Stony Brook University
CSE512 – Machine Learning – Spring 17
Homework 4, Due: April, 20, 2017, 11:59PM
Sudeshna Pal (110938222)

Instructions

- The homework is due on April 20, 2017. Anything that is received after the deadline will not be considered.
- The write-up **must** be prepared in Latex, including the Matlab code and figures in the report, and converted to pdf.
- We can use any Latex class you like, just report question number and your answer.
- If the question requires you to implement a Matlab function, the answer should be your code. Make sure it is sufficiently well documented that the TAs can understand what is happening.
- Each Question, regardless of how many sub-questions contains, is worth 10 points.

MNIST data

The mnist.mat file contains the variables Xtr and ytr, with the vectorization of 60000 grayscale 28x28 images representing handwritten digits and the labels respectively. The variables Xte and yte contains the test data. You can see the original images with, for example,

```
imagesc(reshape(Xtr(:,1),[28,28])')
```

1 Question 1: Varying Learning Rate for SGD

Slighly generalizing the proof seen in class it easy to see that an update of the form $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta g_t$ where g_t is an unbiased estimate of a subgradient in $\partial f(\boldsymbol{w}_t)$, plus the final averaging, guarantees the following convergence rate

$$\mathbb{E}\left[f\left(\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{w}_t\right)\right] - f(\boldsymbol{w}^*) \leq \frac{\|\boldsymbol{w}^*\|^2}{2\eta T} + \frac{\eta\sum_{t=1}^{T}\mathbb{E}[\|g_t\|_2^2]}{2T} \ .$$

(In the above we start with $\boldsymbol{w}_1 = \boldsymbol{0}$.) For Lipschitz functions, the bound suggests a learning rate η of the form $O(\frac{1}{\sqrt{T}})$. However, this requires to know beforehand the number of iterations one want to do.

Here, you have to show that using updates of the form $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t g_t$, where $\eta_t = O(\frac{1}{\sqrt{t}})$, gives convergence as well for Lipschitz functions. Note that you might have to change the averaging procedure to make it work. Also, be careful that here, differently from the proof seen in class, the varying learning rate might prevent to have a telescopic sum.

Assuming f is strongly convex and let g_t be the subgradient at w_t then,

$$\langle w_t - w^*, g_t\rangle \geq f(w_t) - f(w^*) + \frac{\lambda}{2}||w_t - w^*||^2$$

$$Also, since||w_{t+1/2} - w^*|| \geq ||w_{t+1} - w^*||^2$$

$$||w_t - w^*||^2 - ||w_{t+1} - w^*||^2 \geq ||w_t - w^*||^2 - ||w_{t+1/2} - w^*||^2$$

$$= 2\eta_t\langle w_t - w^*, v_t\rangle - \eta^2||v_t||^2$$

$$\sum_{t=1}^{T}(\mathbb{E}[f(w_t)] - f(w^*)) \leq \mathbb{E}\left[\sum_{t=1}^{T}(\frac{||w_t - w^*||^2 - ||w_{t+1} - w^*||^2}{2\eta_t} - \frac{\lambda}{2}||w_t - w^*||^2)\right]$$

For $\lambda = 1$ and η as $\frac{1}{\sqrt{T}}\mathbb{E}\left[f\left(\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{w}_t\right)\right] - f(\boldsymbol{w}^*) \leq \frac{\|\boldsymbol{w}^*\|^2}{2\eta T} + \frac{\eta\sum_{t=1}^{T}\mathbb{E}[\|g_t\|_2^2]}{2T}$

2 SGD for Multiclass Classification with Kernels and Costs

Suppose labeled points $(\boldsymbol{x}, y)$ are drawn from $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is the feature space and $\mathcal{Y}$ your set of labels. Suppose you are also given a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which computes the inner product for a feature map $\phi : \mathcal{X} \to \mathcal{H}$ where $\mathcal{H}$ is a Hilbert space for k. In other words,

$$k(\boldsymbol{x}, \boldsymbol{x}') = \langle\phi(\boldsymbol{x}), \phi(\boldsymbol{x}')\rangle\ .$$

Suppose you have collected a training set of m examples

$$S = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_m, y_m) \in \mathcal{X} \times \mathcal{Y}\}\ .$$

In the following, if an algorithm uses a kernel k, we mean using the algorithm on the transformed dataset

$$S' = \{(\phi(\boldsymbol{x}_1), y_1), (\phi(\boldsymbol{x}_2), y_2), \cdots, (\phi(\boldsymbol{x}_m), y_m) \in \mathcal{H} \times \mathcal{Y}\}\ .$$

Here we will use the polynomial kernel of the form $k(\boldsymbol{x}, \boldsymbol{x}') = (\langle\boldsymbol{x}, \boldsymbol{x}'\rangle)^p$, where p is a parameter.

You have to implement SGD for multiclass classification with kernels, using one different hyperplane for each class. We assume to have k classes, so $\mathcal{Y} = \{1, \cdots, k\}$. Given the matrix W that has the hyperplanes $\boldsymbol{w}_i$ as columns, the multiclass hinge loss is:

$$\ell(W, \phi(\boldsymbol{x}), y) = \max_{y' \in \{1,\cdots,k\}} \Delta(y', y) + \langle\boldsymbol{w}_{y'}, \phi(\boldsymbol{x})\rangle - \langle\boldsymbol{w}_y, \phi(\boldsymbol{x})\rangle \quad (1)$$

where $\Delta(y', y)$ is a matrix of costs for predicting y' instead of the true label y. The pseudocode is in Algorithm 1.

