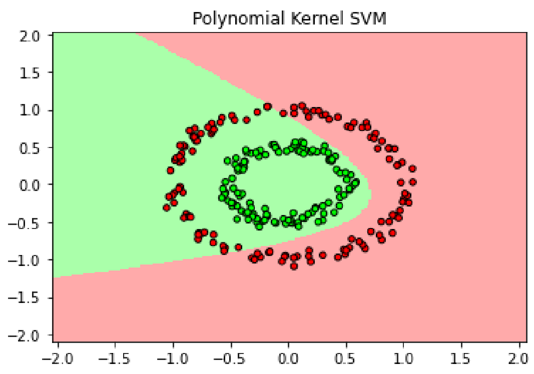
	Test Accuracy
Linear SVM	0.47600001096725464
Logistic Regression	0.26

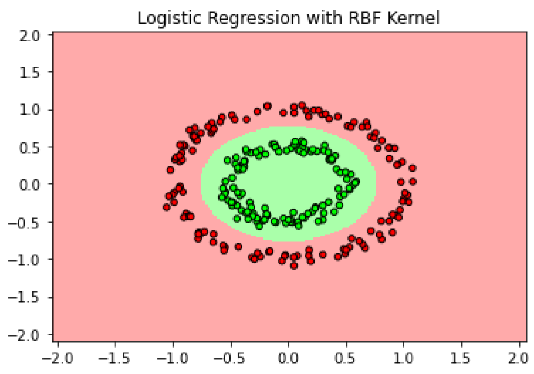


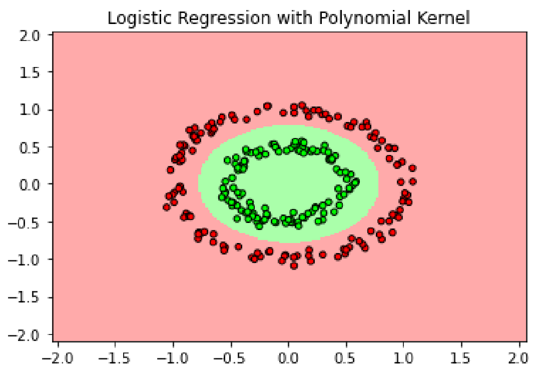
- (5 pts) Show decision boundaries for Kernel SVM and Kernel Logistic Regression (use rbf, polynomial kernels). Try different values of hyperparameters, report results with whichever works the best.

	Test Accuracy
RBF Kernel SVM	0.8240000009536743
Polynomial Kernel SVM	0.7680000066757202
RBF Kernel Logistic Regression	1.0
Polynomial Kernel Logistic Regression	1.0



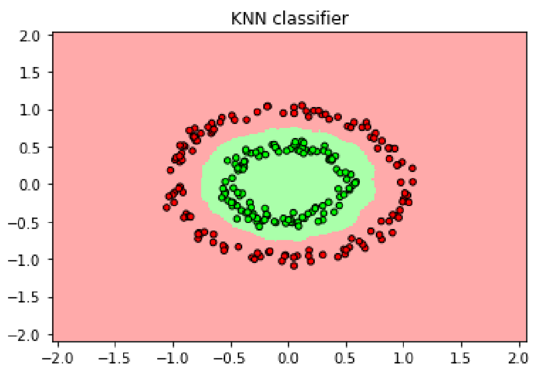


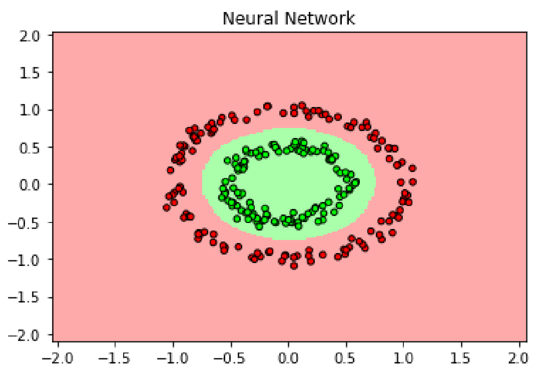




- (5 pts) Train Neural Network from HW4, and K-NN classifiers on this dataset and show decision boundaries. (You can use library implementation for these classifiers).

