CS 229, Fall 2024
Problem Set #1

YOUR NAME HERE (`YOUR SUNET HERE`)

Due Wednesday, October 9 at 11:59 pm on Gradescope.

Notes: (1) These questions require thought, but do not require long answers. Please be as concise as possible.

(2) If you have a question about this homework, we encourage you to post your question on our Ed forum, at `https://edstem.org/us/courses/67631/discussion/`.

(3) If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on the course website before starting work.

(4) For the coding problems, you may not use any libraries except those defined in the provided `environment.yml` file. In particular, ML-specific libraries such as scikit-learn are not permitted.

(5) The due date is Wednesday, October 9 at 11:59 pm. If you submit after Wednesday, October 9 at 11:59 pm, you will begin consuming your late days. The late day policy can be found in the course website: Course Logistics and FAQ.

All students must submit an electronic PDF version of the written question including plots generated from the codes. We highly recommend typesetting your solutions via LaTeX. All students must also submit a zip file of their source code to Gradescope, which should be created using the `make_zip.py` script. You should make sure to (1) restrict yourself to only using libraries included in the `environment.yml` file, and (2) make sure your code runs without errors. Your submission may be evaluated by the auto-grader using a private test set, or used for verifying the outputs reported in the writeup. Please make sure that your PDF file and zip file are submitted to the corresponding Gradescope assignments respectively. We reserve the right to not give any points to the written solutions if the associated code is not submitted.

Honor code: We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down the solution independently, and without referring to written notes from the joint session. Each student must understand the solution well enough in order to reconstruct it by him/herself. It is an honor code violation to copy, refer to, or look at written or code solutions from a previous year, including but not limited to: official solutions from a previous year, solutions posted online, and solutions you or someone else may have written up in a previous year. Furthermore, it is an honor code violation to post your assignment solutions online, such as on a public git repo. We run plagiarism-detection software on your code against past solutions as well as student submissions from previous years. Please take the time to familiarize yourself with the Stanford Honor Code[1] and the Stanford Honor Code[2] as it pertains to CS courses.

[1]https://communitystandards.stanford.edu/policies-and-guidance/honor-code
[2]https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1164/handouts/honor-code.pdf

Regarding Notation: The notation used in this problem set matches the notation used in the lecture notes. Some notable differences from the lecture notation:

- The superscript "(i)" represents an index into the training set – for example, $x^{(i)}$ is the i-th feature vector and $y^{(i)}$ is the i-th output variable. In lecture notation, these would instead be expressed as x_i and y_i.
- The subscript j represents an index in a vector – in particular, $x_j^{(i)}$ represents the j-th feature in the feature vector $x^{(i)}$. In lecture notation, $x_j^{(i)}$ would be $h_j(x_i)$ or $x_i[j]$.
- The vector that contains the weights parameterizing a linear regression is expressed by the variable θ, whereas lectures use the variable $\mathbf{w}$. As such, $\theta_0 = w_0$, $\theta_1 = w_1$, and so on.
- The hypothesis function is expressed as $h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_d x_d = \theta^\top x$. In lecture notation, this would instead be $f_{\mathbf{w}}(x) = w_0 h_0(x) + w_1 h_1(x) + \cdots + w_d h_d(x) = \mathbf{w}^\top h(x)$.
  - $x_0 = 1$ in the problem set notation, just as $h_0(x) = 1$ in the lecture notation.

An overview of this notation is also given at the beginning of the lecture notes (pages 6-7).

1. **[30 points] Convergence and Learning Rate for Gradient Descent**

When running an optimization algorithm, finding a good learning rate may require some tuning. If the learning rate is too high, the gradient descent (GD) algorithm might not converge. If the learning rate is too low, GD may take too long to converge. In this question, we will investigate some theoretical guarantees for the convergence of GD on convex objective functions, and their implications on choosing the learning rate.

Recap and notation. Suppose we have a convex objective function $J(\theta)$. At each iteration, GD updates the iterate $\theta^{[t]}$ as follows:

$$\theta^{[t]} = \theta^{[t-1]} - \alpha \nabla J(\theta^{[t-1]})$$

where α is the learning rate and $\nabla J(\theta^{[t-1]})$ is the gradient of $J(\theta)$ evaluated at $\theta^{[t-1]}$.

(a) **[8 points] Quadratic Objective, Scalar Variable**

In this part, consider the simple objective function with one-dimensional variable θ:

$$J(\theta) = \frac{1}{2}\beta\theta^2$$

where $\theta \in \mathbb{R}$ is our parameter and $\beta > 0$ is a positive, constant scalar. We initialize GD at some initialization $\theta^{[0]} \neq 0$ and run GD with learning rate α.

i. **Derive** the range of α such that the iterate of GD converges in the sense that there exists a scalar $\theta^\dagger$ such that $\lim_{t\to\infty} |\theta^{[t]} - \theta^\dagger| = 0$. Provide the value of $\theta^\dagger$ when GD converges and show that it's equal to the global minimum $\theta^* = \arg\min_\theta J(\theta)$. The range of α can potentially depend on β.

ii. Given a desired accuracy $\epsilon > 0$, for all the α in the range that you derived, **derive** the minimum number of iterations T required to reach a point $\theta^{[T]}$ such that $|\theta^{[T]} - \theta^*| \leq \epsilon$. T can potentially depend on ϵ, $\theta^{[0]}$, α, and β. Investigate the behavior of T and whether T is increasing or decreasing for different values of α. Briefly discuss why choosing an inappropriate α can cause the algorithm to take longer to converge.

Hint: Express $\theta^{[t]}$ in terms of $\theta^{[0]}$, α, t, and β.

Hint: $\lim_{t\to\infty} c^t = 0, \quad \forall c \in \mathbb{R}\ s.t.\ |c| < 1$

Answer: For part (i), using hint 1, derive the following expression

$$\begin{aligned}
\nabla J(\theta^{[t-1]}) &= \frac{1}{2} \cdot 2 \cdot \beta\theta^{[t-1]} \\
&= \theta^{[t-1]}\beta \\
\theta^{[t]} &= \theta^{[t-1]} - \alpha\theta^{[t-1]}\beta \\
&= \theta^{[t-1]}(1-\alpha\beta) \\
&= \theta^{(t-2)}(1-\alpha\beta)^2 \\
&= \cdots \\
&= \theta^{[0]}(1-\alpha\beta)^t
\end{aligned}$$

We know that the optimal value θ^* is 0 (you can either get this by looking at the graph or by setting the derivative to 0 and solving for θ). Thus, in order for the algorithm to converge, we need $\lim_{t\to\infty} \theta^{[t]} = \theta^* = 0$. Using the second hint, we need the condition $|1-\alpha\beta| < 1$.

i. $1 - \alpha\beta < 1$ leads to $\alpha > 0$

ii. $\alpha\beta - 1 < 1$ leads to $\alpha < \frac{2}{\beta}$ (Note dividing by β does not change the sign since $\beta > 0$)

So the answer is $0 < \alpha < \frac{2}{\beta}$.

For part (ii), we first observe that when $\alpha = \frac{1}{\beta}$, we only need 1 iteration to achieve the optimal value $\theta^* = 0$. When $\alpha = \frac{2}{\beta}$, the sign of θ simply changes with each iteration and the algorithm does not converge. For all other $\alpha \neq \frac{1}{\beta}$ within our bound $0 < \alpha < \frac{2}{\beta}$, using the derivation from part (i), we can see that

$$\begin{aligned}
|\theta^{[T]} - \theta^*| &\leq \epsilon \\
|\theta^{[0]}(1 - \alpha\beta)^T| &\leq \epsilon \\
|\theta^{[0]}||(1 - \alpha\beta)^T| &\leq \epsilon \\
|1 - \alpha\beta|^T &\leq \frac{\epsilon}{|\theta^{[0]}|} \\
T \ln|1 - \alpha\beta| &\leq \ln\epsilon - \ln|\theta^{[0]}| \\
T &\geq \frac{\ln\epsilon - \ln|\theta^{[0]}|}{\ln|1 - \alpha\beta|}
\end{aligned}$$

When $0 < \alpha < \frac{1}{\beta}$, as α increases, the RHS decreases, meaning the minimum number of iterations decreases (the algorithm converges faster).