Algorithm 1 Stochatic sub-gradient descent for multiclass with kernels

```
$\boldsymbol{w}_{1,i} = \boldsymbol{0}, i = 1, \cdots, k$
for $t = 1, 2, \cdots, m$ do
    Take $(\phi(\boldsymbol{x}_t), y_t)$ from your training data
    $\eta_t \leftarrow \frac{1}{\sqrt{t}}$
    Calculate the subgradient of the multiclass loss in (1) on $(\phi(\boldsymbol{x}_t), y_t)$ using $\boldsymbol{w}_{t,i}$ for each class and denote
    it by $_{t,i}, i = 1, \cdots, k$
    $\boldsymbol{w}_{t+1,i} = \boldsymbol{w}_{t,i} - \eta_{tt,i},\ \forall i = 1, \cdots, k$
end for
```

2.1 Question 2: Updates

In this question, do not use the kernel yet, just use the notation with ϕ. Assume you want to classifiy handwritten digits. Hence, denotes the classes by the numbers $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

(a) Write the matrix Δ in the case you pay 0 for each correct prediction and 1 for each wrong one.

$$\Delta = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

(b) Write the matrix Δ in the case you pay 0 for each correct prediction, 1 for each wrong prediction between classes whose digits are one number apart one from the other (e.g. you predicted "2" and the correct label is "3"), and pay 2 for all the other cases.

$$\Delta = \begin{bmatrix} 0 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 0 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 2 & 1 & 0 & 1 & 2 & 2 & 2 & 1 & 2 & 2 \\ 2 & 2 & 1 & 0 & 1 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 1 & 0 & 1 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 1 & 0 & 1 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 1 & 0 & 1 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 1 & 0 & 1 & 2 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 0 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 0 \end{bmatrix}$$

(c) Write the subgradient of the multiclass hinge loss in (1) for a generic matrix Δ.

We know $\ell(W, \phi(\boldsymbol{x}), y) = \max_{y' \in \{1,\cdots,k\}} \ \Delta(y', y) + \langle \boldsymbol{w}_{y'}, \phi(\boldsymbol{x})\rangle - \langle \boldsymbol{w}_y, \phi(\boldsymbol{x})\rangle$

Therefore, sub-gradient makes sense when $\ell(W, \phi(\boldsymbol{x}), y)$ is greater than zero.
Let, y_i be the class for which we need to calculate the sub-gradient to update $\boldsymbol{w}_i$.
Let y' be the predicted y from information calculated from previous input samples $\phi(x)$ and y be the correct label for input sample $\phi(x)$.

Sub-gradient of multi-class hinge loss for generic delta is as follow:

$$g_{t,i} = \begin{cases} 0 & \text{y' = y} \\ 0 & y_i \neq \text{y and } y_i \neq y' \\ \phi(x_t) & y_i = y' \text{ and } y' \neq y \\ -\phi(x_t) & y_i = \text{y and } y' \neq y \end{cases} \tag{2}$$

2.2 Question 3: SGD with kernels

In the algorithm, the update depends on $\phi(\boldsymbol{x}_t)$ that we don't know/we don't want to use. Instead, we have to realize that the algorithm can be implemented just using kernels, and never using the function ϕ.

(a) From the algorithm's pseudocode and the subgradient derived in Question 2, show that in each iteration t the hyperplane $\boldsymbol{w}_i$ for each class is expressed as a linear combination of the points $\phi(\boldsymbol{x}_j)$, i.e. $\boldsymbol{w}_{t,i} = \sum_{j=1}^{t-1} \alpha_{j,i}\phi(\boldsymbol{x}_j)$, where $\alpha_{j,i}$ depends on the subgradients.

> Prove, $\boldsymbol{w}_{t,i} = \sum_{j=1}^{t-1} \alpha_{j,i}\phi(\boldsymbol{x}_j)$
>
> From the algorithm $\boldsymbol{w}_{t+1,i} = \boldsymbol{w}_{t,i} - \eta_t g_{t,i}$
>
> We know sub-gradient is defined in terms of $\phi(x)$ only.
>
> Hence, we write, $\boldsymbol{w}_{t+1,i} = \boldsymbol{w}_{t,i} - f(\phi(x_t)),\ \forall i = 1, \cdots, k$ where $f(\phi(x_t)) = \eta\phi(x_t) or - \eta\phi(x_t)$
>
> As the $\boldsymbol{w}_{t+1,i}$ is the linear combination of $\boldsymbol{w}_{t,i}$ and $f(\phi(x_t))$, we can say that, $\boldsymbol{w}_{t+1,i}$ is linear combination of $f(\phi(x_t))$.
>
> So, $\boldsymbol{w}_{t+1,i}$ is linear combination of $\phi(x_t)$ and can be represented as $\boldsymbol{w}_{t,i} = \sum_{j=1}^{t-1} \alpha_{j,i}\phi(\boldsymbol{x}_j)$ where $\alpha_{j,i}$ can be either *zero* or $-\eta$ or η

(b) To calculate the multiclass hinge loss and its subgradient, you need to to be able to calculate $\langle \boldsymbol{w}_{t,i}, \phi(\boldsymbol{x}_t)\rangle$. Show that this is possible without accessing ϕ, just using the kernel function.

