CS 229, Spring 2023
Problem Set #1

YOUR NAME HERE (`YOUR SUNET HERE`)

Due Wednesday, April 26 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/37893/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, April 26 at 11:59 pm. If you submit after Wednesday, April 26 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

1. **[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 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 $a(\eta)$.

Answer: Rewrite the distribution function as:

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

Comparing with the standard form for the exponential family:

$$\begin{aligned} b(y) &= \frac{1}{y!} && (2)\\ \eta &= \log\lambda && (3)\\ T(y) &= y && (4)\\ a(\eta) &= e^{\eta}. \end{aligned}$$

(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:

$$\begin{aligned} g(\eta) &= E[y;\eta] && (5)\\ &= \lambda && (6)\\ &= e^{\eta}. \end{aligned}$$

(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 $\eta = \theta^T x$.

$$\begin{aligned}
\frac{\partial \ell(\theta)}{\partial \theta_j} &= \frac{\partial \log p(y^{(i)}|x^{(i)};\theta)}{\partial \theta_j} && (7)\\
&= \frac{\partial \log\left(\frac{1}{y^{(i)}!}\exp(\eta^T y^{(i)} - e^{\eta})\right)}{\partial \theta_j} && (8)\\
&= \frac{\partial \log\left(\exp((\theta^T x^{(i)})^T y^{(i)} - e^{\theta^T x^{(i)}})\right)}{\partial \theta_j} + \frac{\partial \log\left(\frac{1}{y^{(i)}!}\right)}{\partial \theta_j} && (9)\\
&= \frac{\partial\left((\theta^T x^{(i)})^T y^{(i)} - e^{\theta^T x^{(i)}}\right)}{\partial \theta_j} && (10)\\
&= \frac{\partial\left((\sum_k \theta_k x_k^{(i)}) y^{(i)} - e^{\sum_k \theta_k x_k^{(i)}}\right)}{\partial \theta_j} && (11)\\
&= x_j^{(i)} y^{(i)} - e^{\sum_k \theta_k x_k^{(i)}} x_j^{(i)} && (12)\\
&= (y^{(i)} - e^{\theta^T x^{(i)}}) x_j^{(i)}.
\end{aligned}$$

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:

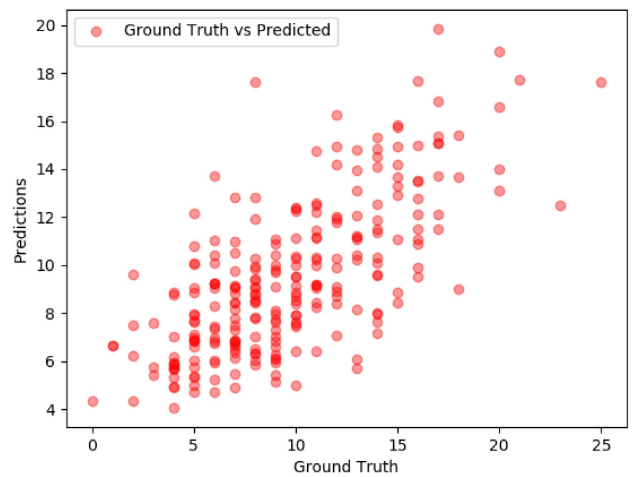

Figure 1: Ground Truth vs Prediction plot on the validation set

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.

2. **[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:

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

(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:

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

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

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

Observe that xx^T is always PSD because for any vector $z \in \mathbb{R}^d$ we have $z^T(xx^T)z = (z^T x)(x^T z) = (z^T x)^2 \geq 0$. Also, for any value of θ, $\mathsf{Var}(Y; \eta)$ is always positive. Therefore the Hessian $\nabla_\theta^2 \ell(\theta) = \mathsf{Var}(Y; \eta) x x^T$ 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.