When $\frac{1}{\beta} < \alpha < \frac{2}{\beta}$, as α increases, the RHS increases, meaning the minimum number of iterations increases (the algorithm converges slower).

(b) **[4 points] Quadratic Objective, d-dimensional Variable**

In this part of the question, consider the objective function with d-dimensional variable θ:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{d} \beta_i \theta_i^2$$

where θ_i's are our parameters and $\beta_i \in \mathbb{R}\ s.t.\ \beta_i > 0$ are positive, constant scalars. Assume that we start from some $\theta^{[0]}$ where $\theta_i^{[0]} \neq 0$ for all i and run GD with learning rate α. Derive the range of learning rate α such that GD converges in the sense that there exists a vector $\theta^\dagger \in \mathbb{R}^d$ such that $\lim_{t\to\infty} \|\theta^{[t]} - \theta^\dagger\|_2 = 0$. Provide the value of $\theta^\dagger$ when GD converges. The range of α can potentially depend on β_i where $i \in \{1, \ldots, d\}$.

Answer: Similar to part (a), we quickly derive the update formula for each $\theta_i^{[t]}$ as follows

$$\begin{aligned}
\theta_i^{[t]} &= \theta_i^{[t-1]} - \alpha\theta_i^{[t-1]}\beta_i \\
&= \theta_i^{[t-1]}(1 - \alpha\beta_i) \\
&= \theta_i^{[0]}(1 - \alpha\beta_i)^t
\end{aligned}$$

It is also obvious that the optimal value $\theta_i^* = 0 \quad \forall i \in (1, d)$. So in order for $\theta_i^{[t]}$ to converge, $|1 - \alpha\beta_i| < 1\ \forall i \in (1, d)$.

i. $1 - \alpha\beta_i < 1$ leads to $\alpha > 0$

ii. $\alpha\beta_i - 1 < 1$ leads to $\alpha < \frac{2}{\beta_{\max}}$ where $\beta_{\max} = \max_i \beta_i$

So the solution is $0 < \alpha < \frac{2}{\beta_{\max}}$

(c) [**4 points**] **Coding Question: Quadratic Multivariate Objective**

Let the objective function be

$$J(\theta) = \theta^\top A\theta$$

where $A \in \mathbb{R}^{2\times 2}$ is a 2×2 real, positive semi-definite matrix. Do the following exercises:

i. Implement the `update_theta` and `gradient_descend` function in `src/gd_convergence/experiment.py`. You can stop the GD algorithm when either of the following condition is satisfied:

   A. $|J(\theta^{[t]}) - J(\theta^{[t-1]})| < \epsilon$ where $\epsilon = 10^{-50}$ is given as the function parameter. This is when we assume the algorithm converged.

   B. $J(\theta^{[t]}) > 10^{20}$. This is to prevent an infinite loop when the algorithm does not converge.

   To test your implementation, run `python src/gd_convergence/experiment.py`, which checks that your θ (approximately) converges to the optimal value.

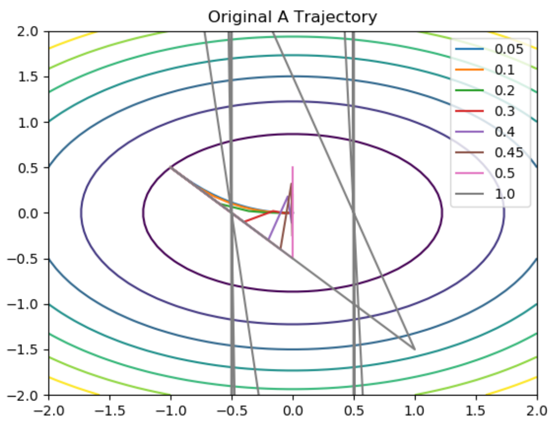
   Note that we have provided you a matrix $A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ and $\theta^{[0]} = [-1, 0.5]$ at the beginning of the file for you to experiment with. Therefore, the objective function is a special case of the objective function in part (b) with dimension $d = 2$. Check if your theoretical derivation of the feasible range of the learning rate indeed matches empirical observations.

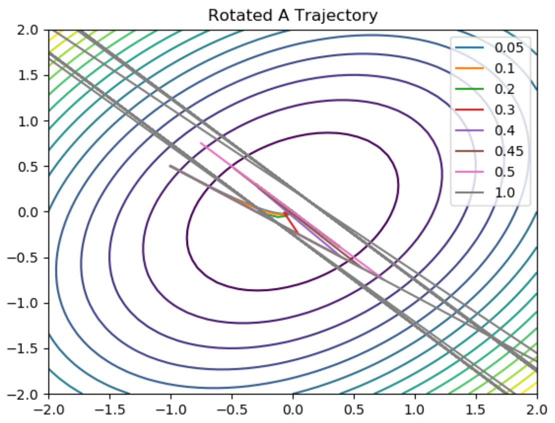
ii. Now, suppose we rotate the matrix A, what do we observe? Plot the trajectories of the GD algorithm using the following learning rates: 0.05, 0.1, 0.2, 0.3, 0.4, 0.45, 0.5, and 1. We have provided the rotation and plotting function for you. You need to simply run `python src/gd_convergence/plotting.py lr1 lr2 lr3 ...` with `lr*` replaced with desired learning rates. Include the output file `trajectories.png` and `trajectories_rotated.png`, as well as a brief discussion on the printed output of `plotting.py` in the write-up.

   Remark: If you find that a learning rate of 0.5 is resulting in non-terminating behavior, this may be due to numerical instability. In this case, feel free to use a learning rate of 0.55 in place of 0.5.

Note that setting the learning rate too high will cause the objective function to not converge. The convergence properties of these learning rates are also rotational invariant. We will show this formally in the next part of the problem.

Answer:





Printed output:

```
[diagonal A, learning rate 0.05] GD took 541 iterations and converged
[diagonal A, learning rate 0.1] GD took 258 iterations and converged
[diagonal A, learning rate 0.2] GD took 115 iterations and converged
[diagonal A, learning rate 0.3] GD took 65 iterations and converged
[diagonal A, learning rate 0.4] GD took 114 iterations and converged
[diagonal A, learning rate 0.45] GD took 257 iterations and converged
[diagonal A, learning rate 0.5] GD took 3 iterations and did not converge
[diagonal A, learning rate 1.0] GD took 23 iterations and did not converge
[rotated A, learning rate 0.05] GD took 531 iterations and converged
[rotated A, learning rate 0.1] GD took 254 iterations and converged
[rotated A, learning rate 0.2] GD took 113 iterations and converged
[rotated A, learning rate 0.3] GD took 64 iterations and converged
[rotated A, learning rate 0.4] GD took 116 iterations and converged
[rotated A, learning rate 0.45] GD took 260 iterations and converged
[rotated A, learning rate 0.5] GD took 3 iterations and did not converge
[rotated A, learning rate 1.0] GD took 22 iterations and did not converge
```

If the learning rate is set too high, GD does not converge. Within the range of learning rates such that GD converges, the convergence speed (i.e. number of iterations to converge) of GD varies, and we observe that increasing the learning rate first causes the number of iterations to converge to decrease but then causes the number of iterations to converge to increase. Note that it is not necessarily true that larger learning rates will make the algorithm converge faster (as shown in problem (a)).

For a specific choice of learning rate, the convergence and the convergence speed of GD are similar for the original A and the rotated A.

(d) **[12 points] Convergence of Gradient Descent for a General Convex Objective**

Now let's consider any convex, twice continuously differentiable objective function $J(\theta)$ that is bounded from below. Suppose that the largest eigenvalue of the Hessian matrix $H = \nabla^2 J(\theta)$ is less than or equal to $\beta_{\max}$ for all points θ.

