CS 229, Summer 2020
Problem Set #1

Due Monday, July 13 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 Piazza forum, at `https://piazza.com/stanford/summer2020/cs229`. (3) This quarter, Summer 2020, students may submit in pairs. If you do so, make sure both names are attached to the Gradescope submission. However, students are not allowed to work with the same partner on more than one assignment. 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) To account for late days, the due date is Monday, July 13 at 11:59 pm. If you submit after Monday, July 13 at 11:59 pm, you will begin consuming your late days. If you wish to submit on time, submit before Monday, July 13 at 11:59 pm.

All students must submit an electronic PDF version of the written questions. We highly recommend typesetting your solutions via LaTeX, and we will award one bonus point for typeset submissions. 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.

1. **[40 points] Linear Classifiers (logistic regression and GDA)**

In this problem, we cover two probabilistic linear classifiers we have covered in class so far. First, a discriminative linear classifier: logistic regression. Second, a generative linear classifier: Gaussian discriminant analysis (GDA). Both the algorithms find a linear decision boundary that separates the data into two classes, but make different assumptions. Our goal in this problem is to get a deeper understanding of the similarities and differences (and, strengths and weaknesses) of these two algorithms.

For this problem, we will consider two datasets, along with starter codes provided in the following files:

- `src/linearclass/ds1_{train,valid}.csv`
- `src/linearclass/ds2_{train,valid}.csv`
- `src/linearclass/logreg.py`
- `src/linearclass/gda.py`

Each file contains n examples, one example $(x^{(i)}, y^{(i)})$ per row. In particular, the i-th row contains columns $x_1^{(i)} \in \mathbb{R}$, $x_2^{(i)} \in \mathbb{R}$, and $y^{(i)} \in \{0, 1\}$. In the subproblems that follow, we will investigate using logistic regression and Gaussian discriminant analysis (GDA) to perform binary classification on these two datasets.

(a) [10 points]
In lecture we saw the average empirical loss for logistic regression:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right),$$

where $y^{(i)} \in \{0, 1\}$, $h_\theta(x) = g(\theta^T x)$ and $g(z) = 1/(1+e^{-z})$.
Find the Hessian H of this function, and show that for any vector z, it holds true that

$$z^T H z \geq 0.$$

Hint: You may want to start by showing that $\sum_i \sum_j z_i x_i x_j z_j = (x^T z)^2 \geq 0$. Recall also that $g'(z) = g(z)(1 - g(z))$.
Remark: This is one of the standard ways of showing that the matrix H is positive semi-definite, written "$H \succeq 0$." This implies that J is convex, and has no local minima other than the global one. If you have some other way of showing $H \succeq 0$, you're also welcome to use your method instead of the one above.

(b) [5 points] **Coding problem.** Follow the instructions in `src/linearclass/logreg.py` to train a logistic regression classifier using Newton's Method. Starting with $\theta = \vec{0}$, run Newton's Method until the updates to θ are small: Specifically, train until the first iteration k such that $\|\theta_k - \theta_{k-1}\|_1 < \epsilon$, where $\epsilon = 1 \times 10^{-5}$. Make sure to write your model's predicted probabilities on the validation set to the file specified in the code.
Include a plot of the **validation data** with x_1 on the horizontal axis and x_2 on the vertical axis. To visualize the two classes, use a different symbol for examples $x^{(i)}$ with $y^{(i)} = 0$ than for those with $y^{(i)} = 1$. On the same figure, plot the decision boundary found by logistic regression (i.e, line corresponding to $p(y|x) = 0.5$).

(c) [5 points] Recall that in GDA we model the joint distribution of (x, y) by the following equations:

$$p(y) = \begin{cases} \phi & \text{if } y = 1 \\ 1 - \phi & \text{if } y = 0 \end{cases}$$
$$p(x|y=0) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right)$$
$$p(x|y=1) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right),$$

where ϕ, μ_0, μ_1, and Σ are the parameters of our model.

Suppose we have already fit ϕ, μ_0, μ_1, and Σ, and now want to predict y given a new point x. To show that GDA results in a classifier that has a linear decision boundary, show the posterior distribution can be written as

$$p(y = 1 \mid x; \phi, \mu_0, \mu_1, \Sigma) = \frac{1}{1 + \exp(-(\theta^T x + \theta_0))},$$

where $\theta \in \mathbb{R}^d$ and $\theta_0 \in \mathbb{R}$ are appropriate functions of ϕ, Σ, μ_0, and μ_1.