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

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \end{bmatrix} \in \mathbb{R}^4 \tag{13}$$

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

$$h_\theta(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0 = \theta_3 \phi(x)_3 + \theta_2 \phi(x)_2 + \theta_1 \phi(x)_1 + \theta_0 = \theta^T \hat{x} \tag{14}$$

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:

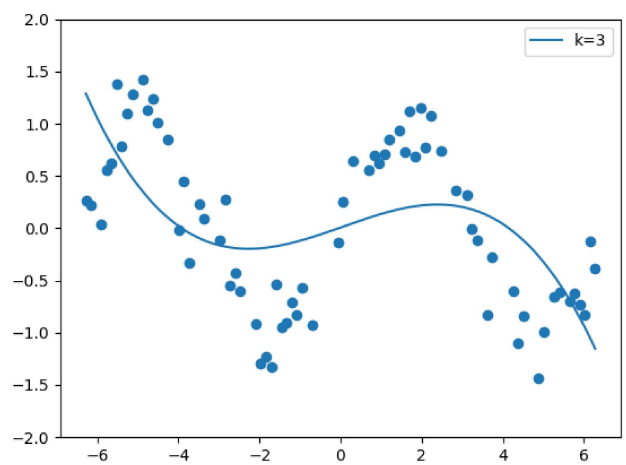

Figure 2: Polynomial regression with degree 3

(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

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \end{bmatrix} \in \mathbb{R}^{k+1} \tag{15}$$

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:**

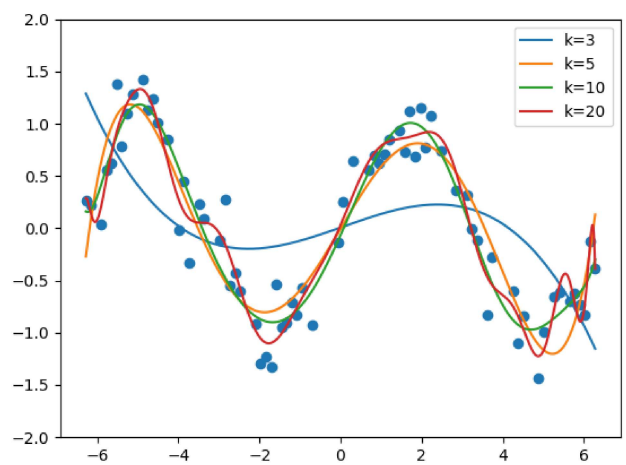

Figure 3: Polynomial regression with kernel sizes 3,5,10 and 20

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:

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \\ \sin(x) \end{bmatrix} \in \mathbb{R}^{k+2} \tag{16}$$

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:**

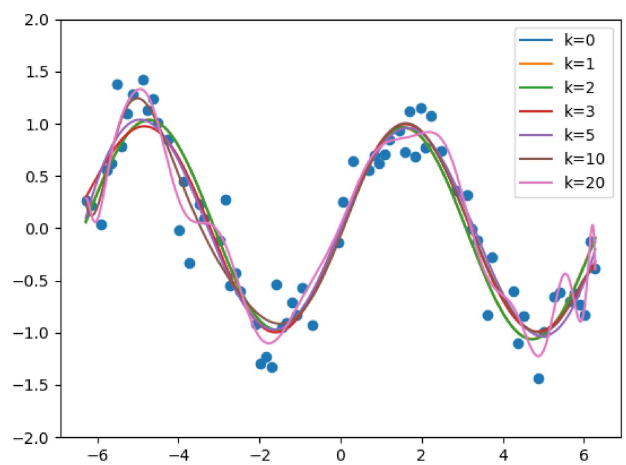

Figure 4: Polynomial regression with other features with kernel sizes 0,1,2,3,5,10 and 20

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

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \end{bmatrix} \in \mathbb{R}^{k+1} \tag{17}$$

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:

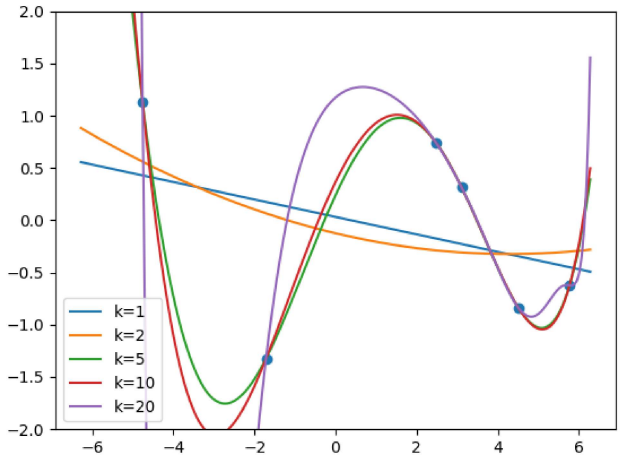

Figure 5: Polynomial regression with kernel sizes 1,2,5,10 and 20 on small dataset

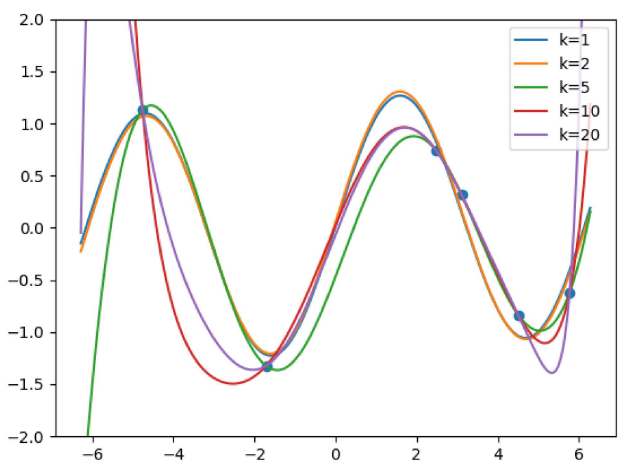

Figure 6: Regression with other features with kernel sizes 1,2,5,10 and 20 on small dataset

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] Learning Imbalanced dataset**

In this problem, we study how to learn a classifier from an imbalanced dataset, where the marginal distribution of the classes/labels are imbalanced. Imbalanced datasets are ubiquitous in real-world applications. For example, in the spam detection problem, the training dataset usually has only a small fraction of spam emails (positive examples) but a large fraction of ordinary emails (negative examples). For simplicity, we consider binary classification problem where the labels are in $\{0, 1\}$ and the number of positive examples is much smaller than the number of negative examples.

(a) [10 points] **Evaluation metrics**

In this sub-question, we discuss the common evaluation metrics for imbalanced dataset. Suppose we have a validation dataset and for some $\rho \in (0, 1)$, we assume that ρ fraction of the validation examples are positive examples (with label 1), and $1 - \rho$ fraction of them are negative examples (with label 0).

Define the accuracy as

$$A \triangleq \frac{\#\text{examples that are predicted correctly by the classifier}}{\#\text{examples}}$$

(i) (3 point) Show that for any dataset with ρ fraction of positive examples and $1 - \rho$ fraction of negative examples, there exists a (trivial) classifier with accuracy at least $1 - \rho$.

The statement above suggests that the accuracy is not an ideal evaluation metric when ρ is close to 0. E.g., imagine that for spam detection ρ can be smaller than 1%. The statement suggests there is a trivial classifier that gets more than 99% accuracy. This could be misleading — 99% seems to be almost perfect, but actually you don't need to learn anything from the dataset to achieve it.