	Test Accuracy
K-NN	1.0
Neural Network	1.0





- (5 pts) What are your conclusions from this exercise?
  Based on the results presented in this exercise, the following conclusions can be drawn:

  1. Linear SVM and Logistic Regression classifiers are not suitable for the given 2-D circles dataset as they have low test accuracy (0.476 and 0.26, respectively). This is because these linear classifiers cannot capture the complex, non-linear decision boundaries required for this problem.

2. Kernel SVM (RBF and Polynomial) are doing quite well on this dataset to classify non-linearly data. Few more tunes are needed to do with these classifiers in order to maximize their accuracy
3. Kernel Logistic Regression (RBF and Polynomial) classifiers perform significantly better than their linear counterparts, achieving perfect test accuracy of 1.0. These classifiers can model non-linear decision boundaries, making them more suitable for the given dataset.
4. K-NN and Neural Network classifiers also achieve perfect test accuracy (1.0) on this dataset. The K-NN classifier captures the local structure of the dataset, while the Neural Network can learn complex, non-linear decision boundaries. Both methods are suitable for this dataset.

In conclusion, for the 2-D circles dataset, non-linear classifiers such as Kernel SVM, Kernel Logistic Regression, K-NN, and Neural Networks are more effective than linear classifiers like Linear SVM and Logistic Regression. This exercise highlights the importance of selecting appropriate classifiers based on the characteristics and complexity of the dataset.

2.3 Evaluation on Real Dataset (10 pts)

Lets put all this to some real use. For this problem use the the Wisconsin Breast Cancer dataset. You can download it from sklearn library(`sklearn.datasets.load_breast_cancer`)

- (5 pts) Do all the points of section 2.2.2 on this dataset. Since this is high-dimensional data, so you don't have to show the decision boundaries. Just report test accuracies for these classifiers and discuss your findings.

	Test Accuracy
Linear SVM	0.8771929740905762
Logistic Regression	0.5964912280701754
Kernel SVM rbf	0.859649121761322
Kernel SVM polynomial	0.8859649300575256
Kernel rbf Logistic Regression	0.9824561403508771
Kernel polynomial Logistic Regression	0.956140350877193
Neural Network	0.9557522123893806
K-NN	0.9473684210526315

The classifier kernel rbf Logistic Regression is the most suitable one for this problem, while Logistic Regression seem to not work well on this data, since the data is complex and hard for Logistic Regression to classify correctly.

- (5 pts) In addition, you also want to figure out the important features which determine the class. Which regularization will you use for this? Upgrade your SVM, Kernel SVM implementation to include this regularization. Discuss what are the important features that you obtain by running your regularized SVM on this dataset. (You might have to normalize this dataset before training any classifier).
To find the important features, I trained an L1-regularized SVM and looked at the absolute values of the weights. The features corresponding to the largest absolute weights are the most important. The indices of the important features that I obtain by running regularized SVM on this dataset:

$$[6, 16, 0, 20, 26, 1, 9, 24, 4, 13]$$

. These are accuracy after training with these features are:

	Test Accuracy
Linear SVM SVM with the selected features	0.7168141603469849
Kernel SVM RBF with the selected features	0.8684210777282715
Kernel SVM Polynomial with the selected features	0.8245614171028137

3 VC dimension [20 pts]

1. (7 pts) Let the input $x \in \mathcal{X} = \mathbb{R}$. Consider $\mathcal{F} = \{f(x) = \text{sgn}(ax^2 + bx + c) : a, b, c \in \mathbb{R}\}$, where $\text{sgn}(z) = 1$ if $z \geq 0$, and 0 otherwise. What is $VC(\mathcal{F})$? Prove it.