> To calculate $\langle \boldsymbol{w}_{t,i}, \phi(\boldsymbol{x}_t)\rangle$ without accessing ϕ, just using the kernel function As we have proved above that $\boldsymbol{w}_{t,i} = \sum_{j=1}^{t-1} \alpha_{j,i}\phi(\boldsymbol{x}_j)$ where α is number of classes x support vectors and each $\alpha_{j,i}$ is learning rate which can be zero, $-\eta$ or η.
>
> To calculate hinge loss, we multiply weights from all previous iterations with the current iteration $\phi(\boldsymbol{x}_t)$. So, $\langle \boldsymbol{w}_{t,i}, \phi(\boldsymbol{x}_t)\rangle$ can be written as $\sum_{j=1}^{t-1} \alpha_{j,i}\phi(\boldsymbol{x}_j)\phi(\boldsymbol{x}_t)$ (Using definition of w).
>
> $\langle \boldsymbol{w}_{t,i}, \phi(\boldsymbol{x}_t)\rangle = \sum_{j=1}^{t-1} \alpha_{j,i} K(\phi(\boldsymbol{x}_j), \phi(\boldsymbol{x}_t))$

2.3 Question 4: Implementation and Use

(a) Implement SGD over the multiclass hinge loss with kernels. Only one pass is used and no regularizer, in order to directly minimize the true error, as seen in class. The suggested learning rate is $\eta = \frac{1}{\sqrt{m}}$, where m is the number of examples you have. Initialize to $\mathbf{0}$ all the hyperplanes. The prototype should be:

```
[alpha, Xsv] = train_mhinge_krnel_sgd(Xtr, ytr, Delta, p)
```

where `Xtr` is the input matrix, `ytr` is the vector of labels, `Delta` is the matrix of the costs in (1), `p` is the parameter p of the polynomial kernel, `alpha` is the matrix of the $\alpha_{i,j}$ found at the end of the SGD procedure, and `Xsv` is the subset of the input matrix of which you performed an update. (Note that on my laptop this takes less than 4 minutes).

```matlab
% Stochastic sub-gradient descent for multiclass with kernels
function [alpha, Xsv] = train_mhinge_krnel_sgd(Xtr, ytr, Delta, p)

    [nFtrs, nSamples] = size(Xtr);
    nClasses = length(unique(ytr));
    ytr = ytr-min(ytr)+1;
    alpha = zeros(nClasses, 1);
    svIndex = [1];

        %for each sample you keep updating weights in Stochstic case
        for i = 2:nSamples
            eta = 1/sqrt(i);
            x_i = Xtr(:, i);
            y_i = ytr(i);

            % compute kernel<x_i, Xtr(suppportIndices)> size(K)=[sv x 1]
            K = polynomialKernel(Xtr(:, svIndex), x_i, 5);

            % size(alpha) = [nclasses x sv]
            % size(y_i_pred) = [nclasses x 1]
            y_i_pred_matrix = repmat(K, nClasses, 1) .* alpha; %----------classes x nSamples
            y_i_pred = sum(y_i_pred_matrix, 2); % ------- classes x 1

            % val = hinge loss and y_cap is pred_y
            [val, y_cap] = max(Delta(:, y_i) + y_i_pred - y_i_pred(y_i));

            % set gt = si(xt, y_cap) - si(xt, yt)
            % wt+1,i = wt,i - eta * gt,i; => alpha = alpha - eta(plus or minus)
            % update only when hinge loss is > 0
            if val > 0
                newSVIndex = find(svIndex==i);
                if sum(newSVIndex)==0 % new support vector 'i' is not in svIndex
                    svIndex = [svIndex i];
                    alpha = horzcat(alpha, zeros(nClasses, 1));
                    alpha(y_cap, end) = alpha(y_cap, end) - eta;
                    alpha(y_i, end) = alpha(y_i, end) + eta;
                else % support vector 'i' is in svIndex
                    %alpha(y_cap, newSVIndex) = alpha(y_cap, newSVIndex) - eta;
                    %alpha(y_i, newSVIndex) = alpha(y_i, newSVIndex) + eta;
                end

                %w(:, y_cap) = w(:, y_cap) - eta * Xtr(i, :)';
                %w(:, y_i) = w(:, y_i) + eta * Xtr(i, :)';
            end
        end

    alpha = alpha;
    Xsv = Xtr(:, svIndex);
end
```

```
% Polynomial Kernel is distance between derived polynomial features of
% x1 and x2
function polyDistance = polynomialKernel(X1, X2, degree)
    %gaussian kernel
    %polyDistance = exp(-0.001.*pdist2(X1',X2','euclidean').^2);
    X2 = repmat(X2, 1, size(X1,2));
    dot_prod = X1 .* X2;
    K = sum(dot_prod, 1);
    polyDistance = K.^5;

end
```

(b) Write a function to test the matrix of hyperplanes generated by your SGD algorithm. The prototype should be

`[ypred] = test_mhinge_kernel_sgd(alpha, Xsv, Xte, p)`

where `alpha` is the matrix of coefficents from the training procedure, `Xsv` is the subset of the input matrix of which you performed an update during the training phase, `Xte` is the matrix of test samples, `p` is the parameter p of the polynomial kernel, and `ypred` is the vector of the predicted classes.

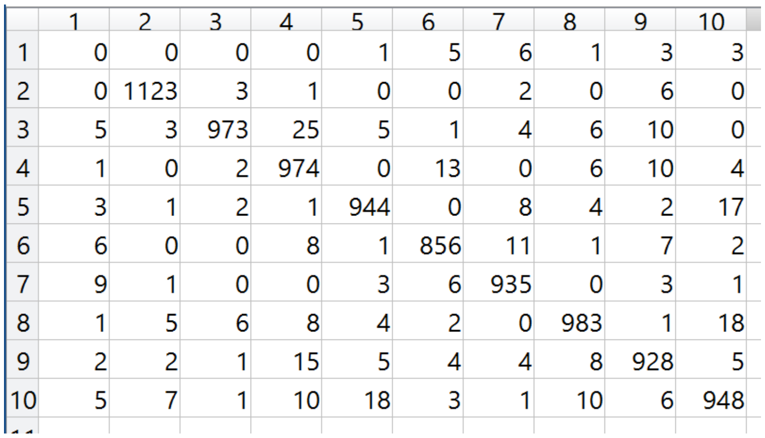
```
function [ypred] = test_mhinge_kernel_sgd(alpha, Xsv, Xte, p)
    [nFtrs, nTestSamples] = size(Xte);
    ypred = zeros(1, nTestSamples);
    nClasses = 10;

    for i=1:nTestSamples
        x_i = Xte(:, i);
        K = polynomialKernel(Xsv, x_i, 5);
        y_i_pred_matrix = repmat(K, nClasses, 1) .* alpha;
        y_i_pred = sum(y_i_pred_matrix, 2);
        [val, y_cap] = max(y_i_pred);
        ypred(i) = y_cap;
    end
end
```

