Homework 0
CS534 Machine Learning, Spring 2019

This timed homework assignment tests prerequisite knowledge in linear algebra, statistics, and programming. Please observe the time limits in each problem (these limits include time to lookup facts you may have forgotten). Show your work, attach any code in subsequent pages, and include a filled and signed copy of the honor pledge.

Problem 1 (linear algebra - 2 hours)

1.a. Given the pairs of vectors $v_1 = [\frac{\sqrt{3}}{2}, \frac{1}{2}, 0]^T, v_2 = [0, \frac{1}{2}, \frac{\sqrt{3}}{2}]^T$ and $\hat{v}_1 = [\frac{1}{\sqrt{2}}, 0, \frac{-1}{\sqrt{2}}]^T, \hat{v}_2 = [\frac{-1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0]^T$, use software to find the matrix $A \in \mathbb{R}^{3x3}$ such that

$$[\hat{v}_1, \hat{v}_2] = A[v_1, v_2],\ [v_1, v_2] = A^{-1}[\hat{v}_1, \hat{v}_2]. \tag{1}$$

This is a three-dimensional problem but you only have correspondences between two pairs of vectors. One approach is to complement these vector pairs using the cross-products $v_1 \times v_2$, $\hat{v}_1 \times \hat{v}_2$ to create full-rank matrices

$$\hat{V} = [\hat{v}_1, \hat{v}_2, \hat{v}_1 \times \hat{v}_2] = A[v_1, v_2, v_1 \times v_2] = A * V, \tag{2}$$

then use the inverse $A = \hat{V} * V^{-1}$. You can also use Gram-Schmidt orthogonalization to generate the full-rank matrices.

My solution using the cross-product approach is approximately

$$A = \begin{bmatrix} 1.0747 & -0.4472 & -0.5583 \\ 0.0949 & -0.1644 & 0.9114 \\ -0.3950 & -0.7301 & 0.4215 \end{bmatrix} \tag{3}$$

which yields residuals $< 1e-15$.

1.b. Show that the inverse of a nonsingular skew-symmetric matrix $A^T = -A$ is also skew-symmetric.

A is non-singular $\implies$ A^{-1} exists.

$(A^{-1})^T = (A^T)^{-1} = (-A)^{-1} = -(A^{-1})$

q.e.d.

1.c. Show that the eigenvalues of a real symmetric matrix $A = A^T$ are always real (use the conjugate transpose).

Let $Ax = \lambda x$ (eigenvalue definition)

$x^H Ax = \lambda x^H x$ (where H is the conjugate-transpose)

$\lambda = \frac{x^H Ax}{x^H x}$ (eigenvalue for A)

$\lambda^H = \frac{(Ax)^H (x^H)^H}{(x)^H (x^H)^H}$

$\lambda^H = \frac{x^H A^H x}{x^H x}$ (eigenvalue for A^H)

since A is real, $A^H = A^T = A$ and $\lambda = \frac{x^H Ax}{x^H x} = \frac{x^H A^H x}{x^H x} = \lambda^H$

q.e.d.

Problem 2 (programming)

Gradient-based optimization is a fundamental technique in machine learning. This technique seeks to minimize a cost f that is a function of some parameters β by iteratively updating the parameters using information from the gradient of the cost.

Given Himmelblau's function

$$f(\beta_1, \beta_2) = (\beta_1^2 + \beta_2^2 - 11)^2 + (\beta_1 + \beta_2^2 - 7)^2 \tag{4}$$

2.a. Identify the *critical points* $\beta^* = [\beta_1^*, \beta_2^*]^T$ of f using a software solver. Classify each critical point as a local minimum, maximum, or saddle point by analyzing the Hessian matrix of second partial derivatives H at β^*

$$H\, f = \left[\begin{array}{cc} \frac{\partial^2 f}{\partial \beta_1^2} & \frac{\partial^2 f}{\partial \beta_1 \partial \beta_2} \\ \frac{\partial^2 f}{\partial \beta_2 \partial \beta_1} & \frac{\partial^2 f}{\partial \beta_2^2} \end{array} \right] \Bigg|_{\beta=\beta^*} \tag{5}$$

Generate a contour plot of the cost f in the range $\hat{\beta}_1, \hat{\beta}_2 \in [-5, 5]$. Plot the critical points and indicate their classifications.