(d) [7 points] Given the dataset, we claim that the maximum likelihood estimates of the parameters are given by

$$\phi = \frac{1}{n}\sum_{i=1}^{n} 1\{y^{(i)} = 1\}$$
$$\mu_0 = \frac{\sum_{i=1}^{n} 1\{y^{(i)} = 0\}x^{(i)}}{\sum_{i=1}^{n} 1\{y^{(i)} = 0\}}$$
$$\mu_1 = \frac{\sum_{i=1}^{n} 1\{y^{(i)} = 1\}x^{(i)}}{\sum_{i=1}^{n} 1\{y^{(i)} = 1\}}$$
$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}(x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$$

The log-likelihood of the data is

$$\begin{aligned}\ell(\phi, \mu_0, \mu_1, \Sigma) &= \log \prod_{i=1}^{n} p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma) \\ &= \log \prod_{i=1}^{n} p(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma)p(y^{(i)}; \phi).\end{aligned}$$

By maximizing ℓ with respect to the four parameters, prove that the maximum likelihood estimates of ϕ, μ_0, μ_1, and Σ are indeed as given in the formulas above. (You may assume that there is at least one positive and one negative example, so that the denominators in the definitions of μ_0 and μ_1 above are non-zero.)

(e) [5 points] **Coding problem.** In `src/linearclass/gda.py`, fill in the code to calculate ϕ, μ_0, μ_1, and Σ, use these parameters to derive θ, and use the resulting GDA model to make predictions on the validation set. Make sure to write your model's predictions on the validation set to the file specified in the code.

Include a plot of the **validation data** with x_1 on the horizontal axis and x_2 on the vertical axis. To visualize the two classes, use a different symbol for examples $x^{(i)}$ with $y^{(i)} = 0$ than for those with $y^{(i)} = 1$. On the same figure, plot the decision boundary found by GDA (i.e, line corresponding to $p(y|x) = 0.5$).

(f) [2 points] For Dataset 1, compare the validation set plots obtained in part (b) and part (e) from logistic regression and GDA respectively, and briefly comment on your observation in a couple of lines.

(g) [5 points] Repeat the steps in part (b) and part (e) for Dataset 2. Create similar plots on the **validation set** of Dataset 2 and include those plots in your writeup.

On which dataset does GDA seem to perform worse than logistic regression? Why might this be the case?

(h) **[1 points]** For the dataset where GDA performed worse in parts (f) and (g), can you find a transformation of the $x^{(i)}$'s such that GDA performs significantly better? What might this transformation be?

2. **[30 points] Incomplete, Positive-Only Labels**

In this problem we will consider training binary classifiers in situations where we do not have full access to the labels. In particular, we consider a scenario, which is not too infrequent in real life, where we have labels only for a subset of the positive examples. All the negative examples and the rest of the positive examples are unlabelled.

We formalize the scenario as follows. Let $\{(x^{(i)}, t^{(i)})\}_{i=1}^n$ be a standard dataset of i.i.d distributed examples. Here $x^{(i)}$'s are the inputs/features and $t^{(i)}$ are the labels. Now consider the situation where $t^{(i)}$'s are not observed by us. Instead, we only observe the labels of some of the positive examples. Concretely, we assume that we observe $y^{(i)}$'s that are generated by

$$\begin{aligned}
&\forall x, \quad p(y^{(i)} = 1 \mid t^{(i)} = 1, x^{(i)} = x) = \alpha, \\
&\forall x, \quad p(y^{(i)} = 0 \mid t^{(i)} = 1, x^{(i)} = x) = 1 - \alpha \\
&\forall x, \quad p(y^{(i)} = 1 \mid t^{(i)} = 0, x^{(i)} = x) = 0, \\
&\forall x, \quad p(y^{(i)} = 0 \mid t^{(i)} = 0, x^{(i)} = x) = 1
\end{aligned}$$

where $\alpha \in (0, 1)$ is some unknown scalar. In other words, if the unobserved "true" label $t^{(i)}$ is 1, then with α chance we observe a label $y^{(i)} = 1$. On the other hand, if the unobserved "true" label $t^{(i)} = 0$, then we always observe the label $y^{(i)} = 0$.

Our final goal in the problem is to construct a binary classifier h of the true label t, with only access to the partial label y. In other words, we want to construct h such that $h(x^{(i)}) \approx p(t^{(i)} = 1 \mid x^{(i)})$ as closely as possible, using only x and y.