Show that when running gradient descent, choosing a step size $0 < \alpha < \frac{1}{\beta_{\max}}$ guarantees that $J(\theta^{[t]})$ will converge to some finite value as t approaches infinity.[3] Specifically, you should derive an inequality for $J(\theta^{[t]})$ in terms of $J(\theta^{[t-1]})$, $\nabla J(\theta^{[t-1]})$, α, and $\beta_{\max}$, and show that the objective function is strictly decreasing during each iteration, i.e. $J(\theta^{[t]}) < J(\theta^{[t-1]})$.

Hint: You can assume that, by Taylor's theorem, the following statement is true:

$$J(y) = J(x) + \nabla J(x)^\top (y-x) + \frac{1}{2}(y-x)^\top \nabla^2 J(x + c(y-x))(y-x) \text{ for some } 0 \le c \le 1$$

Hint: The Hessian of a convex function is symmetric and positive semi-definite at any point θ, which means all of its eigenvalues are real and non-negative.

Hint: The Hessian matrix is symmetric. Consider using the spectral theorem introduced in problem 3 in homework 0.

Optional (no credit): Using the gradient descent inequality that you derived, show that the GD algorithm converges in the sense that $\lim_{t\to\infty} ||\nabla J(\theta^{[t]})||_2^2 = 0$.[4]

Remark: This question suggests that a smaller learning rate should be applied if we believe the curvature of the objective function is big.

Answer: First, using hint 3, we can see that

$$\begin{aligned} H &= U\Lambda U^\top \\ &\preceq U\beta_{\max} I U^\top \\ &= \beta_{\max} U U^\top \\ &= \beta_{\max} I \end{aligned}$$

By Taylor's theorem for $J(\theta)$ around $\theta^{[t-1]}$ and then plugging in $\theta = \theta^{[t]}$

$$\begin{aligned} J(\theta^{[t]}) &= J(\theta^{[t-1]}) + \nabla J(\theta^{[t-1]})^\top (\theta^{[t]} - \theta^{[t-1]}) + \frac{1}{2}(\theta^{[t]} - \theta^{[t-1]})^\top \nabla^2 f(\theta^{[t-1]} + c(\theta^{[t]} - \theta^{[t-1]}))(\theta^{[t]} - \theta^{[t-1]}) \\ &\le J(\theta^{[t-1]}) + \nabla J(\theta^{[t-1]})^\top (\theta^{[t]} - \theta^{[t-1]}) + \frac{1}{2}\beta_{\max} ||\theta^{[t]} - \theta^{[t-1]}||_2^2 \end{aligned}$$

[3]This definition of convergence is different from the definition in previous questions where we explicitly prove that the parameter θ converges to an optimal parameter θ^*. In this case, we do not know the optimal value θ^*, and it would be difficult to prove convergence using the previous definition.

[4]Note that $\lim_{t\to\infty} ||\nabla J(\theta^{[t]})||_2^2 = 0$ does not necessarily mean that there exists a vector $\hat{\theta}$ such that $\theta^{[t]} \to \hat{\theta}$. (Even an 1-dimensional counterexample exists.) However, for convex functions, when $\theta^{[t]}$ stays in a bounded set, $\lim_{t\to\infty} ||\nabla J(\theta^{[t]})||_2^2 = 0$ does imply that $J(\theta^{[t]}) \to \inf_\theta J(\theta)$ as $t \to \infty$. Therefore, $\lim_{t\to\infty} ||\nabla J(\theta^{[t]})||_2^2 = 0$ is typically considered a reasonable definition for an optimization algorithm's convergence.

Now plug in the update formula $\theta^{[t]} = \theta^{[t-1]} - \alpha\nabla J(\theta^{[t-1]})$

$$\begin{aligned}
J(\theta^{[t]}) &\leq J(\theta^{[t-1]}) + \nabla J(\theta^{[t-1]})^\top(\theta^{[t]} - \theta^{[t-1]}) + \frac{1}{2}\beta_{\max}||\theta^{[t]} - \theta^{[t-1]}||_2^2 \\
&= J(\theta^{[t-1]}) + \nabla J(\theta^{[t-1]})^\top(\theta^{[t-1]} - \alpha\nabla J(\theta^{[t-1]}) - \theta^{[t-1]}) + \frac{1}{2}\beta_{\max}||\theta^{[t-1]} - \alpha\nabla J(\theta^{[t-1]}) - \theta^{[t-1]}||_2^2 \\
&= J(\theta^{[t-1]}) - \nabla J(\theta^{[t-1]})^\top(\alpha\nabla J(\theta^{[t-1]})) + \frac{1}{2}\beta_{\max}||\alpha\nabla J(\theta^{[t-1]})||_2^2 \\
&= J(\theta^{[t-1]}) - \alpha||\nabla J(\theta^{[t-1]})||_2^2 + \frac{1}{2}\beta_{\max}\alpha^2||\nabla J(\theta^{[t-1]})||_2^2 \\
&= J(\theta^{[t-1]}) - (1 - \frac{1}{2}\alpha\beta_{\max})\alpha||\nabla J(\theta^{[t-1]})||_2^2
\end{aligned}$$

When $\theta^{[t-1]}$ is not at an optimal solution, the gradient $\nabla J(\theta^{[t-1]}) \neq 0$, so $||\nabla J(\theta^{[t-1]})||_2^2 > 0$.
Since $0 < \alpha < \frac{1}{\beta_{\max}}$, $(1 - \frac{1}{2}\alpha\beta_{\max}) > \frac{1}{2}$ and $\alpha > 0$.
Thus, $(1 - \frac{1}{2}\alpha\beta_{\max})\alpha||\nabla J(\theta^{[t-1]})||_2^2 > 0$, which means that $J(\theta^{[t]})$ is strictly decreasing.

Solution for optional part

By rearranging the inequality above, we have that

$$0 \leq (1 - \frac{1}{2}\alpha\beta_{\max})\alpha||\nabla J(\theta^{[t-1]})||_2^2 \leq J(\theta^{[t-1]}) - J(\theta^{[t]})$$

$$0 \leq ||\nabla J(\theta^{[t-1]})||_2^2 \leq \frac{J(\theta^{[t-1]}) - J(\theta^{[t]})}{(1 - \frac{1}{2}\alpha\beta_{\max})\alpha}$$

Now take the limit as $t \to \infty$. Since $\lim_{t\to\infty} \frac{J(\theta^{[t-1]}) - J(\theta^{[t]})}{(1-\frac{1}{2}\alpha\beta_{\max})\alpha} = 0$, this shows that $\lim_{t\to\infty} ||\nabla J(\theta^{[t-1]})||_2^2 = 0$.

(e) **[2 points] Learning Rate for Linear Regression**

Consider using GD on the LMS objective introduced in lecture:

$$J(\theta) = \frac{1}{2}||X\theta - y||_2^2$$

where $X \in \mathbb{R}^{n\times d}$ is the design matrix of our data, $\theta \in \mathbb{R}^d$ is our parameter, and $\vec{y} \in \mathbb{R}^n$ is the response variable.

Let $\beta_{\max}$ be the largest eigenvalue of $X^\top X$. Prove that for all $\alpha \in (0, \frac{1}{\beta_{\max}})$, GD with learning rate α satisfies that $J(\theta^{[t]})$ converges as $t \to \infty$.

Hint: You can invoke the statements in the part (d) (including the optional question in part (d)) to solve this part (even if you were not able to prove part (d).)

Remark: The conclusion here suggests that for linear regression, roughly speaking, you should use smaller learning rates if the scale of your data X is big or if the data points are correlated with each other, both of which will enable a large top eigenvalue of $XX^\top$.

Remark: Even though in many cases the LMS objective can be solved exactly by solving the normal equation as shown in lecture, it is still useful to be able to guarantee convergence when using the Gradient Descent algorithm in situations where it is difficult to solve for the optimal θ^* directly (such as having large amount of data making inverting X very expensive).