I used Matlab's solver to identify the approximate critical points and evaluated the eigenvalues of the Hessian at these locations:

minima $\{(-3.8175, -3.4404), (3.5993, -2.0900), (2.9259, 2.2612), (-2.8088, 3.2836)\}$

maxima $\{(-0.2876, -0.7565)\}$

saddle $\{(-0.0724, 2.2693), (-3.0717, -0.07403), (3.3874, 0.0577), (0.1454, 3.0277)\}$.

See plot next page.

2.b. Implement gradient descent to find solutions minimizing f using the updating function

$$\hat{\beta}^{i+1} = \hat{\beta}^i - \gamma \nabla f(\hat{\beta}^i) \tag{6}$$

where $\hat{\beta}^i = [\hat{\beta}_1^i, \hat{\beta}_2^i]^T$ is the solution at iteration i, and the *learning rate* $\gamma = 1e{-}3$. Plot the trajectory of your solutions $\hat{\beta}^0, \hat{\beta}^1, \ldots \hat{\beta}^N$ where $N = 10^5$ iterations on the plot from **2.a.**. Illustrate trajectories for ten random initializations of $\hat{\beta}^0 \in [-5, 5]$.

See plot next page.

2.c. Implement Newton's method to find solutions minimizing f using the updating function

$$\hat{\beta}^{i+1} = \hat{\beta}^i - \gamma[H\ f(\hat{\beta}^i)]^{-1}\nabla f(\hat{\beta}^i), \tag{7}$$

where H is the Hessian. Use *learning rate* $\gamma = 1e - 2$. Illustrate trajectories for ten random initializations as in as in **2.b**.

See plot next page.