Therefore, for imbalanced dataset, we need more informative evaluation metrics. We define the number of true positive, true negative, false positive, false negative examples as

$$\begin{aligned}
TP &\triangleq \#\text{ positive examples with a correct (positive) prediction} \\
TN &\triangleq \#\text{ negative examples with a correct (negative) prediction} \\
FP &\triangleq \#\text{ negative examples with a incorrect (positive) prediction} \\
FN &\triangleq \#\text{ positive examples with a incorrect (negative) prediction}
\end{aligned}$$

Define the accuracy of positive examples as

$$A_1 \triangleq \frac{TP}{TP + FN} = \frac{\#\text{ positive examples with a correct (positive) prediction}}{\#\text{ positive examples}}$$

Define the accuracy of negative examples as

$$A_0 \triangleq \frac{TN}{TN + FP} = \frac{\#\text{ negative examples with a correct (negative) prediction}}{\#\text{ negative examples}}$$

We define the balanced accuracy as

$$\overline{A} \triangleq \frac{1}{2}(A_0 + A_1) \tag{18}$$

With these notations, we can verify that the accuracy is equal to $A = \frac{TP+TN}{TP+TN+FP+FN}$.
(ii) (4 point) Show that

$$\rho = \frac{TP + FN}{TP + TN + FP + FN}$$

and

$$A = \rho \cdot A_1 + (1 - \rho)A_0 \tag{19}$$

Comparing equation (18) and (19), we can see that the accuracy and balanced accuracy are both linear combination of A_0 and A_1 but with different weighting.
(iii) (3 point) Show that the trivial classifier you constructed for part (i) has balanced accuracy 50%.
Partly because of (iii), the balanced accuracy $\overline{A}$ is often a preferable evaluation metric than the accuracy A. Sometimes people also report the accuracies for the two classes (A_0 and A_1) to demonstrate the performance for each class.
Answer:
(i) The classifier that always outputs negative (label 0).
(ii) The total number of positive examples is $TP + FN$ and the total number of examples is $TP + TN + FP + FN$, so

$$\rho = \frac{TP + FN}{TP + TN + FP + FN} \tag{20}$$

Since $A = \frac{TP+TN}{TP+TN+FP+FN}$,

$$A = \frac{TP + FN}{TP + TN + FP + FN} \cdot \frac{TP}{TP + FN} + \frac{TN + FP}{TP + TN + FP + FN} \cdot \frac{TN}{TN + FP} \tag{21}$$

$$= \rho \cdot A_1 + (1 - \rho)A_0 \tag{22}$$

(iii) The classifier has $TP = 0, TN =$ total number of negative examples, $FP = 0, FN =$ total number of positive examples, so

$$A_1 = 0 \tag{23}$$

$$A_0 = 1 \tag{24}$$

$$\overline{A} = \frac{1}{2} \tag{25}$$

(b) [5 points] **Coding problem: vanilla logistic regression**
First, we use the vanilla logistic regression to learn an imbalanced dataset. For the rest of the question, we will use the dataset and starter code provided in the following files:

- `src/imbalanced/{train,validation}.csv`
- `src/imbalanced/imbalanced.py`

Each file contains n examples, one example $(x^{(i)}, y^{(i)})$ per row. x is two-dimensional, i.e., the i-th row contains columns $x_1^{(i)} \in \mathbb{R}$, $x_2^{(i)} \in \mathbb{R}$, and $y^{(i)} \in \{0, 1\}$. Let $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$ be our training dataset. $\mathcal{D}$ has ρn examples with label 1 and $(1 - \rho)n$ with label 0. In the dataset we constructed, $\rho = 1/11$.

You will train a linear classifier $h_\theta(x)$ with average empirical loss for logistical regression, where $h_\theta(x) = g(\theta^T x), g(z) = 1/(1+e^{-z})$, similar to Problem 1 Part a:

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

You are encouraged to use your code for problem 1, but you are also allowed to use any standard logistic regression library or package to optimize the objective above. After obtaining the classifier, compute the classifier's accuracy (A), balanced accuracy ($\overline{A}$), accuracies for the two classes (A_0, A_1) on the validation dataset, and report them in the writeup. You are expected to observe that the minority class (positive class) has significantly lower accuracy than the majority class.

Create a plot to visualize the validation 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 $y^{(i)} = 1$ than those with $y^{(i)} = 0$. 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.