Answer: Referring to the lecture notes, we can see that

$$\nabla J(\theta) = X^\top X\theta - X^\top y$$

So it is easy to see that

$$\nabla^2 J(\theta) = X^\top X$$

Let $\beta_{\max}$ be the largest eigenvalue of $X^\top X$. By part (d), the range is $0 < \alpha < \frac{1}{\beta_{\max}}$

2. **[25 points] Locally Weighted Linear Regression**

(a) [10 points] Consider a linear regression problem in which we want to "weight" different training examples differently. Specifically, suppose we want to minimize

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n} w^{(i)} \left(\theta^T x^{(i)} - y^{(i)}\right)^2 .$$

In class, we worked out what happens for the case where all the weights (the $w^{(i)}$'s) are the same. In this problem, we will generalize some of those ideas to the weighted setting. While this problem is self-contained, you can reference section 1.4 in the lecture notes for additional information on locally weighted linear regression.
We will assume $w^{(i)} > 0$ for all i.

i. [2 points] Show that $J(\theta)$ can also be written

$$J(\theta) = (X\theta - y)^T W (X\theta - y)$$

for an appropriate matrix W, and where X and y are as defined in class. Clearly specify the value of each element of the matrix W.

ii. [4 points] If all the $w^{(i)}$'s equal 1, then we saw in class that the normal equation is

$$X^T X\theta = X^T y,$$

and that the value of θ that minimizes $J(\theta)$ is given by $(X^T X)^{-1} X^T y$. By finding the derivative $\nabla_\theta J(\theta)$ and setting that to zero, generalize the normal equation to this weighted setting, and give the new value of θ that minimizes $J(\theta)$ in closed form as a function of X, W and y.

iii. [4 points] Suppose we have a dataset $\{(x^{(i)}, y^{(i)}); i = 1 \ldots, n\}$ of n independent examples, but we model the $y^{(i)}$'s as drawn from conditional distributions with different levels of variance $(\sigma^{(i)})^2$. Specifically, assume the model

$$p(y^{(i)}|x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi}\sigma^{(i)}} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2(\sigma^{(i)})^2}\right)$$

That is, each $y^{(i)}$ is drawn from a Gaussian distribution with mean $\theta^T x^{(i)}$ and variance $(\sigma^{(i)})^2$ (where the $\sigma^{(i)}$'s are fixed, known, constants). Show that finding the maximum likelihood estimate of θ reduces to solving a weighted linear regression problem. State clearly what the $w^{(i)}$'s are in terms of the $\sigma^{(i)}$'s.
In other words, this suggests that if we have prior knowledge on the noise levels (the variance of the label $y^{(i)}$ conditioned on $x^{(i)}$) of all the examples, then we should use weighted least squares with weights depending on the variances.

Answer:

i. Let $W_{ii} = \frac{1}{2}w^{(i)}, W_{ij} = 0$ for $i \neq j$, let $\vec{z} = X\theta - \vec{y}$, i.e. $z_i = \theta^T x^{(i)} - y^{(i)}$. Then we have:

$$\begin{aligned}(X\theta - \vec{y})^T W (X\theta - \vec{y}) &= \vec{z}^T W \vec{z} \\ &= \frac{1}{2}\sum_{i=1}^{n} w^{(i)} z_i^2 \\ &= \frac{1}{2}\sum_{i=1}^{n} w^{(i)} (\theta^T x^{(i)} - y^{(i)})^2 \\ &= J(\theta)\end{aligned}$$

ii.
$$\nabla_\theta J(\theta) = \nabla_\theta(\theta^T X^T W X\theta + \vec{y}^T W\vec{y} - 2\vec{y}^T W X\theta) = 2(X^T W X\theta - X^T W\vec{y}),$$

so we have $\nabla_\theta J(\theta) = 0$ if and only if

$$X^T W X\theta = X^T W\vec{y}$$

These are the normal equations, from which we can get a closed form formula for θ.

$$\theta = (X^T W X)^{-1} X^T W\vec{y}$$

iii.
$$\begin{aligned}\arg\max_\theta \prod_{i=1}^n p(y^{(i)}|x^{(i)};\theta) &= \arg\max_\theta \sum_{i=1}^n \log p(y^{(i)}|x^{(i)};\theta)\\ &= \arg\max_\theta \sum_{i=1}^n \left(\log\frac{1}{\sqrt{2\pi}\sigma^{(i)}} - \frac{(y^{(i)}-\theta^T x^{(i)})^2}{2(\sigma^{(i)})^2}\right)\\ &= \arg\max_\theta - \sum_{i=1}^n \frac{(y^{(i)}-\theta^T x^{(i)})^2}{2(\sigma^{(i)})^2}\\ &= \arg\min_\theta \frac{1}{2}\sum_{i=1}^n \frac{1}{(\sigma^{(i)})^2}(y^{(i)}-\theta^T x^{(i)})^2\\ &= \arg\min_\theta \frac{1}{2}\sum_{i=1}^n w^{(i)}(y^{(i)}-\theta^T x^{(i)})^2\end{aligned}$$

where in the last step, we substituted: $w^{(i)} = \frac{1}{(\sigma^{(i)})^2}$ to get the linear regression form.

(b) [10 points] **Coding problem.**

We will now consider the following dataset:

`src/lwr/{train,valid,test}.csv`

Each file contains n examples, one example $(x^{(i)}, y^{(i)})$ per row. In particular, the i-th row contains columns $x_1^{(i)} \in \mathbb{R}$ and $y^{(i)} \in \{0,1\}$.

In `src/lwr/lwr.py`, implement locally weighted linear regression using the normal equations you derived in Part (a) and using

$$w^{(i)} = \exp\left(-\frac{\|x^{(i)} - x\|_2^2}{2\tau^2}\right).$$

This is a fairly standard choice for weights, where the weight $w^{(i)}$ depends on the particular point x at which we're trying to evaluate y: if $|x^{(i)} - x|$ is small, then $w^{(i)}$ is close to 1; if $|x^{(i)} - x|$ is large, then $w^{(i)}$ is close to 0. Here, τ is the bandwidth parameter, and it controls how quickly the weight of a training example falls off with distance of its $x^{(i)}$ from the query point x.

Train your model on the `train` split using $\tau = 0.5$, then run your model on the `valid` split and report the mean squared error (MSE). Finally plot your model's predictions on the validation set (plot the training set with blue 'x' markers and the predictions on the validation set with a red 'o' markers). Does the model seem to be under- or overfitting?

Answer: See `../src/lwr/lwr.py` for code. Validation MSE: 0.3305. Underfitting.

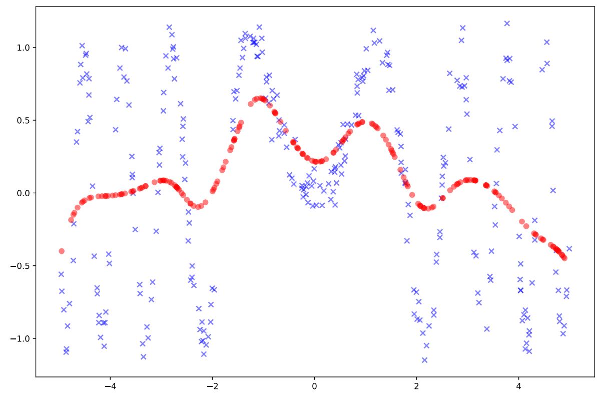


Figure 1: Training set and first validation example with predicted regression line using $\tau = 0.5$.

(c) [5 points] **Coding problem.**

We will now tune the hyperparameter τ. In `src/lwr/tau.py`, find the MSE value of your model on the validation set for each of the values of τ specified in the code. For each τ, plot your model's predictions on the validation set in the format described in part (b). Please include each of the plots separately in your answer. Report the value of τ which achieves the lowest MSE on the `valid` split, and finally report the MSE on the `test` split using this τ-value.

