Instruction

This HW includes both theory and implementation problems. Please note,

- Your code must work with Python 3.7+
- You need to submit a report in PDF including all written deliverable, and all implementation codes so one could regenerate your results.
- For programming part of PCA problem, you can import any module you would like from scikit-learn or other external libraries to minimize the amount of implementation required to get the coding problem done. Submit your code in Problem.py.

PCA

Problem 1. [20 points] As we discussed in the lecture, the PCA algorithm is sensitive to scaling and pre-processing of data. In this problem, we explore few data pre-processing operations on data matrices.

1. Explain what does each of the following data processing operation mean, why it is important, and how you apply to a data matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, n samples each with d features [1] (if it helps, use mathematical equations to show the calculations).
   (a) Centering or mean removal
   (b) Scaling of a feature to a range $[a, b]$
   (c) Standardization
   (d) Normalization (between 0 and 1)
2. Apply the above operations to the following data matrix with 5 samples and two features independently and show the processed data matrix. For scaling pick $a = 0$ and $b = 1$.

$$\boldsymbol{X} = \begin{bmatrix} 1 & 2 \\ -1 & 1 \\ 0 & 1 \\ 2 & 4 \\ 3 & 1 \end{bmatrix}$$

[1] For an implementation of these operation in scikit-learn please see this.

Problem 2. [20 points] In this assignment, you will explore PCA as a technique for discerning whether low-dimensional structure exists in a set of data and for finding good representations of the data in that subspace. To this end, you will do PCA on a Iris dataset which can be loaded in scikit-learn using following commands:

```
from sklearn.datasets import load_iris iris = load_iris()
X = iris.data
y = iris.target
```

1. Carry out a principal component analysis on the entire **raw** dataset (4-dimensional instances) for $k = 1, 2, 3, 4$ components. How much of variance in data can be explained by the first principal component? How does the fraction of variation explained in data vary as k varies?

2. Apply the preprocessing operations from Problem 1 on raw data set and repeat part (1) on processed data. Explain any differences you observe compared to part (1) and justify.

3. Project the raw four dimensional data down to a two dimensional subspace generated by first two top principle components (PCs) from part (2) and produce a scatter plot of the data. Make sure you plot each of the three classes differently (using color or different markers). Can you see the three Iris flower clusters?

4. Either use your k-means++ implementation from previous homework or from scikit-learn to cluster data from part (3) into three clusters. Explain your observations.

Matrix Factorization

Problem 3. [30 points] Recall the following objective for extracting latent features from a partially observed rating matrix via matrix factorization (MF) for making recommendations, discussed in the class:

$$\min_{\boldsymbol{U},\boldsymbol{V}} \left[f(\boldsymbol{U},\boldsymbol{V}) \equiv \sum_{(i,j)\in\Omega} (r_{i,j} - \boldsymbol{u}_i^\top \boldsymbol{v}_j)^2 + \alpha \sum_{i=1}^{n} \|\boldsymbol{u}_i\|_2^2 + \beta \sum_{j=1}^{m} \|\boldsymbol{v}_j\|_2^2 \right] \tag{1}$$

where

- n: number of users
- m: number of items
- $r_{i,j}$: (i,j)-th element in $\boldsymbol{R} \in \mathbb{R}^{n\times m}$ input partially observed rating matrix
- $\Omega \subseteq [n] \times [m]$: index of observed entries in rating matrix, where $[n]$ denotes the sequence of numbers $\{1, 2, \ldots, n\}$.
- k: number of latent features
- $\boldsymbol{U} \in \mathbb{R}^{n\times k}$: the (unknown) matrix of latent feature vectors for n users (the ith row $\boldsymbol{u}_i \in \mathbb{R}^k$ is the latent features for ith user)

- $\boldsymbol{V} \in \mathbb{R}^{m \times k}$: the (unknown) matrix of latent feature vectors for m items (the jth row $\boldsymbol{v}_j \in \mathbb{R}^k$ is the latent features for jth movie)

Please do the followings:

1. In solving Equation 1 with iterative Alternating Minimization algorithm (fixing $\boldsymbol{V}^{(t)}$ and taking gradient step for $\boldsymbol{U}^{(t)}$ and vice versa), discuss what happens if $\boldsymbol{U}^{(0)}$ and $\boldsymbol{V}^{(0)}$ are initialized to zero?