Answer:

$A_1 = 0.570, A_0 = 0.985, \overline{A} = 0.777, A = 0.947.$

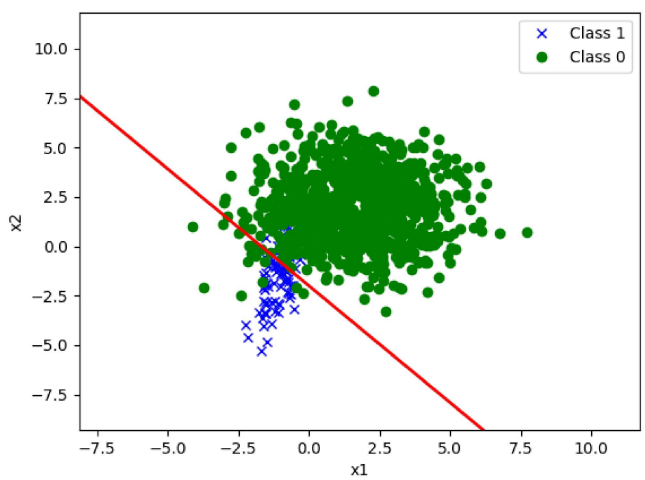

Figure 7: Decision boundary for vanilla logistic regression

(c) [5 points] **Re-sampling/Re-weighting Logistic Regression**

The relatively low accuracy for the minority class and the resulting low balanced accuracy are undesirable for many applications. Various methods have been proposed to improve the accuracy for the minority class, and learning imbalanced datasets is an active open research direction. Here we introduce a simple and classical re-sampling/re-weighting technique that helps improve the balanced accuracy in most of the cases.

We observe that the logistic regression algorithm works well for the accuracy but not for the balanced accuracy. Let $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$ denote the existing training dataset. We will

create a new dataset $\mathcal{D}'$ such that the accuracy on $\mathcal{D}'$ is the same as the balanced accuracy on $\mathcal{D}$.

Assume $\rho < 1/2$ without loss of generality, and let $\kappa = \frac{\rho}{1-\rho}$. This means the number of positive examples is κ times the number of negative examples in $\mathcal{D}$. Assume for convenience $1/\kappa$ is an integer.[3] Let $\mathcal{D}'$ be the dataset that contain each negative example once and $1/\kappa$ repetitions of each positive example in $\mathcal{D}$. Then we will apply the logistic regression on the new dataset $\mathcal{D}'$.

Prove that for any classifier, the balanced accuracy on $\mathcal{D}$ is equal to the accuracy on $\mathcal{D}'$. Moreover, **show that** the average empirical loss for logistical regression on the dataset $\mathcal{D}'$ is equal to

$$J(\theta) = -\frac{1+\kappa}{2n} \sum_{i=1}^{n} w^{(i)} \left(y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right),$$

where $w^{(i)} = 1$ if $y^{(i)} = 0$ and $w^{(i)} = 1/\kappa$ if $y^{(i)} = 1$.

Observe effectively we are re-weighting the loss function for each example by some constant that depends on the frequency of the class it belongs to.

Answer:

Let $FN(\mathcal{D}), FP(\mathcal{D}), TN(\mathcal{D}), TP(\mathcal{D})$ be the number of false negatives, false positives, true negatives, true positives of classifier h on $\mathcal{D}$. Then the corresponding metrics on $\mathcal{D}'$ are: $FN(\mathcal{D}') = FN(\mathcal{D})/\kappa, FP(\mathcal{D}') = FP(\mathcal{D}), TN(\mathcal{D}') = TN(\mathcal{D}), TP(\mathcal{D}') = TP(\mathcal{D})/\kappa$. Let the balanced accuracy on $\mathcal{D}$ be $\overline{A}(\mathcal{D})$, and the accuracy on $\mathcal{D}'$ be $A(\mathcal{D}')$.