Answer: See `src/lwr/tau.py` for code.

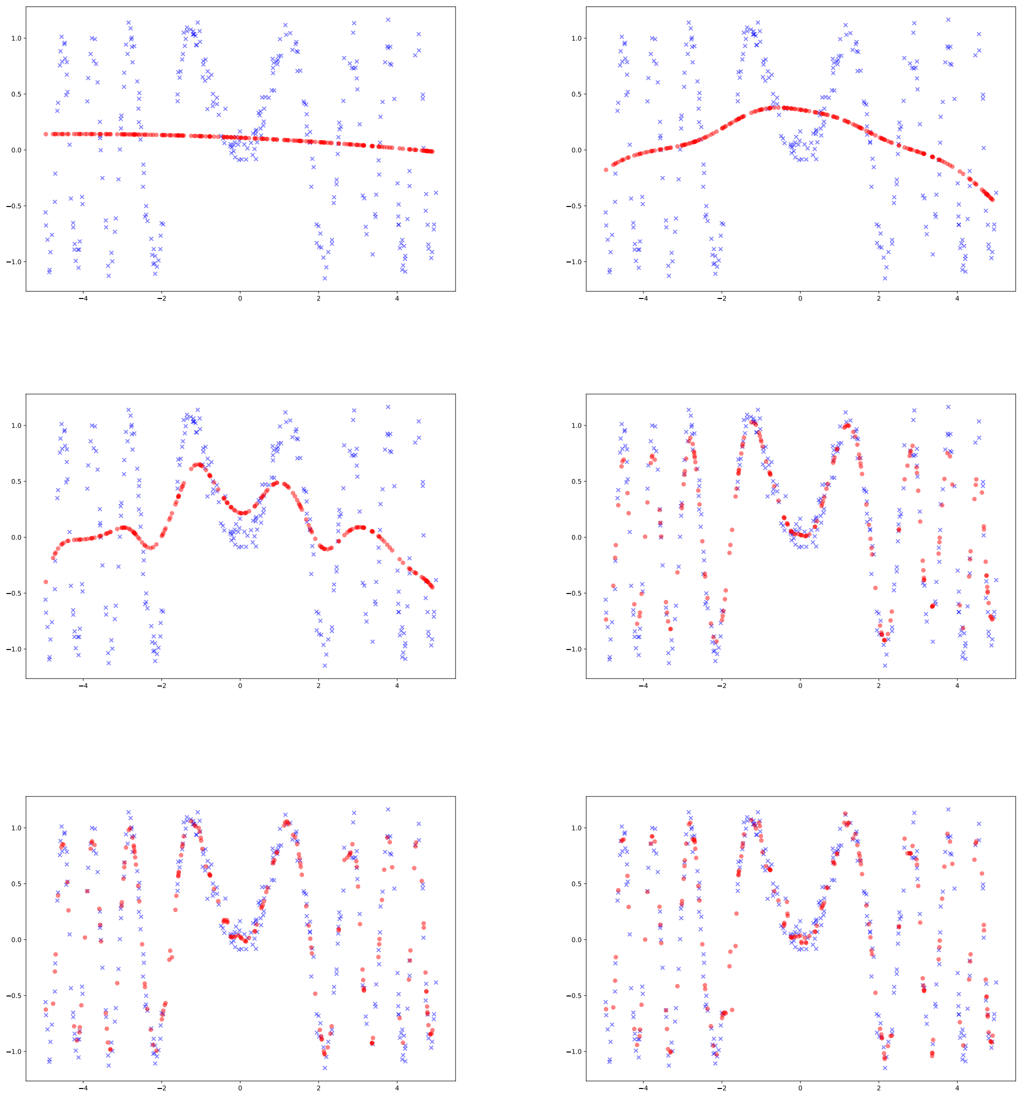


Figure 2: LWR predictions on the validation set for $\tau = 10, 1, 0.5, 0.1, 0.05, 0.03$ (from left-to-right, then top-to-bottom). Notice the progression from underfitting to overfitting as we decrease τ.

$\tau^* = 0.05$
$\mathsf{MSE}_{\texttt{valid}} = 0.0124001$
$\mathsf{MSE}_{\texttt{test}} = 0.0169901.$

3. **[25 points] Linear Regression: Linear in What?**

In the first two lectures, you have seen how to fit a linear function of the data for the regression problem. In this question, we will see how linear regression can be used to fit non-linear functions of the data using feature maps. We will also explore some of its limitations, for which future lectures will discuss fixes.

(a) [5 points] **Learning degree-3 polynomials of the input**

Suppose we have a dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ where $x^{(i)}, y^{(i)} \in \mathbb{R}$. We would like to fit a third degree polynomial $h_\theta(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0$ to the dataset. The key observation here is that the function $h_\theta(x)$ is still linear in the unknown parameter θ, even though it's not linear in the input x. This allows us to convert the problem into a linear regression problem as follows.

Let $\phi : \mathbb{R} \to \mathbb{R}^4$ be a function that transforms the original input x to a 4-dimensional vector defined as

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \end{bmatrix} \in \mathbb{R}^4 \tag{1}$$

Let $\hat{x} \in \mathbb{R}^4$ be a shorthand for $\phi(x)$, and let $\hat{x}^{(i)} \triangleq \phi(x^{(i)})$ be the transformed input in the training dataset. We construct a new dataset $\{(\phi(x^{(i)}), y^{(i)})\}_{i=1}^{n} = \{(\hat{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$ by replacing the original inputs $x^{(i)}$'s by $\hat{x}^{(i)}$'s. We see that fitting $h_\theta(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0$ to the old dataset is equivalent to fitting a linear function $h_\theta(\hat{x}) = \theta_3 \hat{x}_3 + \theta_2 \hat{x}_2 + \theta_1 \hat{x}_1 + \theta_0$ to the new dataset because

$$h_\theta(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0 = \theta_3 \phi(x)_3 + \theta_2 \phi(x)_2 + \theta_1 \phi(x)_1 + \theta_0 = \theta^T \hat{x} \tag{2}$$

In other words, we can use linear regression on the new dataset to find parameters $\theta_0, \ldots, \theta_3$. Please write down 1) the objective function $J(\theta)$ of the linear regression problem on the new dataset $\{(\hat{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$ and 2) the update rule of the batch gradient descent algorithm for linear regression on the dataset $\{(\hat{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$.

Terminology: In machine learning, ϕ is often called the feature map which maps the original input x to a new set of variables. To distinguish between these two sets of variables, we will call x the input **attributes**, and call $\phi(x)$ the **features**. (Unfortunately, different authors use different terms to describe these two things. In this course, we will do our best to follow the above convention consistently.)

Answer:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{n} (\theta^T \hat{x}^{(i)} - y^{(i)})^2$$

Differentiating this objective, we get the update rule:

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_{i=1}^{n} (\theta^T \hat{x}^{(i)} - y^{(i)})(\hat{x}^{(i)}) \\ &\propto \lambda \sum_{i=1}^{n} (\theta^T \hat{x}^{(i)} - y^{(i)})(\hat{x}^{(i)}) \end{aligned}$$

Where λ is the learning rate. The update rule is:

$$\theta := \theta - \lambda \sum_{i=1}^{n} (\theta^T \hat{x}^{(i)} - y^{(i)})(\hat{x}^{(i)})$$

(b) [5 points] **Coding question: degree-3 polynomial regression**

For this sub-question question, we will use the dataset provided in the following files:

`src/featuremaps/{train,valid,test}.csv`

Each file contains two columns: x and y. In the terminology described in the introduction, x is the attribute (in this case one dimensional) and y is the output label.

Using the formulation of the previous sub-question, implement linear regression with **normal equations** using the feature map of degree-3 polynomials. Use the starter code provided in `src/featuremaps/featuremap.py` to implement the algorithm.

Create a scatter plot of the training data, and plot the learnt hypothesis as a smooth curve over it. Submit the plot in the writeup as the solution for this problem.