2. Discuss why when there is no regularization in basic MF formulated in Equation 1, i.e., $\alpha = \beta = 0$, each user must have rated at least k movies, and each movie must have been rated by at least k users (Hint: consider the closed form solution for $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$ in notes and argue why these conditions are necessary without regularization).

3. Computing the closed form solution in part (2) could be computational burden for large number of users or movies. A remedy for this would be using iterative optimization algorithms such as Stochastic Gradient Descent (SGD) where you sample movies in updating the latent features for users and sample users in updating the latent features of movies. Derive the updating rules for $\boldsymbol{u}_i^{(t)}$ and $\boldsymbol{v}_j^{(t)}$ at tth iteration of SGD. Show the detailed steps.

Reinforcement Learning

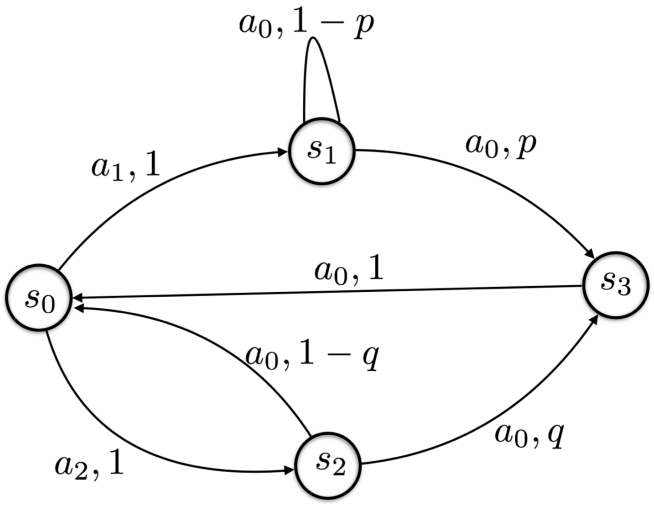

Figure 1: A MDP with states $\mathcal{S} = \{s_0, s_1, s_2, s_3\}$ and actions $\mathcal{A} = \{a_0, a_1, a_2\}$. The reward is 10 for state s_3, 1 for state s_2, and 0 otherwise.

Problem 4. [20 points] Consider the MDP represented by a graph in Figure 1 with discount factor $\gamma \in [0, 1)$. States are represented by circles. The pair on the arrows shows the action to be taken and the transition probability from a state to another state, respectively (please note that the representation of MDP as a graph here is slightly different from Example 3.3 in the book). Each of the parameters p and q are in the interval $[0, 1]$. The reward is 10 for state s_3, 1 for state s_2, and 0 otherwise. Note that this means, from s_0, after taking an action, it will receive 0 reward no matter what the action is. This is similar for other states.

1. List all two deterministic policies (for each state, choose an action with probability 1) for MDP in Figure 1 starting from the initial state s_0 and ending at state s_3. For each policy π compute the value of each state, $v_\pi(s_0)$, $v_\pi(s_1)$, $v_\pi(s_2)$, and $v_\pi(s_3)$ for $\gamma = 0$ and $p = q = \frac{1}{2}$.

2. Show the equation representing the optimal value function for each state, i.e. $v_*(s_0)$, $v_*(s_1)$, $v_*(s_2)$, and $v_*(s_3)$. Please note the value of states will be a function of parameters p, q, and γ.

3. Using $p = q = 0.25$ and $\gamma = 0.9$, compute π_* and v_* for all states of MDP. You can either solve the recursion of part (2) or implement value iteration (if you intend to do the latter, consider the approximation error $\epsilon = 10^{-3}$ as stopping criteria).

Problem 5. [10 points] As discussed in the lecture, a key difference between Sarsa and Q-learning is that Sarsa is on-policy, while Q-learning is off-policy. Now, Suppose action selection is greedy in Sarsa and Q-learning. Is Q-learning then exactly the same algorithm as Sarsa? Will they make exactly the same action selections and weight updates? The pseudocode of both algorithms are shown below.

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure 2: The pseudocode of SARSA and Q-Learning.