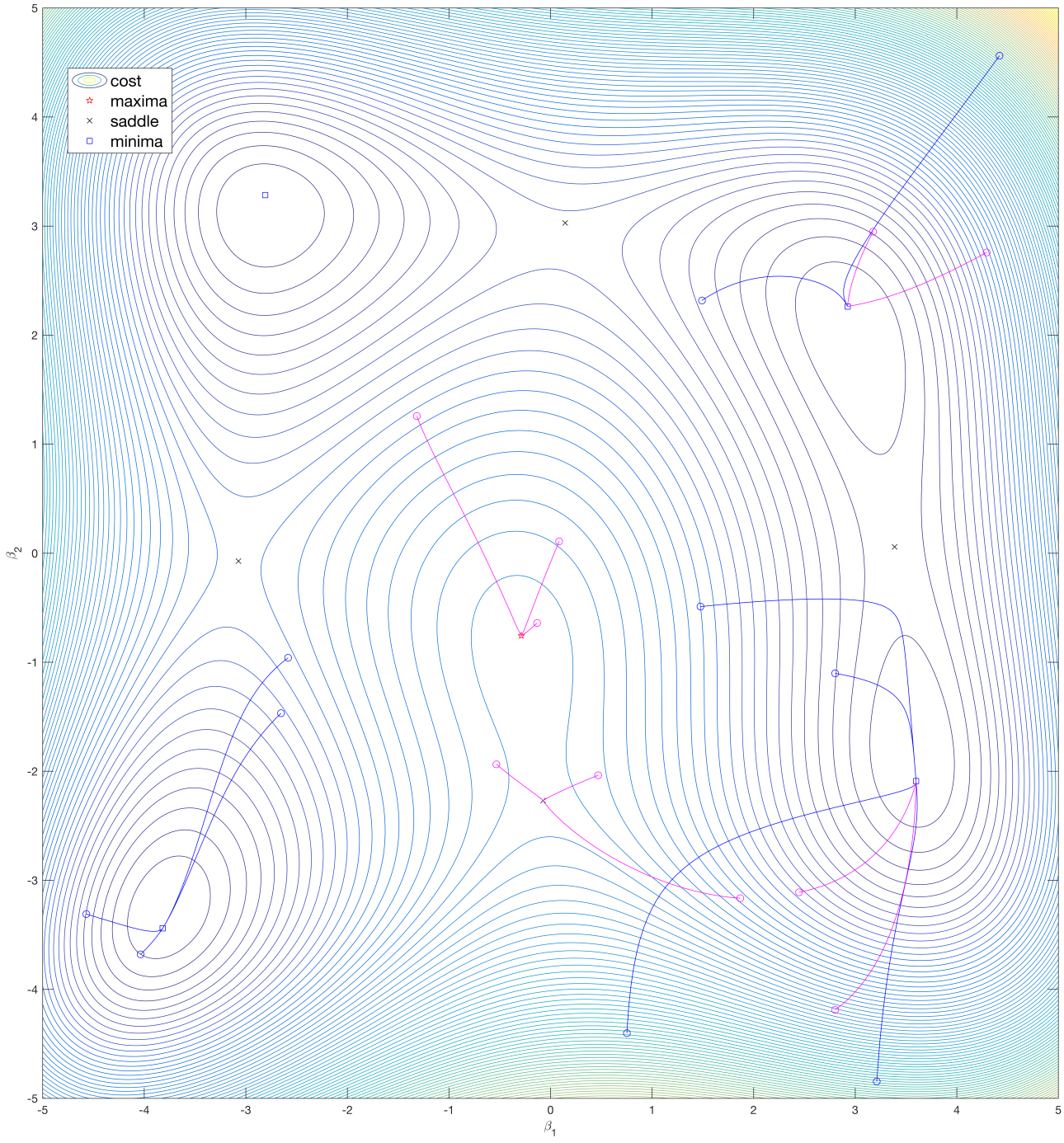

Figure 1: Solutions to problem 2. Gradient trajectories shown in blue, Newton trajectories shown in magenta.

Problem 3 (statistics & probability - 1 hour)

A discrete-valued random variable is defined by its probability mass function $P = \{p_1, p_2, \ldots p_n\}$, where $0 \leq p_i \leq 1$ is the probability of outcome X_i.

One interpretation of the *Shannon entropy* of a random variable $\mathbf{X}$ is the degree of uncertainty in the outcome of $\mathbf{X}$. For discrete-valued random variables the Shannon entropy is calculated as

$$S(\mathbf{X}) = -\sum_i p_i \, log(p_i). \tag{8}$$

3.a. Suppose there is a game with n possible outcomes and probability mass function $p = \{p_1, p_2, \ldots p_n\}$. What P maximizes the entropy of the game? Explain your answer in plain English.

The game with the most uncertain outcome is where all players are equally likely to win, $p\{p_1 = \frac{1}{n}, p_2 = \frac{1}{n}, \ldots, p_n = \frac{1}{n}\}$ (you would not want to bet on this game!)

3.b. Show that this choice of p maximizes S (Note that for a concave function f, $\sum_i a_i f(x_i) \leq f(\sum_i a_i x_i)$ for $a_i \geq 0$, $\sum_i a_i = 1$).

Note first that $-p * log(p)$ is a concave function. With $\sum_i p_i = 1$, we can use the relation provided above to show the entropy upper bound

$-\sum_i p_i log(p_i) = \sum_i p_i log(\frac{1}{p_i}) \leq log(\sum_i p_i \frac{1}{p_i}) = log(n)$

This bound is achieved for $p_i = \frac{1}{n}$

$$\begin{aligned} -\sum_i \frac{1}{n} log(\frac{1}{n}) &= -\sum_i \frac{1}{n}(log(1) - log(n)) \\ &= \sum_i \frac{1}{n}(log(n) - 0) \\ &= log(n) \end{aligned}$$

q.e.d.