Remark: Suppose $\widehat{X}$ is the design matrix of the transformed dataset. You may sometimes encounter a non-invertible matrix $\widehat{X}^T \widehat{X}$. For a numerically stable code implementation, always use `np.linalg.solve` to obtain the parameters directly, rather than explicitly calculating the inverse and then multiplying it with $\widehat{X}^T y$.

Answer:

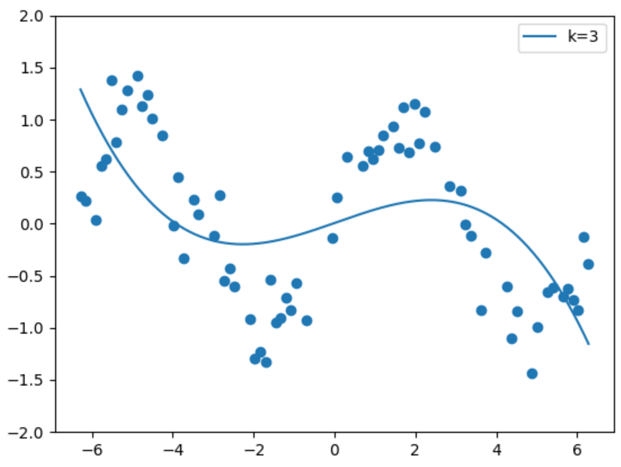


Figure 3: Polynomial regression with degree 3

(c) [5 points] **Coding question: degree-k polynomial regression**

Now we extend the idea above to degree-k polynomials by considering $\phi : \mathbb{R} \to \mathbb{R}^{k+1}$ to be

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \end{bmatrix} \in \mathbb{R}^{k+1} \tag{3}$$

Follow the same procedure as the previous sub-question, and implement the algorithm with $k = 3, 5, 10, 20$. Create a similar plot as in the previous sub-question, and include the hypothesis curves for each value of k with a different color. Include a legend in the plot to indicate which color is for which value of k.

Submit the plot in the writeup as the solution for this sub-problem. Observe how the fitting of the training dataset changes as k increases. Briefly comment on your observations in the plot.

Answer:

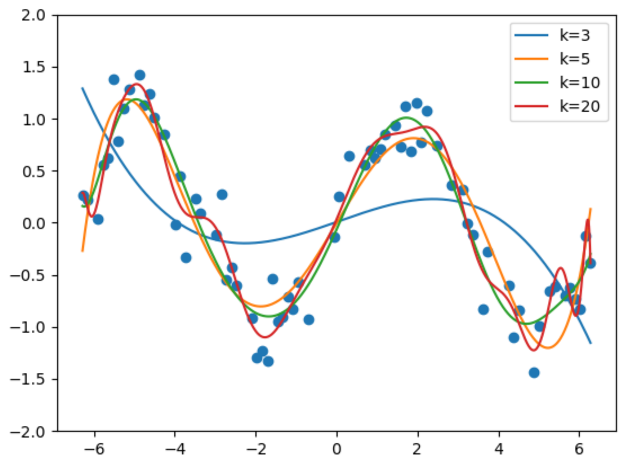


Figure 4: Polynomial regression with kernel sizes 3,5,10 and 20

Generally higher degree polynomials tend to fit the data better, though very high degree polynomials can be numerically unstable.

(d) [5 points] **Coding question: other feature maps**

You may have observed that it requires a relatively high degree k to fit the given training data, and this is because the dataset cannot be explained (i.e., approximated) very well by low-degree polynomials. By visualizing the data, you may have realized that y can be approximated well by a sine wave. In fact, we generated the data by sampling from $y = \sin(x) + \xi$, where ξ is noise with Gaussian distribution. Please update the feature map ϕ to include a sine transformation as follows:

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \\ \sin(x) \end{bmatrix} \in \mathbb{R}^{k+2} \tag{4}$$

With the updated feature map, train different models for values of $k = 0, 1, 2, 3, 5, 10, 20$, and plot the resulting hypothesis curves over the data as before.

Submit the plot as a solution to this sub-problem. Compare the fitted models with the previous sub-question, and briefly comment about noticeable differences in the fit with this feature map.

Answer:

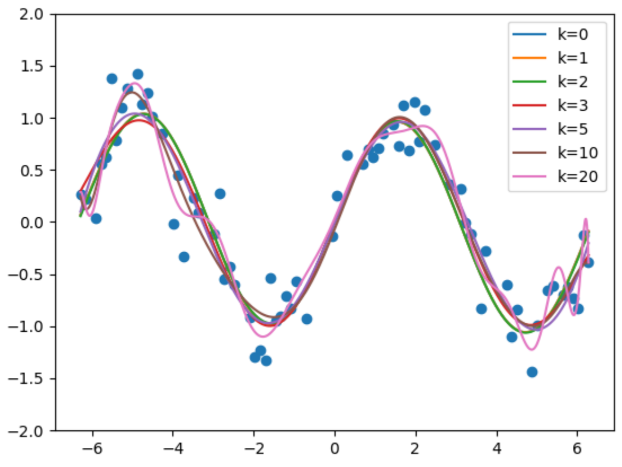


Figure 5: Polynomial regression with other features with kernel sizes 0,1,2,3,5,10 and 20

In the presence of the $\sin(x)$ feature, the models seem to fit the data better / more robustly. However the numerical instability that comes with high degree polynomials remain.

(e) [5 points] **Overfitting with expressive models and small data**

For the rest of the problem, we will consider a small dataset (a random subset of the dataset you have been using so far) with much fewer examples, provided in the following file:

`src/featuremaps/small.csv`

We will be exploring what happens when the number of features start becoming bigger than the number of examples in the training set. Run your algorithm on this small dataset using the following feature map

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \end{bmatrix} \in \mathbb{R}^{k+1} \tag{5}$$

with $k = 1, 2, 5, 10, 20$.

Create a plot of the various hypothesis curves (just like previous sub-questions). Observe how the fitting of the training dataset changes as k increases. Submit the plot in the writeup and comment on what you observe.

Remark: The phenomenon you observe where the models start to fit the training dataset very well, but suddenly "goes wild" is due to what is called *overfitting*. The intuition to have for now is that, when the amount of data you have is small relative to the expressive capacity of the family of possible models (that is, the hypothesis class, which, in this case, is the family of all degree k polynomials), it results in overfitting.

Loosely speaking, the set of hypothesis function is "very flexible" and can be easily forced to pass through all your data points especially in unnatural ways. In other words, the model explains the noises in the training dataset, which shouldn't be explained in the first place. This hurts the predictive power of the model on test examples. We will describe overfitting in more detail in future lectures when we cover learning theory and bias-variance tradeoffs.

Answer:

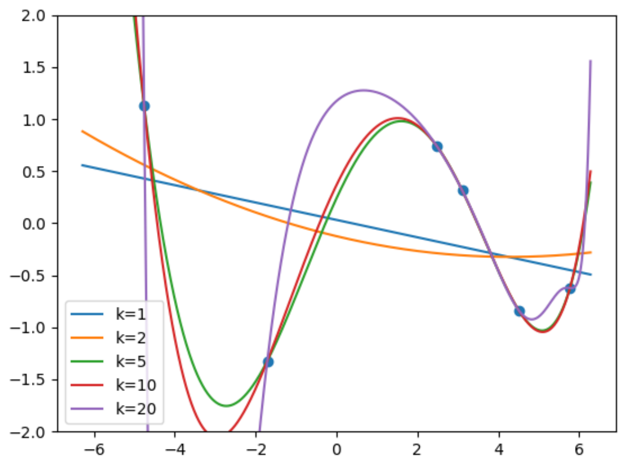


Figure 6: Polynomial regression with kernel sizes 1,2,5,10 and 20 on small dataset

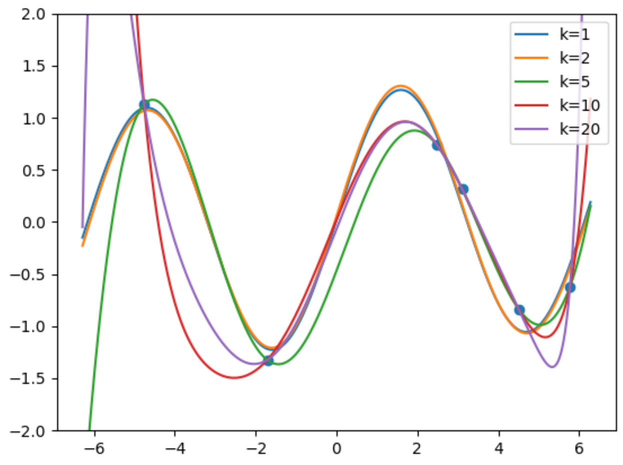


Figure 7: Regression with other features with kernel sizes 1,2,5,10 and 20 on small dataset

We see that when the dataset is small, higher degree polynomials tend to pass through all the points, but qualitatively seem like a poor fit. Numerical instability with high degree polynomials remain a problem even with small data, with or without sin(x).