Real world example: Suppose we maintain a database of proteins which are involved in transmitting signals across membranes. Every example added to the database is involved in a signaling process, but there are many proteins involved in cross-membrane signaling which are missing from the database. It would be useful to train a classifier to identify proteins that should be added to the database. In our notation, each example $x^{(i)}$ corresponds to a protein, $y^{(i)} = 1$ if the protein is in the database and 0 otherwise, and $t^{(i)} = 1$ if the protein is involved in a cross-membrane signaling process and thus should be added to the database, and 0 otherwise.

For the rest of the question, we will use the dataset and starter code provided in the following files:

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

Each file contains the following columns: x_1, x_2, y, and t. As in Problem 1, there is one example per row. The $y^{(i)}$'s are generated from the process defined above with some unknown α.

(a) [5 points] **Coding problem: ideal (fully observed) case**

First we will consider the hypothetical (and uninteresting) case, where we have access to the true t-labels for training. In `src/posonly/posonly.py`, write a logistic regression classifier that uses x_1 and x_2 as input features, and train it using the t-labels. We will ignore the y-labels for this part. Output the trained model's predictions on the **test set** to the file specified in the code.

Create a plot to visualize the test set with x_1 on the horizontal axis and x_2 on the vertical axis. Use different symbols for examples $x^{(i)}$ with true label $t^{(i)} = 1$ than those with $t^{(i)} = 0$. On the same figure, plot the decision boundary obtained by your model (i.e, line

corresponding to model's predicted probability = 0.5) in red color. Include this plot in your writeup.

(b) [5 points] **Coding problem: The naive method on partial labels**

We now consider the case where the t-labels are unavailable, so you only have access to the y-labels at training time. Extend your code in `src/posonly/posonly.py` to re-train the classifier (still using x_1 and x_2 as input features), but using the y-labels only. Output the predictions on the **test set** to the appropriate file (as described in the code comments).

Create a plot to visualize the test set with x_1 on the horizontal axis and x_2 on the vertical axis. Use different symbols for examples $x^{(i)}$ with true label $t^{(i)} = 1$ (even though we only used the $y^{(i)}$ labels for training, use the true $t^{(i)}$ labels for plotting) than those with $t^{(i)} = 0$. On the same figure, plot the decision boundary obtained by your model (i.e, line corresponding to model's predicted probability = 0.5) in red color. Include this plot in your writeup.

Note that the algorithm should learn a function $h(\cdot)$ that approximately predicts the probability $p(y^{(i)} = 1 \mid x^{(i)})$. Also note that we expect it to perform poorly on predicting the probability of interest, namely $p(t^{(i)} = 1 \mid x^{(i)})$.

In the following sub-questions we will attempt to solve the problem with only partial observations. That is, we only have access to $\{(x^{(i)}, y^{(i)})\}_{i=1}^n$, and will try to predict $p(t^{(i)} = 1|x^{(i)})$.

(c) [5 points] **Warm-up with Bayes rule**

Show that under our assumptions, for any i,

$$p(t^{(i)} = 1 \mid y^{(i)} = 1, x^{(i)}) = 1 \tag{1}$$

That is, observing a positive partial label $y^{(i)} = 1$ tells us for sure the hidden true label is 1. Use Bayes rule to derive this (an informal explanation will not earn credit).

(d) [5 points] Show that for any example, the probability that true label $t^{(i)}$ is positive is $1/\alpha$ times the probability that the partial label is positive. That is, show that

$$p(t^{(i)} = 1 \mid x^{(i)}) = \frac{1}{\alpha} \cdot p(y^{(i)} = 1 \mid x^{(i)}) \tag{2}$$

Note that the equation above suggests that if we know the value of α, then we can convert a function $h(\cdot)$ that approximately predicts the probability $h(x^{(i)}) \approx p(y^{(i)} = 1 \mid x^{(i)})$ into a function that approximately predicts $p(t^{(i)} = 1 \mid x^{(i)})$ by multiplying the factor $1/\alpha$.

(e) [5 points] **Estimating α**

The solution to estimate $p(t^{(i)}|x^{(i)})$ outlined in the previous sub-question requires the knowledge of α which we don't have. Now we will design a way to estimate α based on the function $h(\cdot)$ that approximately predicts $p(y^{(i)} = 1 \mid x^{(i)})$ (which we obtained in part b).