(c) Use the two functions above to train and test on the MNIST dataset, using a polynomial kernel of degree 5. Consider the last solution of the SGD and report the obtained test error (0/1 loss) and the confusion matrices when using Δ as in Question 2.a and with Δ as in Question 2.b

0/1 Loss of 2.a. delta1 = 3.75%
0/1 Loss of 2.b. delta2 = 3.75%

Same confusion matrix for both deltas.

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 1 & 5 & 6 & 1 & 3 & 3 \\
0 & 1123 & 3 & 1 & 0 & 0 & 2 & 0 & 6 & 0 \\
5 & 3 & 973 & 25 & 5 & 1 & 4 & 6 & 10 & 0 \\
1 & 0 & 2 & 974 & 0 & 13 & 0 & 6 & 10 & 4 \\
3 & 1 & 2 & 1 & 944 & 0 & 8 & 4 & 2 & 17 \\
6 & 0 & 0 & 8 & 1 & 856 & 11 & 1 & 7 & 2 \\
9 & 1 & 0 & 0 & 3 & 6 & 935 & 0 & 3 & 1 \\
1 & 5 & 6 & 8 & 4 & 2 & 0 & 983 & 1 & 18 \\
2 & 2 & 1 & 15 & 5 & 4 & 4 & 8 & 928 & 5 \\
5 & 7 & 1 & 10 & 18 & 3 & 1 & 10 & 6 & 948
\end{pmatrix}$$



	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	1	5	6	1	3	3
2	0	1123	3	1	0	0	2	0	6	0
3	5	3	973	25	5	1	4	6	10	0
4	1	0	2	974	0	13	0	6	10	4
5	3	1	2	1	944	0	8	4	2	17
6	6	0	0	8	1	856	11	1	7	2
7	9	1	0	0	3	6	935	0	3	1
8	1	5	6	8	4	2	0	983	1	18
9	2	2	1	15	5	4	4	8	928	5
10	5	7	1	10	18	3	1	10	6	948

3 Question 5: Boosting

We have informally argued that the AdaBoost algorithm uses the weighting mechanism to "force" the weak learner to focus on the problematic examples in the next iteration. In this question we will find some rigorous justification for this argument. Show that the 0/1 error of f_t w.r.t. the distribution D_{t+1} is exactly 1/2. That is, show that for every $t \in [1, T]$

$$\sum_{i=1}^{m} D_{t+1}(\boldsymbol{x}_i, y_i) \cdot \mathbf{1}[y_i \neq f_t(\boldsymbol{x}_i)] = \frac{1}{2} \ .$$

$$D_{t+1}(x,y) \propto D_t(x,y) exp(-\alpha_t \cdot y f_t(x)) \tag{3}$$

$$= \frac{D_t(x,y)}{z} exp(-\alpha_t \cdot y f_t(x)) \tag{4}$$

$$\text{where } \alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t}) \tag{5}$$

We know that

$$y \cdot f_t(x) = \begin{cases} 1, & \text{if correct} \\ -1, & \text{if wrong} \end{cases} \tag{6}$$

Then substituting in the above equation we get,

$$D_{t+1}(x,y) = \begin{cases} \frac{D_t(x,y)}{z}\sqrt{\frac{\epsilon_t}{1-\epsilon_t}}, & \text{if correct} \\ \frac{D_t(x,y)}{z}\sqrt{\frac{1-\epsilon_t}{\epsilon_t}}, & \text{if wrong} \end{cases} \tag{7}$$

Also, all the weights should sum to 1. Thus,

$$\sum_{i=1}^{m} D_{t+1}(x_i, y_i) = 1 \tag{8}$$

$$\sqrt{\frac{\epsilon_t}{1-\epsilon_t}}\frac{1-\epsilon_t}{z} + \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}\frac{\epsilon_t}{z} = 1 \tag{9}$$

$$z = 2\sqrt{\epsilon_t(1-\epsilon_t)} \tag{10}$$

Therefore,

$$D_{t+1}(x,y) = \begin{cases} \frac{D_t(x,y)}{2}\frac{1}{1-\epsilon_t}, & \text{if correct} \\ \\ \frac{D_t(x,y)}{2}\frac{1}{\epsilon_t}, & \text{if wrong} \end{cases} \tag{11}$$

0/1 error of f_t w.r.t. the distribution $D_{t+1} =$

$$\sum_{i=1}^{m} D_{t+1}(x_i,y_i)\cdot \mathbf{1}[y_i \neq f_t(\boldsymbol{x}_i)] = \frac{1}{2\epsilon_t}\cdot \sum_{i=1}^{m} D_t(x_i,y_i)\cdot \mathbf{1}[y_i \neq f_t(\boldsymbol{x}_i)] \tag{12}$$

$$= \frac{1}{2\epsilon_t}\epsilon_t \tag{13}$$

$$= \frac{1}{2} \tag{14}$$

Hence proved.

4 PCA via Successive Deflation

Suppose we have a set of m data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, where each $\boldsymbol{x}_i$ is represented as a d-dimensional column vector.