$$\begin{aligned}
\overline{A}(\mathcal{D}) &= \frac{1}{2}\left(\frac{TP(\mathcal{D})}{TP(\mathcal{D}) + FN(\mathcal{D})} + \frac{TN(\mathcal{D})}{TN(\mathcal{D}) + FP(\mathcal{D})}\right) && (26)\\
&= \frac{1}{2}\left(\frac{TP(\mathcal{D})}{\kappa(TN(\mathcal{D}) + FP(\mathcal{D}))} + \frac{TN(\mathcal{D})}{TN(\mathcal{D}) + FP(\mathcal{D})}\right) && (27)\\
&= \frac{1}{2} \cdot \frac{TP(\mathcal{D})/\kappa + TN(\mathcal{D})}{TN(\mathcal{D}) + FP(\mathcal{D})} && (28)\\
&= \frac{TP(\mathcal{D})/\kappa + TN(\mathcal{D})}{TP(\mathcal{D})/\kappa + TN(\mathcal{D}) + FP(\mathcal{D}) + FN(\mathcal{D})/\kappa} && (29)\\
&= \frac{TP(\mathcal{D}') + TN(\mathcal{D}')}{TP(\mathcal{D}') + TN(\mathcal{D}') + FP(\mathcal{D}') + FN(\mathcal{D}')} && (30)\\
&= A(\mathcal{D}') && (31)
\end{aligned}$$

$\mathcal{D}'$ has size $n' = n\rho/\kappa + n(1-\rho) = \frac{2n}{1+\kappa}$, so

$$\begin{aligned}
J(\theta) &= -\frac{1}{n'} \sum_{i=1}^{n'} \left(y^{(i)} \log(h_\theta({x^{(i)}}')) + (1 - y^{(i)}) \log(1 - h_\theta({x^{(i)}}')) \right) && (32)\\
&= -\frac{1+\kappa}{2n} \sum_{i=1}^{n} w^{(i)} \left(y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right) && (33)
\end{aligned}$$

[3] otherwise we can round up to the nearest integer with introducing a slight approximation.

(d) [5 points] **Coding problem: re-weighting minority class**

In `src/imbalanced/imbalanced.py`, implement the logistic regression algorithm on $\mathcal{D}'$. Compute and report the classifier's accuracy (A), balanced accuracy ($\overline{A}$), accuracies for the two classes (A_0, A_1) on the validation dataset. You are expected to see that the accuracy of minority class (class 1) improved significantly whereas that of the majority class dropped compared to vanilla logistic regression. However, the balanced accuracy is significantly greater than that of the vanilla logistic regression.

Create a plot to visualize the validation 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 $y^{(i)} = 1$ than those with $y^{(i)} = 0$. 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.

Answer:

$A_1 = 0.910, A_0 = 0.898, \overline{A} = 0.904, A = 0.899.$

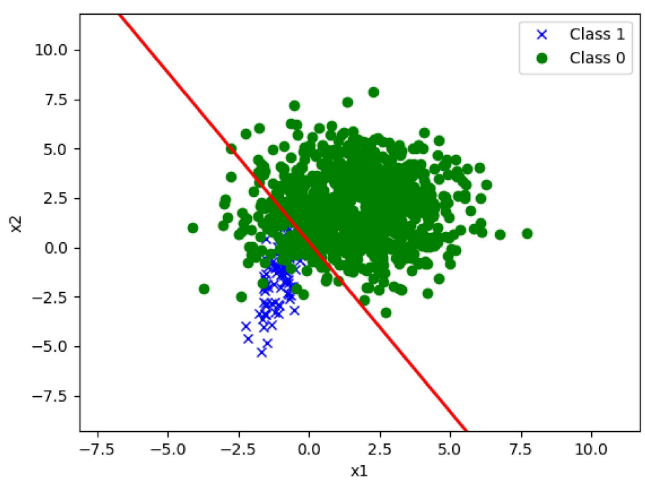

Figure 8: Decision boundary for upsampling algorithm