4. **[25 points] Poisson Regression**

In this question we will construct another kind of a commonly used GLM, which is called Poisson Regression. In a GLM, the choice of the exponential family distribution is based on the kind of problem at hand. If we are solving a classification problem, then we use an exponential family distribution with support over discrete classes (such as Bernoulli, or Categorical). Similarly, if the output is real valued, we can use Gaussian or Laplace (both are in the exponential family). Sometimes the desired output is to predict counts, for example, predicting the number of emails expected in a day, or the number of customers expected to enter a store in the next hour, etc. based on input features (also called covariates). You may recall that a probability distribution with support over integers (i.e., counts) is the Poisson distribution, and it also happens to be in the exponential family.

In the following sub-problems, we will start by showing that the Poisson distribution is in the exponential family, derive the functional form of the hypothesis, derive the update rules for training models, and finally using the provided dataset to train a real model and make predictions on the test set.

(a) [5 points] Consider the Poisson distribution parameterized by λ:

$$p(y; \lambda) = \frac{e^{-\lambda}\lambda^y}{y!}.$$

(Here y has positive integer values and $y!$ is the factorial of y.) Show that the Poisson distribution is in the exponential family, and clearly state the values for $b(y)$, η, $T(y)$, and $a(\eta)$. Note your answer of $a(\eta)$ should be a function of η only.

Answer: Rewrite the distribution function as:

$$\begin{aligned} p(y; \lambda) &= \frac{e^{-\lambda} e^{y \log \lambda}}{y!} \qquad (6) \\ &= \frac{1}{y!} \exp(y \log \lambda - \lambda) \end{aligned}$$

Comparing with the standard form for the exponential family:

$$\begin{aligned} b(y) &= \frac{1}{y!} \qquad (7) \\ \eta &= \log \lambda \qquad (8) \\ T(y) &= y \qquad (9) \\ a(\eta) &= e^{\eta}. \end{aligned}$$

(b) [3 points] Consider performing regression using a GLM model with a Poisson response variable. What is the canonical response function for the family? (You may use the fact that a Poisson random variable with parameter λ has mean λ.)

Answer: The canonical response function for the GLM model will be:

$$\begin{aligned} g(\eta) &= E[y; \eta] \qquad (10) \\ &= \lambda \qquad (11) \\ &= e^{\eta}. \end{aligned}$$

(c) [7 points] For a training set $\{(x^{(i)}, y^{(i)}); i = 1, \ldots, n\}$, let the log-likelihood of an example be $\log p(y^{(i)}|x^{(i)}; \theta)$. By taking the derivative of the log-likelihood with respect to θ_j, derive the stochastic gradient ascent update rule for learning using a GLM model with Poisson responses y and the canonical response function.

Answer: The log-likelihood of an example $(x^{(i)}, y^{(i)})$ is defined as $\ell(\theta) = \log p(y^{(i)}|x^{(i)}; \theta)$. To derive the stochastic gradient ascent rule, use the results in part (a) and the standard GLM assumption that $\eta = \theta^T x$.

$$\begin{aligned}
\frac{\partial \ell(\theta)}{\partial \theta_j} &= \frac{\partial \log p(y^{(i)}|x^{(i)}; \theta)}{\partial \theta_j} && (12)\\
&= \frac{\partial \log \left(\frac{1}{y^{(i)}!} \exp(\eta^T y^{(i)} - e^{\eta})\right)}{\partial \theta_j} && (13)\\
&= \frac{\partial \log \left(\exp((\theta^T x^{(i)})^T y^{(i)} - e^{\theta^T x^{(i)}})\right)}{\partial \theta_j} + \frac{\partial \log \left(\frac{1}{y^{(i)}!}\right)}{\partial \theta_j} && (14)\\
&= \frac{\partial \left((\theta^T x^{(i)})^T y^{(i)} - e^{\theta^T x^{(i)}}\right)}{\partial \theta_j} && (15)\\
&= \frac{\partial \left((\sum_k \theta_k x_k^{(i)}) y^{(i)} - e^{\sum_k \theta_k x_k^{(i)}}\right)}{\partial \theta_j} && (16)\\
&= x_j^{(i)} y^{(i)} - e^{\sum_k \theta_k x_k^{(i)}} x_j^{(i)} && (17)\\
&= (y^{(i)} - e^{\theta^T x^{(i)}}) x_j^{(i)}.
\end{aligned}$$

Thus the stochastic gradient ascent update rule should be:

$$\theta_j := \theta_j + \alpha \frac{\partial \ell(\theta)}{\partial \theta_j},$$

which reduces here to:

$$\theta_j := \theta_j + \alpha (y^{(i)} - e^{\theta^T x^{(i)}}) x_j^{(i)}.$$

(d) [10 points] **Coding problem**

Consider a website that wants to predict its daily traffic. The website owners have collected a dataset of past traffic to their website, along with some features which they think are useful in predicting the number of visitors per day. The dataset is split into train/valid sets and the starter code is provided in the following files:

- `src/poisson/{train,valid}.csv`
- `src/poisson/poisson.py`

We will apply Poisson regression to model the number of visitors per day. Note that applying Poisson regression in particular assumes that the data follows a Poisson distribution whose natural parameter is a linear combination of the input features (*i.e.*, $\eta = \theta^T x$). In `src/poisson/poisson.py`, implement Poisson regression for this dataset and use *full batch gradient ascent* to maximize the log-likelihood of θ. For the stopping criterion, check if the change in parameters has a norm smaller than a small value such as 10^{-5}.

Using the trained model, predict the expected counts for the **validation set**, and create a scatter plot between the true counts vs predicted counts (on the validation set). In the scatter plot, let x-axis be the true count and y-axis be the corresponding predicted expected count. Note that the true counts are integers while the expected counts are generally real values.

Answer:

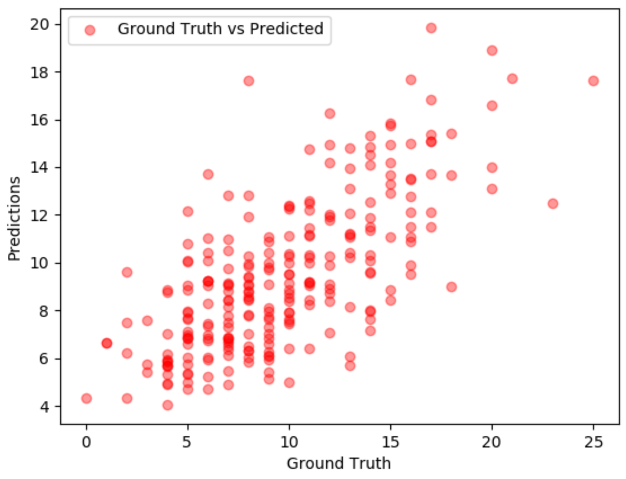


Figure 8: Ground Truth vs Prediction plot on the validation set

Note: Depending on the initial value of θ, the final value of θ may be different. However, the log-likelihoods and predictions are nearly identical. This happens because of the way the data was simulated. x_1 is set to 0 or 1, and $x_2 = 1 - x_1$. Then, with the intercept term, $\theta_0 + \theta_1 x_1 + \theta_2(1 - x_1) = (\theta_0 + \theta_2) + (\theta_1 - \theta_2)x_1$. So we can only recover $(\theta_0 + \theta_2)$, $(\theta_1 - \theta_2)$, θ_3, and θ_4.