Let $\boldsymbol{X} = [\boldsymbol{x}_1 \cdots \boldsymbol{x}_m]$ be the $(d \times m)$ matrix where column i is equal to $\boldsymbol{x}_i$. Define $C = \frac{1}{m}\boldsymbol{X}\boldsymbol{X}^\top$ to be the covariance matrix of $\boldsymbol{X}$, where $c_{ij} = \sum_{l=1}^{m} \boldsymbol{x}_{il}\boldsymbol{x}_{jl} = covar(i,j)$.

Next, order the eigenvectors of C by their eigenvalues (largest first), and let $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$ be the first k eigenvectors. These satisfy

$$\boldsymbol{v}_i^\top \boldsymbol{v}_j = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}$$

$\boldsymbol{v}_1$ is the first principal eigenvector of (the eigenvector with the largest eigenvalue), and as such satisfies $\boldsymbol{v}_1 = \lambda_1 \boldsymbol{v}_1$. Now define $\tilde{\boldsymbol{x}}_i$ as the orthogonal projection of $\boldsymbol{x}_i$ onto the space orthogonal to $\boldsymbol{v}_1$:

$$\tilde{\boldsymbol{x}}_i = (-\boldsymbol{v}_1\boldsymbol{v}_1^\top)\boldsymbol{x}_i \ .$$

Finally, define $\tilde{\boldsymbol{X}} = [\tilde{\boldsymbol{x}}_1 \cdots \tilde{\boldsymbol{x}}_n]$ as the **deflated matrix** of rank $d-1$, which is obtained by removing from the d-dimensional data the component that lies in the direction of the first principal eigenvector:

$$\tilde{\boldsymbol{X}} = (-\boldsymbol{v}_1\boldsymbol{v}_1^\top)\boldsymbol{X} \ .$$

4.1 Question 6

(a) Show that the covariance of the deflated matrix,

$$\tilde{C} = \frac{1}{m}\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^\top$$

is given by

$$\tilde{C} = \frac{1}{m}\boldsymbol{X}\boldsymbol{X}^\top - \lambda_1\boldsymbol{v}_1\boldsymbol{v}_1^\top$$

(Hint: Some useful facts: $(I - \boldsymbol{v}_1\boldsymbol{v}_1^\top)$ is symmetric, $\boldsymbol{X}\boldsymbol{X}^\top\boldsymbol{v}_1 = m\lambda_1\boldsymbol{v}_1$, and $\boldsymbol{v}_1^\top\boldsymbol{v}_1 = 1$. Also, for any matrices A and B, $(AB)^\top = B^\top A^\top$.)

SOLUTION

We have:

$\tilde{C} = \frac{1}{m}\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^\top = \frac{1}{m}((I - v_1v_1^T)\boldsymbol{X}((I - v_1v_1^T)\boldsymbol{X})^T)$

(because $(AB)^T = B^TA^T$ and $(I - v_1v_1^T)$ is symmetric)

$$\tilde{C} = \frac{1}{m}(I - v_1v_1^T)XX^T(I - v_1v_1^T)$$
$$\tilde{C} = \frac{1}{m}(XX^T - v_1v_1^TXX^T - XX^Tv_1v_1^T + v_1v_1^TXX^Tv_1v_1^T)$$

We know that $XX^Tv_1 = m\lambda_1v_1 \Rightarrow (XX^Tv_1)^T = (m\lambda_1v_1)^T \Rightarrow v_1^TXX^T = m\lambda_1v_1^T$. So,

$$\tilde{C} = \frac{1}{m}(XX^T - v_1m\lambda_1v_1^T - m\lambda_1v_1v_1^T + v_1m\lambda_1v_1^Tv_1v_1^T)$$

Finally, since $v_1^Tv_1 = 1$

$$\tilde{C} = \frac{1}{m}XX^T - \lambda_1v_1v_1^T$$

Hence proved.

(b) Show that for $j \neq 1$, if $\boldsymbol{v}_j$ is a principal eigenvector of C with corresponding eigenvalue λ_j (that is, $C\boldsymbol{v}_j = \lambda_j\boldsymbol{v}_j$), then $\boldsymbol{v}_j$ is also a principal eigenvector of $\tilde{C}$ with the same eigenvalue λ_j.
SOLUTION

$$\begin{aligned}\tilde{C}v_j &= (\frac{1}{m}XX^T - \lambda_1v_1v_1^T)v_j \\ &= \frac{1}{m}(XX^Tv_j) - \lambda_1v_1v_1^Tv_j \\ &= \lambda_jv_j - \lambda_1v_1v_1^Tv_j \ \texttt{(since } XX^Tv_j = m\lambda_jv_j\texttt{)}\end{aligned}$$

$\Rightarrow \tilde{C}v_j = \lambda_jv_j$ (since $v_1^Tv_j = 0$ for $j \neq 1$)
and for $j = 1$,
$\tilde{C}v_1 = \lambda_1v_1 - \lambda_1v_1v_1^Tv_1 = \lambda_1v_1 - \lambda_1v_1 = 0$

Hence, for $j \neq 1$, v_j is also a principle eigenvector of $\tilde{C}$ with same eigenvalue λ_j. **Also,** v_1 **is an eigenvector of** $\tilde{C}$ **with eigenvalue 0.**

(c) Let $\boldsymbol{u}$ be the first principal eigenvector of $\tilde{C}$. Explain why $\boldsymbol{u} = \boldsymbol{v}_2$. (You may assume $\boldsymbol{u}$ is unit norm.)
SOLUTION

Since $v_1, v_2, \ldots v_k$ are the first k eigenvectors with largest eigenvalues of C, i.e., the principal basis vectors, therefore
$$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_k$$
From part (b), we know that for $\tilde{C}$, the principle eigenvectors v_j have eigenvalues $(0, \lambda_2, \lambda_3, \ldots, \lambda_k)$. Since λ_2 is the largest eigenvalue of $\tilde{C}$, v_2 is the first principle eigenvector of $\tilde{C}$.