Shatter Coefficient and VC dimension

The main intuition behind VC theory is that, although a collection of classifiers may be infinite, using a finite set of training data to select a good rule effectively reduces the number of different classifiers we need to consider. We can measure the effective size of class $\mathcal{F}$ using *shatter coefficient*. Suppose we have a training set $D_n = \{(x_i, y_i)\}_{i=1}^n$ for a binary classification problem with labels $y_i = \{-1, +1\}$. Each classifier in $\mathcal{F}$ produces a binary label sequence

$$\big(f(x_1), \cdots, f(x_n)\big) \in \{-1, +1\}^n$$

There are at most 2^n distinct sequences, but often no all sequences can be generated by functions in $\mathcal{F}$. Shatter coefficient $\mathcal{S}(\mathcal{F}, n)$ is defined as the maximum number of labeling sequences the class $\mathcal{F}$ induces over n training points in the feature space $\mathcal{X}$. More formally,

$$\mathcal{S}(\mathcal{F}, n) = \max_{x_1,\cdots,x_n \in \mathcal{X}} \left| \Big\{ \big(f(x_1), \cdots, f(x_n)\big) \Big\} \in \{-1, +1\}^n, f \in \mathcal{F} \right|$$

where $|\cdot|$ denotes the number of elements in the set. The *Vapnik-Chervonenkis (VC) dimension* $V(\mathcal{F})$ of a class $\mathcal{F}$ is defined as the largest integer k such that $\mathcal{S}(\mathcal{F}, k) = 2^k$.
To find the VC dimension of the function class $\mathcal{F} = f(x) = \text{sgn}(ax^2 + bx + c) : a, b, c \in \mathbb{R}$, we need to find the largest integer k for which we can shatter a set of k points. We do this by first attempting to shatter a set of 3 points, and then showing that we cannot shatter a set of 4 points.

Shattering k=3 points, we can find $2^3 = 8$ possible labellings : Suppose $x_1 < x_2 < x_3$, let us consider 3 points: $x_1, x_2, x_3 \in \mathcal{X} = \mathbb{R}$. Define $\Delta = b^2 - 4ac$, and $ax_i^2 + bx_i + c = z_i$, we'll find a function $f(x) = \text{sgn}(ax^2 + bx + c)$ that can create the label sequence:

(a) $(1, 1, 1)$: We choose $\Delta \leq 0$ and $a > 0$ to ensure every x is ≥ 0
(b) $(1, 1, 0)$: We choose $a = 0$ and b such that we have $bx_3 + c < 0$ and $bx_1 + c \geq 0$ and $bx_2 + c \geq 0$
(c) $(1, 0, 1)$: We choose $\Delta = 0$ and $a > 0$ to ensure the quadratic function has positive z_1, z_3 at the two extremes for x_1, x_3 and negative z in the middle curve for x_2
(d) $(1, 0, 0)$: We choose $a = 0$ and b such that $bx_1 + c \geq 0$, $bx_1 + c < 0$, and $bx_3 + c < 0$
(e) $(0, 1, 1)$: We choose $a = 0$ and b such that $bx_1 + c < 0$, $bx_1 + c \geq 0$, and $bx_3 + c \geq 0$
(f) $(0, 1, 0)$: We choose $\Delta = 0$ and $a < 0$ to ensure the quadratic function has negative z_1, z_3 at the two extremes for x_1, x_3 and positive z in the middle curve for x_2
(g) $(0, 0, 1)$: We choose $a = 0$ and b such that $bx_1 + c < 0$, $bx_1 + c < 0$, and $bx_3 + c \geq 0$
(h) $(0, 0, 0)$: We choose $\Delta < 0$ and $a < 0$ to ensure every $x < 0$

Since $x_1 < x_2 < x_3$, we can observe that the quadratic function $ax^2 + bx + c$ that can label all 8 cases; therefore, $\mathcal{F} = f(x) = \text{sgn}(ax^2 + bx + c) : a, b, c \in \mathbb{R}$ shatters k = 3 points.