To simplify the analysis, let's assume that we have magically obtained a function $h(x)$ that perfectly predicts the value of $p(y^{(i)} = 1 \mid x^{(i)})$, that is, $h(x^{(i)}) = p(y^{(i)} = 1 \mid x^{(i)})$.

We make the crucial assumption that $p(t^{(i)} = 1 \mid x^{(i)}) \in \{0, 1\}$. This assumption means that the process of generating the "true" label $t^{(i)}$ is a noise-free process. This assumption is not very unreasonable to make. Note, we are NOT assuming that the observed label $y^{(i)}$ is noise-free, which would be an unreasonable assumption!

Now we will show that:

$$\alpha = \mathbb{E}[h(x^{(i)}) \mid y^{(i)} = 1] \tag{3}$$

To show this, prove that $h(x^{(i)}) = \alpha$ when $y^{(i)} = 1$, and $h(x^{(i)}) = 0$ when $y^{(i)} = 0$.

The above result motivates the following algorithm to estimate α by estimating the RHS of the equation above using samples: Let V_+ be the set of labeled (and hence positive) examples in the validation set V, given by $V_+ = \{x^{(i)} \in V \mid y^{(i)} = 1\}$.

Then we use

$$\alpha \approx \frac{1}{|V_+|} \sum_{x^{(i)} \in V_+} h(x^{(i)}).$$

to estimate α. (You will be asked to implement this algorithm in the next sub-question. For this sub-question, you only need to show equation (3). Moreover, this sub-question may be slightly harder than other sub-questions.)

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

Using the validation set, estimate the constant α by averaging your classifier's predictions over all labeled examples in the validation set:[1]

$$\alpha \approx \frac{1}{|V_+|} \sum_{x^{(i)} \in V_+} h(x^{(i)}).$$

Add code in `src/posonly/posonly.py` to rescale your predictions $h(y^{(i)} = 1 \mid x^{(i)})$ of the classifier that is obtained from part b, using the equation (2) obtained in part (d) and using the estimated value for α.

Finally, create a plot to visualize the test set with x_1 on the horizontal axis and x_2 on the vertical axis. Use different symbols for examples $x^{(i)}$ with true label $t^{(i)} = 1$ (even though we only used the $y^{(i)}$ labels for training, use the true $t^{(i)}$ labels for plotting) than those with $t^{(i)} = 0$. On the same figure, plot the decision boundary obtained by your model (i.e, line corresponding to model's **adjusted** predicted probability = 0.5) in red color. Include this plot in your writeup.

Remark: We saw that the true probability $p(t \mid x)$ was only a constant factor away from $p(y \mid x)$. This means, if our task is to only rank examples (*i.e.* sort them) in a particular order (e.g, sort the proteins in order of being most likely to be involved in transmitting signals across membranes), then in fact we do not even need to estimate α. The rank based on $p(y \mid x)$ will agree with the rank based on $p(t \mid x)$.

[1] There is a reason to use the validation set, instead of the training set, to estimate the α. However, for the purpose of this question, we sweep the subtlety here under the rug, and you don't need to understand the difference between the two for this question.

3. **[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). Simiarly, 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 e.g., 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 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 $a(\eta)$.

(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.)

(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.

(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.

4. **[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

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

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

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

(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 $\frac{\partial}{\partial \eta}\int p(y;\eta)dy = \int \frac{\partial}{\partial \eta} p(y;\eta)dy$.

(b) [5 points] Next, derive an expression for the variance of the distribution. In particular, show that $\text{Var}(Y;\eta) = \frac{\partial^2}{\partial \eta^2} a(\eta)$ (again, note that $\text{Var}(Y;\eta) = \text{Var}(Y|X;\theta)$). 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.

(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.

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.

5. **[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

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

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

$$h_\theta(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0 = \theta_3 \phi(x)_3 + \theta_2 \phi(x)_2 + \theta_1 \phi(x)_1 + \theta_0 = \theta^T \hat{x} \tag{5}$$

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.)

(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 $\widehat{X}^T y$.

(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

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \end{bmatrix} \in \mathbb{R}^{k+1} \tag{6}$$

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.

(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:

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \\ \sin(x) \end{bmatrix} \in \mathbb{R}^{k+2} \tag{7}$$

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 noticable differences in the fit with this feature map.

(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

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \end{bmatrix} \in \mathbb{R}^{k+1} \tag{8}$$

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.