Hence $\boldsymbol{u} = \boldsymbol{v}_2$, where $\boldsymbol{u}$ be the first principal eigenvector of $\tilde{C}$.

(d) Suppose we have a simple method f for finding the leading eigenvector and eigenvalue of a positive-definite matrix, denoted by $[\lambda, \boldsymbol{u}] = f(C)$. Write some pseudocode for finding the first k principal basis vectors of $\boldsymbol{X}$ that only uses the special f function and simple vector arithmetic.

(Hint: This should be a simple iterative routine that takes only a few lines to write. The input is C, k, and the function f, the output should be $\boldsymbol{v}_j$ and λ_j for $j \in 1, \cdots, k$)

Pseudocode for finding the first k principal eigenvectors of X:

```
function [lambda_list, v_list] = findEigenVectors(k,C,f):
        lambda_list = []
        v_list = []
        for i in range(k):
                lambda, v = f(C)
                C = C − lambda * v * transpose(v)
                lambda_list.append(lambda)
                v_list.append(v)
        return lambda_list, v_list
```

5 Clustering with k-means

In class we discussed the k-means clustering algorithm. Your programming assignment this time is to implement the k-means algorithm on digit data.

The goal of clustering can be thought of as minimizing the variation within groups. A good clustering model has low sum of squares within each group.

We define the sum of squares in the traditional way. Let $_k$ be the k^{th} cluster and let $_k$ be the mean of the observations in cluster $_k$. Then the within group sum of squares for cluster $_k$ is defined as:

$$SS(k) = \sum_{i \in _k} ||\boldsymbol{x}_i -_k ||_2^2 \; .$$

Please note that the term $||\boldsymbol{x}_i -_k ||_2$ is the euclidean distance between $\boldsymbol{x}_i$ and $_k$. If there are K clusters in total then the "total within group sum of squares" is just the sum of all K of these individual $SS(k)$ terms.

The k-Means algorithm is a heuristic to minimize the above objective function.

5.1 The k-Means algorithm

1. Select k starting centers that are points from your data set.
2. Assign each data point to the cluster associated with the nearest of the k center points.
3. Re-calculate the centers as the mean vector of each cluster from (2).
4. Repeat steps (2) and (3) until convergence or the limit on the number of iterations is met.

Define convergence as no change in label assignment from one step to another **or** you have iterated 100 times (whichever comes first). Please count your iterations as follows: after 100 iterations, you should have assigned the points 100 times.

5.2 Question 7

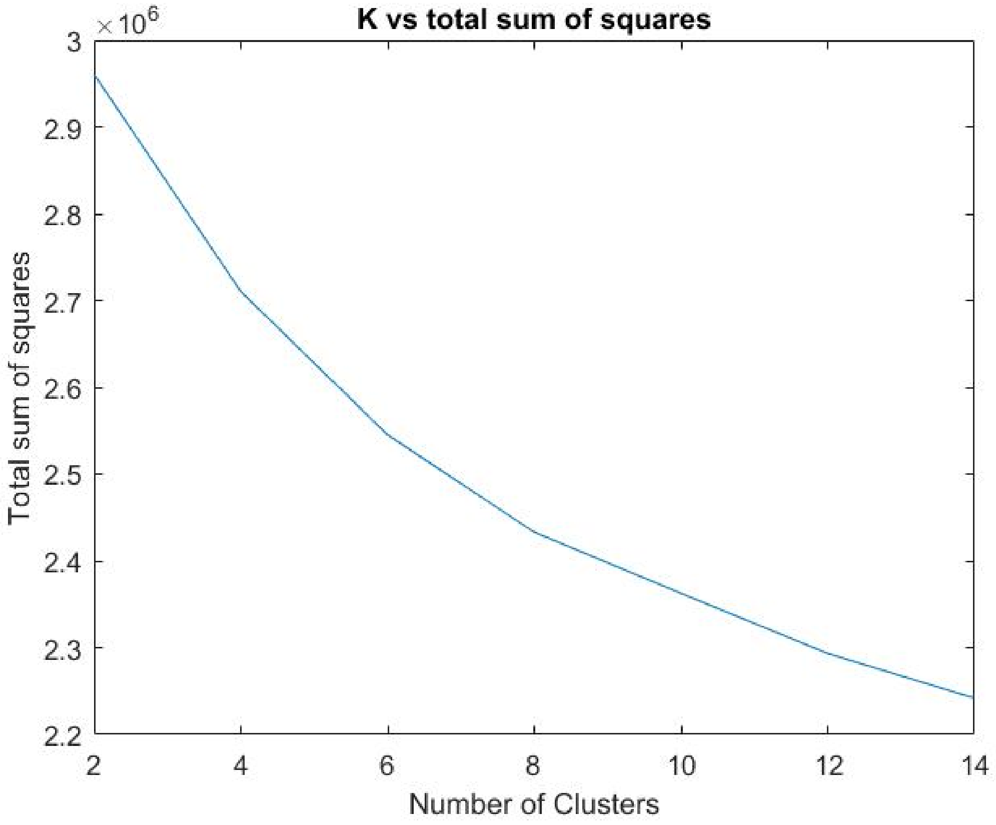
(a) Implement the k-means algorithm as described above. The prototype should be

```
M = kmeans(X, k, T)
```

where `X` is the input matrix and `k` the number of clusters, `T` the maximum number of iterations to use, and `M` is the matrix containing the centers.

```
% Kmeans algorithm.
% X = Xtrain
% k = number of clusters to form
% T = number of iterations
function [M] = kmeans(X, k, T)
    nSamples = size(X,1);
    nFtrs = size(X,2);

    % intialize centroids
    centroids = zeros(k, nFtrs);
    randomCentroids = randperm(size(X,1));
    centroids = X(randomCentroids(1:k), :);

    for i=1:T
        % find closest centroid
        closestCentroidID = zeros(nSamples,1);
        for j=1:nSamples
            kth = 1;
            minDist = sum(X(j,:) - centroids(kth, :)).^2;
            for m = 2:k
                dist = sum(X(j,:) - centroids(m, :)).^2;
                if(dist < minDist)
                    minDist = dist;
                    kth = m;
                end
            end
            closestCentroidID(j) = kth;
        end

        % recompute centroids
        newCentroids = zeros(k, nFtrs);
        for j=1:k
            xj = X(closestCentroidID==j, :);
            count = size(xj, 1);
            newCentroids(j,:) = (1/count) * sum(xj);
        end
        centroids = newCentroids;

    end
    M = centroids;
end
```