Cannot shatter 4 points:

Supposed $x_1 < x_2 < x_3 < x_4$, we consider 4 points: $x_1, x_2, x_3, x_4 \in \mathcal{X} = \mathbb{R}$. We want to show that we cannot shatter these 4 points with a function of the form $f(x) = \text{sgn}(ax^2 + bx + c)$. We consider the label sequence $(1, -1, 1, -1)$. To satisfy this label sequence, we need the following inequalities:

$ax_1^2 + bx_1 + c \geq 0$
$ax_2^2 + bx_2 + c < 0$
$ax_3^2 + bx_3 + c \geq 0$
$ax_4^2 + bx_4 + c < 0$

This is for the curve change the sign 3 times, which leads to the function $ax^2 + bx + c = 0$ needing to have 3 solutions. This is impossible for quadratic functions since they have at most 2 solutions. Therefore, there is no function that can map $f(x_i)$ to y_i in the above case.

Conclusion:

Since we can shatter 3 points but not 4 points with functions from $\mathcal{F}$, we can conclude that the VC dimension of $\mathcal{F}$ is 3, i.e., $VC(\mathcal{F}) = 3$.

2. (7 pts) Suppose there are n points $(x_1, \cdots, x_n) \in \mathbb{R}$. Let $\mathcal{F}$ be the collection of 1-d linear classifiers: for each $t \in [0,1]$, $f_t \in \mathcal{F}$ labels $x \le t$ as -1 and $x > t$ as $+1$, or vice-versa. What is the shatter coefficient $S(\mathcal{F}, n)$? What is VC-dimension $V(\mathcal{F})$? How can you get it from shatter coefficient?
To find the shatter coefficient $S(\mathcal{F}, n)$ for the given function class $\mathcal{F}$ of 1-dimensional linear classifiers, we need to calculate the maximum number of distinct label sequences that the class $\mathcal{F}$ can generate for a given set of n points $x_1, x_2, x_3, \ldots, x_n$.

Case 1: $f_t(x) = 1$ for $x > t$ and $f_t(x) = -1$ for $x \le t$

Consider the sequence in the ascending order $x_1 < x_2 < x_3 < \cdots < x_n$, in which we have n + 1 positions to put t to separate those in the left of t is $\le t$, else $> t$. So we have n+1 distinct labels

Case 2: $f_t(x) = -1$ for $x > t$ and $f_t(x) = 1$ for $x \le t$

Consider the sequence in the ascending order $x_1 < x_2 < x_3 < \cdots < x_n$, we can do a similar procedure as above and have n+1 distinct labels. However, when putting t at the last position, we got the same result as t at the first position of case 1, and vice versa. So we have n-1 distinct labelings and also distinct to the elements of the set above.

Therefore, we have $S(\mathcal{F}, n) = 2n$ distinct labelings in total.

The VC-dimension is the largest integer k such that the shatter coefficient $S(\mathcal{F}, k) = 2^k$. We already found that $S(\mathcal{F}, n) = 2n$. Thus, we are looking for the largest $k \le n$ such that $2k = 2^k \to k = 1$. For any $k > 1$, we have $2k < 2^k$, which means that the class $\mathcal{F}$ cannot shatter more than 1 point. Therefore, the VC-dimension of $\mathcal{F}$ is 1, i.e., $V(\mathcal{F}) = 1$.