5. **[15 points] Convexity of Generalized Linear Models**

In this question we will explore and show some nice properties of Generalized Linear Models, specifically those related to its use of Exponential Family distributions to model the output.

Most commonly, GLMs are trained by using the negative log-likelihood (NLL) as the loss function. This is mathematically equivalent to Maximum Likelihood Estimation (*i.e.,* maximizing the log-likelihood is equivalent to minimizing the negative log-likelihood). In this problem, our goal is to show that the NLL loss of a GLM is a *convex* function w.r.t the model parameters. As a reminder, this is convenient because a convex function is one for which any local minimum is also a global minimum, and there is extensive research on how to optimize various types of convex functions efficiently with various algorithms such as gradient descent or stochastic gradient descent.

To recap, an exponential family distribution is one whose probability density can be represented as

$$p(y;\eta) = b(y)\exp(\eta^T T(y) - a(\eta)),$$

where η is the *natural parameter* of the distribution. Moreover, in a Generalized Linear Model, η is modeled as $\theta^T x$, where $x \in \mathbb{R}^d$ are the input features of the example, and $\theta \in \mathbb{R}^d$ are learnable parameters. In order to show that the NLL loss is convex for GLMs, we break down the process into sub-parts, and approach them one at a time. Our approach is to show that the second derivative (*i.e.,* Hessian) of the loss w.r.t the model parameters is Positive Semi-Definite (PSD) at all values of the model parameters. We will also show some nice properties of Exponential Family distributions as intermediate steps.

For the sake of convenience we restrict ourselves to the case where η is a scalar. Assume $p(Y|X;\theta) \sim \text{ExponentialFamily}(\eta)$, where $\eta \in \mathbb{R}$ is a scalar, and $T(y) = y$. This makes the exponential family representation take the form

$$p(y;\eta) = b(y)\exp(\eta y - a(\eta)).$$

Note that the above probability density is for a single example (x, y).

(a) [5 points] Derive an expression for the mean of the distribution. Show that $\mathbb{E}[Y;\eta] = \frac{\partial}{\partial\eta}a(\eta)$ (note that $\mathbb{E}[Y;\eta] = \mathbb{E}[Y|X;\theta]$ since $\eta = \theta^T x$). In other words, show that the mean of an exponential family distribution is the first derivative of the log-partition function with respect to the natural parameter.

Hint: Start with observing that $\frac{\partial}{\partial\eta}\int p(y;\eta)dy = \int \frac{\partial}{\partial\eta}p(y;\eta)dy$.

Answer: The expectation of an exponential family distribution is first derivative of the log-partition function $a(\eta)$ w.r.t the natural parameters η.

$$\frac{\partial}{\partial\eta}p(y;\eta) = p(y;\eta)\left(y - \frac{\partial}{\partial\eta}a(\eta)\right)$$

$$\Rightarrow \int \frac{\partial}{\partial\eta}p(y;\eta)dy = \int p(y;\eta)\left(y - \frac{\partial}{\partial\eta}a(\eta)\right)dy$$

$$\Rightarrow \frac{\partial}{\partial\eta}\int p(y;\eta)dy = \int yp(y;\eta)dy - \int p(y;\eta)\frac{\partial}{\partial\eta}a(\eta)dy$$

$$\Rightarrow \frac{\partial}{\partial\eta}1 = \mathbb{E}[Y;\eta] - \frac{\partial}{\partial\eta}a(\eta)\int p(y;\eta)dy$$

$$\Rightarrow \mathbb{E}[Y;\eta] = \frac{\partial}{\partial\eta}a(\eta)$$

(b) [5 points] Next, derive an expression for the variance of the distribution. In particular, show that $\mathrm{Var}(Y;\eta) = \frac{\partial^2}{\partial\eta^2}a(\eta)$ (again, note that $\mathrm{Var}(Y;\eta) = \mathrm{Var}(Y|X;\theta)$). In other words, show that the variance of an exponential family distribution is the second derivative of the log-partition function w.r.t. the natural parameter.

Hint: Building upon the result in the previous sub-problem can simplify the derivation.

Answer:

$$\begin{aligned}
\frac{\partial^2}{\partial\eta^2}a(\eta) &= \frac{\partial}{\partial\eta}\mathbb{E}[Y|X;\theta] \\
&= \frac{\partial}{\partial\eta}\int yp(y;\eta)dy \\
&= \int yp(y;\eta)\left(y - \frac{\partial}{\partial\eta}a(\eta)\right)dy \\
&= \int y^2p(y;\eta)dy - \frac{\partial}{\partial\eta}a(\eta)\int yp(y;\eta)dy \\
&= \mathbb{E}[Y^2;\eta] - \mathbb{E}[Y;\eta]^2 \\
&= \mathsf{Var}(Y;\eta).
\end{aligned}$$

(c) [5 points] Finally, write out the loss function $\ell(\theta)$, the NLL of the distribution, as a function of θ. Then, calculate the Hessian of the loss w.r.t θ, and show that it is always PSD. This concludes the proof that NLL loss of GLM is convex.

Hint 1: Use the chain rule of calculus along with the results of the previous parts to simplify your derivations.

Hint 2: Recall that variance of any probability distribution is non-negative.

Answer:

$$\begin{aligned}
\ell(\theta) &= -\log\left[p(y;\eta)\right] \\
&= -\log\left[b(y)\exp(\eta y - a(\eta))\right] \\
&= a(\eta) - \eta y + C \\
&= a(\theta^T x) - \theta^T xy + C
\end{aligned}$$

Gradient of the loss w.r.t. θ is

$$\begin{aligned}
\nabla_\theta\ell(\theta) &= \frac{\partial}{\partial\eta}a(\eta)\nabla_\theta\eta - yx \\
&= \frac{\partial}{\partial\eta}a(\eta)x - yx
\end{aligned}$$

Hessian of the loss w.r.t. θ is

$$\begin{aligned}
\nabla_\theta^2 \ell(\theta) &= \nabla_\theta \left(\nabla_\theta \ell(\theta)\right) \\
&= \nabla_\theta \left(\frac{\partial}{\partial \eta} a(\eta)x - yx\right) \\
&= x\nabla_\theta \left(\frac{\partial}{\partial \eta} a(\eta)\right) \\
&= x\frac{\partial}{\partial \eta}\left(\frac{\partial}{\partial \eta} a(\eta)\right)\nabla_\theta \eta \\
&= \frac{\partial^2}{\partial \eta^2} a(\eta)xx^T \\
&= \mathsf{Var}(Y;\eta)xx^T
\end{aligned}$$

Observe that xx^T is always PSD because for any vector $z \in \mathbb{R}^d$ we have $z^T(xx^T)z = (z^Tx)(x^Tz) = (z^Tx)^2 \geq 0$. Also, for any value of θ, $\mathsf{Var}(Y;\eta)$ is always positive. Therefore the Hessian $\nabla_\theta^2 \ell(\theta) = \mathsf{Var}(Y;\eta)xx^T$ of GLM's NLL loss is PSD, and hence convex. (Note that just saying elements of H is non-negative is not sufficient in showing it is a PSD)

Remark: The main takeaways from this problem are:

- Any GLM model is convex in its model parameters.
- The exponential family of probability distributions are mathematically nice. Whereas calculating mean and variance of distributions in general involves integrals (hard), surprisingly we can calculate them using derivatives (easy) for exponential family.