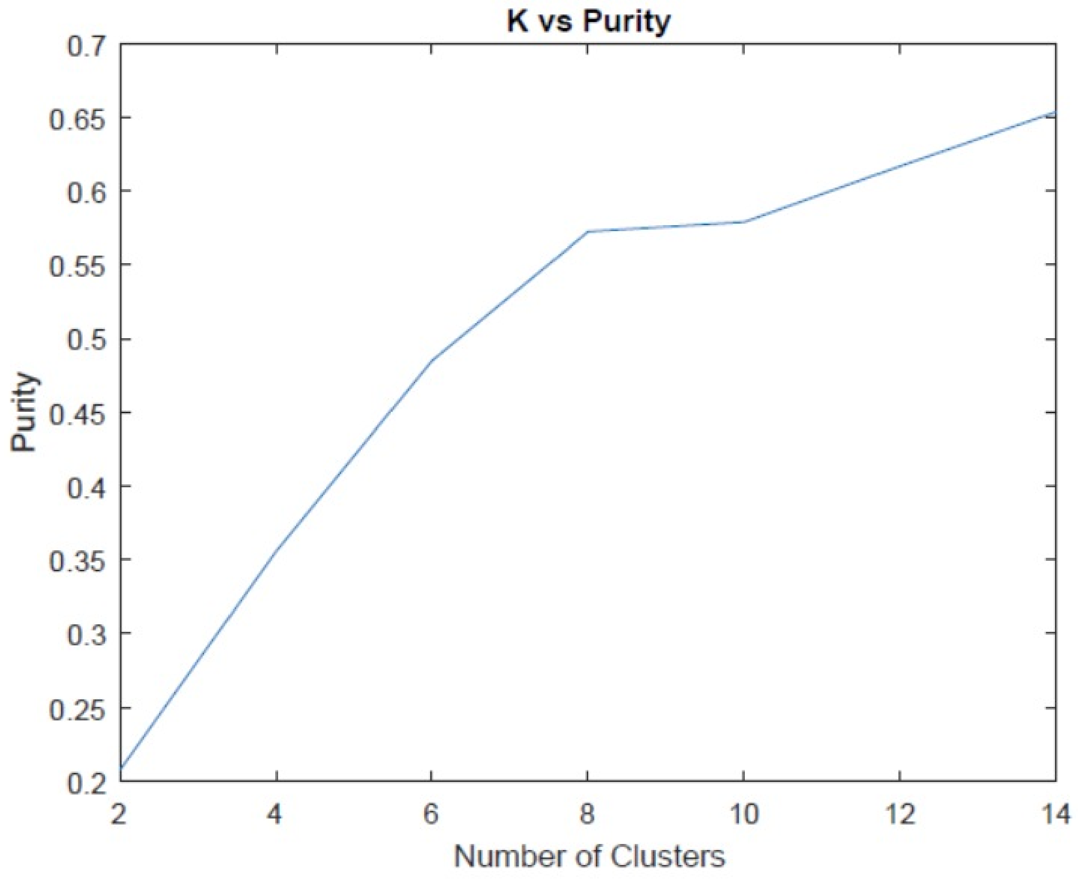
(b) Run the k-means algorithm on the training data of MNIST. A plot of the values of the total within group sum of squares for $k = 2$, $k = 4$, $k = 6$, $k = 8$, $k = 10$, $k = 12$, and $k = 14$. Start your centers with the first k points in the dataset.



(c) Let's try to understand how good is k-means in recovering the clusters corresponding to the classes. We will use the *cluster purity* as a measure of success. It is defined as

$$Purity = \frac{1}{m} \sum_{i=1}^{k} \max_t |\{j \in C_i \text{ and } y_j = t\}| \ .$$

In words, we associate to each cluster the label corresponding to the class with the majority of the points in the cluster. Hence, having associated a label to each cluster, we can count how many points are correctly classified. Note that the purity can only be calculated in retrospect using the labels associated to each sample, that were unknown during the clustering process. Plot how the purity change with $k = 2$, $k = 4$, $k = 6$, $k = 8$, $k = 10$, $k = 12$, and $k = 14$.



For the last two items, you should run k-means algorithm 10 times and average the results. For each question, submit a single plot, which is the average of the runs.

6 Manual calculation of one round of EM for a Mixture of Gaussians

In this question we consider clustering 1D data with a mixture of 2 Gaussians using the EM algorithm. You are given the 1-D data points $x = [1 \;\; 10 \;\; 20]$.

Question 8

M step. Suppose the output of the E step is the following matrix:

$$Q = \begin{pmatrix} 1 & 0 \\ 0.4 & 0.6 \\ 0 & 1 \end{pmatrix}$$

where entry $Q_{i,c}$ is the probability of observation x_i belonging to cluster c. You just have to compute the M step. You may state the equations for maximum likelihood estimates of these quantities without proof; you just have to apply the equations to this data set. You may leave your answer in fractional form.