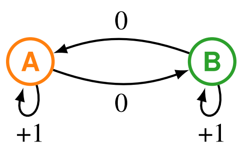
3. (6 pts) Show the following monotonicity property of VC-dimension: if $\mathcal{F}$ and $\mathcal{F}'$ are hypothesis classes with $\mathcal{F} \subset \mathcal{F}'$, then the VC-dimension of $\mathcal{F}'$ is greater than or equal to that of $\mathcal{F}$.
There is the largest integer k such that $\mathcal{F}$ can shatter a set of k points. This means that there exists a set of k points that can be shattered by $\mathcal{F}$, and no set of $k+1$ points can be shattered by $\mathcal{F}$. Similarly, there is the largest integer $m \ge k$ such that $\mathcal{F}'$ can shatter a set of m points. Since $\mathcal{F} \subset \mathcal{F}'$, any labeling of a set of k points that can be achieved by a function in $\mathcal{F}$ can also be achieved by a function in $\mathcal{F}'$.

Therefore, if there exists a set of k points that can be shattered by $\mathcal{F}$, then this same set of k points can also be shattered by $\mathcal{F}'$. This implies that the VC-dimension of $\mathcal{F}'$ is at least k, i.e., $V(\mathcal{F}') = m \ge k$.

Thus, we have shown the monotonicity property of VC-dimension: if $\mathcal{F}$ and $\mathcal{F}'$ are hypothesis classes with $\mathcal{F} \subset \mathcal{F}'$, then the VC-dimension of $\mathcal{F}'$ is greater than or equal to that of $\mathcal{F}$.

4 Q-learning [15 pts]

Consider the following Markov Decision Process. It has two states s. It has two actions a: move and stay. The state transition is deterministic: "move" moves to the other state, while "stay' stays at the current state. The reward r is 0 for move, 1 for stay. There is a discounting factor $\gamma = 0.8$.



The reinforcement learning agent performs Q-learning. Recall the Q table has entries $Q(s,a)$. The Q table is initialized with all zeros. The agent starts in state $s_1 = A$. In any state s_t, the agent chooses the action a_t according to a behavior policy $a_t = \pi_B(s_t)$. Upon experiencing the next state and reward s_{t+1}, r_t the update is:

$$Q(s_t, a_t) \Leftarrow (1-\alpha)Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a'} Q(s_{t+1}, a') \right).$$

Let the step size parameter $\alpha = 0.5$.

1. (5 pts) Run Q-learning for 200 steps with a deterministic greedy behavior policy: at each state s_t use the best action $a_t \in \arg\max_a Q(s_t, a)$ indicated by the current Q table. If there is a tie, prefer move. Show the Q table at the end.
Simulation can be found in chunk comment Q1 of file 'qlearning.ipynb'
As we initialize Q-table as a table full of 0's, the agent will start with the "move" action. In the first step,

the agents start in the state A and choose to move, based on the fact that both move and stay are 0 and they prefer move. As a result, it transitions to B

$$Q(A, move) \Leftarrow (1-\alpha)Q(s_t, a_t) + \alpha\,(r_t + \gamma Q(B, move))$$

$$= Q(A, move) \Leftarrow (1-0.5) \times 0 + 0.5 \times (0 + 0.8 \times 0) = 0$$

Similarly, for the next step, the agent will move to A as both stay and move are 0.

$$Q(B, move) \Leftarrow (1-\alpha)Q(s_t, a_t) + \alpha\,(r_t + \gamma Q(A, move))$$

$$= Q(B, move) \Leftarrow (1-0.5) \times 0 + 0.5 \times (0 + 0.8 \times 0) = 0$$

For the remaining steps, the Q-values for "move" will remain unchanged since the rewards and Q-values used for updates are always 0. The updated Q-table after 200 steps with the deterministic greedy behavior policy will be:

State	Move	Stay
A	0	0
B	0	0

2. (5 pts) Reset and repeat the above, but with an ϵ-greedy behavior policy: at each state s_t, with probability $1-\epsilon$ choose what the current Q table says is the best action: $\arg\max_a Q(s_t, a)$; Break ties arbitrarily. Otherwise, (with probability ϵ) uniformly chooses between move and stay (move or stay both with 1/2 probability). Use $\epsilon = 0.5$.
Simulation can be found in chunk comment Q2 of file 'qlearning.ipynb'
Each step would perform similarly to the following first step: At state s_t, use $1-\epsilon$ to choose action; in this case the probability is $1-\epsilon = 0.5$ to choose between $\arg\max_a Q(s_t, a)$ and uniformly choosing between move and stay. At the start, since the tables are all 0's, the agent will choose a random actions based on ϵ. Suppose the agent choose to stay with the probability of 0.5:

$$Q(A, stay) \Leftarrow (1-\alpha)Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a'} Q(A, a')\right)$$

$$= (1-\alpha)Q(s_t, a_t) + \alpha\,(r_t + \gamma \max(Q(A, move), Q(A, stay))$$

$$= (1-0.5) + 0.5 \times (1 + 0.8 \times \max(0,0)) = 0.5$$

The updated Q-table is:

State	Move	Stay
A	0	0.5
B	0	0

The result of the simulation after 200 steps is:

State	Move	Stay
A	3.99756331	4.99878625
B	3.99863442	4.99746236

3. (5 pts) Without doing simulation, use Bellman equation to derive the true Q table induced by the MDP.
To find the true Q-table, we can use the Bellman equation for Q-values:

$$\mathcal{Q}(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_a Q(s', a')$$

Since the MDP is deterministic, $P(s'|s,a) = 1$ for given action and state. We'll derive the Q values for each state-action pair using the Bellman equation:

$$\mathcal{Q}(A, move) = 0 + \gamma \times \max(\mathcal{Q}(B, move), \mathcal{Q}(B, stay))$$

$$\mathcal{Q}(A, stay) = 1 + \gamma \times \max(\mathcal{Q}(A, stay), \mathcal{Q}(A, move))$$

$$\mathcal{Q}(B, move) = 0 + \gamma \times \max(\mathcal{Q}(A, move), \mathcal{Q}(A, stay))$$

$$\mathcal{Q}(B, stay) = 1 + \gamma \times \max(\mathcal{Q}(B, move), \mathcal{Q}(B, stay))$$

We'll find the optimal policy that maximizes the expected cumulative discount rewards over time. Since staying at each state yields a higher immediate reward than moving, the optimal policy is staying at its stay.

Therefore, $\mathcal{Q}(A, stay)$, $\mathcal{Q}(B, stay)$ are greater than $\mathcal{Q}(A, move)$, $\mathcal{Q}(move)$ respectively. Therefore we have:

$$\mathcal{Q}(A, stay) = 1 + \gamma \times \max(\mathcal{Q}(A, stay), \mathcal{Q}(A, move)) = 1 + \gamma \times \mathcal{Q}(A, stay)$$

$$\mathcal{Q}(B, stay) = 1 + \gamma \times \max(\mathcal{Q}(B, move), \mathcal{Q}(B, stay)) = 1 + \gamma \times \mathcal{Q}(B, stay)$$

Therefore, we have:

$$\mathcal{Q}(A, stay) = \frac{1}{1-\gamma} = 5$$

$$\mathcal{Q}(B, stay) = \frac{1}{1-\gamma} = 5$$

Thus, we can compute $\mathcal{Q}(A, move)$ and $\mathcal{Q}(B, move)$

$$\mathcal{Q}(A, move) = \gamma \times \max(\mathcal{Q}(A, move), \mathcal{Q}(A, stay)) = \gamma \times \mathcal{Q}(A, stay) = 0.8 \times 5 = 0.4$$

$$\mathcal{Q}(A, move) = \gamma \times \max(\mathcal{Q}(B, move), \mathcal{Q}(B, stay)) = \gamma \times \mathcal{Q}(B, stay) = 0.8 \times 5 = 0.4$$

Then the Q table induced by the MDP is:

State	Move	Stay
A	4	5
B	4	5