(a) Write down the likelihood function you are trying to optimize.

$$Q(\theta, \theta^{(t-1)}) = \sum_i \sum_c Q_{ic} log(\pi_c) + \sum_i \sum_c Q_{ic} log(p(x_i|\theta_c)) \tag{15}$$

If the log-liklihod or the parameters converge, we stop else repeat EM.

(b) After performing the M step for the mixing weights π_1, π_2, what are the new values?

For any mixture model mixing weights are given by,

$$\pi_c = \frac{1}{N} \sum_i Q_{ic} = \frac{Q_c}{N} \tag{16}$$

Hence,

$$\pi_1 = \frac{1}{3}(1 + 0.4 + 0) = \frac{1.4}{3} = \frac{7}{15} \tag{17}$$

$$\pi_2 = \frac{1}{3}(0 + 0.6 + 1) = \frac{1.6}{3} = \frac{8}{15} \tag{18}$$

(c) After performing the M step for the means μ_1 and μ_2, what are the new values?

For a Gaussian mixture model, mean is given by,

$$\mu_c = \frac{\sum_i Q_{ic} x_i}{Q_c} \tag{19}$$

Given 1-D data points x= [1 10 20] and probability matrix Q, we get,

$$\mu_1 = \frac{1 * 1 + 0.4 * 10 + 0 * 20}{1 + 0.4 + 0} = \frac{5}{1.4} = \frac{25}{7} \tag{20}$$

$$\mu_2 = \frac{0 * 1 + 0.6 * 10 + 1 * 20}{1 + 0.6 + 0} = \frac{26}{1.6} = \frac{65}{4} \tag{21}$$

(d) After performing the M step for the standard deviations σ_1 and σ_2, what are the new values?

For a Gaussian mixture model, variance is given by,

$$\Sigma_c = \frac{\sum_i (Q_{ic})(x_i - \mu_c)(x_i - \mu_c)^T}{Q_c} \tag{22}$$

and standard deviation σ is given by,

$$\sigma_c = \sqrt{\Sigma_c} \tag{23}$$

Substituting 1D data point x = [1 10 20] and probability matrix Q, we get,

$$\Sigma_1 = \frac{1(1 - \frac{25}{7})^2 + 0.4(10 - \frac{25}{7})^2 + 0(20 - \frac{25}{7})^2}{1 + 0.4 + 0} = 16.53 \Rightarrow \sigma_1 = 4.065 \tag{24}$$

$$\Sigma_2 = \frac{0(1 - \frac{65}{4})^2 + 0.6(10 - \frac{65}{4})^2 + 1(20 - \frac{65}{4})^2}{0 + 0.6 + 1} = 23.4375 \Rightarrow \sigma_2 = 4.84 \tag{25}$$

E step. Now suppose the output of the M step is the answer to the previous section. You will compute the subsequent E step.

(e) Write down the formula for the probability of observation x_i belonging to cluster c.

The probability of observation x_i belonging to cluster c:

$$Q_{ic} = \frac{\pi_c p(x_i|\theta_c^{(t-1)})}{\sum_{c'} \pi_{c'} p(x_i|\theta_{c'}^{(t-1)})} \tag{26}$$

where,

$$p(x_i|\theta_c^{(t-1)}) = \frac{1}{\sigma_c\sqrt{2\pi}} \exp -\frac{1}{2}\left(\frac{x_i - \mu_c}{\sigma_c}\right)^2 \tag{27}$$

(f) After performing the E step, what is the new value of Q?

Using the two equations above and plugging in the values from **M-step**, we get:

$$p(x_1|\theta_1) = \frac{1}{4.065\sqrt{2\pi}} \exp -\frac{1}{2}\left(\frac{1 - \frac{25}{7}}{4.065}\right)^2 = 0.0803$$
$$p(x_1|\theta_2) = \frac{1}{4.84\sqrt{2\pi}} \exp -\frac{1}{2}(\frac{1 - \frac{65}{4}}{4.84})^2 = 0.000575$$
$$p(x_2|\theta_1) = \frac{1}{4.065\sqrt{2\pi}} \exp -\frac{1}{2}(\frac{10 - \frac{25}{7}}{4.065})^2 = 0.028$$
$$p(x_2|\theta_2) = \frac{1}{4.84\sqrt{2\pi}} \exp -\frac{1}{2}(\frac{10 - \frac{65}{4}}{4.84})^2 = 0.0358$$
$$p(x_3|\theta_1) = \frac{1}{4.065\sqrt{2\pi}} \exp -\frac{1}{2}(\frac{20 - \frac{25}{7}}{4.065})^2 = 0.0000278$$
$$p(x_3|\theta_2) = \frac{1}{4.84\sqrt{2\pi}} \exp -\frac{1}{2}(\frac{20 - \frac{65}{4}}{4.84})^2 = 0.061$$

Hence,

$$Q_{11} = \frac{\frac{7}{15}0.0803}{\frac{7}{15}0.0803 + \frac{8}{15}0.000575} = 0.992$$

$$Q_{12} = 1 - Q_{11} = 1 - 0.992 = 0.008$$

$$Q_{21} = \frac{\frac{7}{15}0.028}{\frac{7}{15}0.028 + \frac{8}{15}0.0358} = 0.406$$

$$Q_{22} = 1 - Q_{21} = 1 - 0.406 = 0.594$$

$$Q_{31} = \frac{\frac{7}{15}0.0000278}{\frac{7}{15}0.0000278 + \frac{8}{15}0.061} = 0.00039$$

$$Q_{32} = 1 - Q_{31} = 1 - 0.00039 = 0.99961$$

$$Q_{new} = \begin{pmatrix} 0.992 & 0.008 \\ 0.406 & 0.594 \\ 0.00039 & 0.99961 \end{pmatrix}$$