Ho chi minh university

university of science

falcuty of information technology

K23 - 23TNT1

Problem set 1

Course: CS229 - Machine Learning

Student's Name: Pham Phu Hoa

August 15, 2024

Contents

1 Logistic Regression

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, provided in the following files:

i `datads1_{train, valid}.csv`

ii `datads2_{train, valid}.csv`

Each file contains m examples, one example $(x^{(i)}), y^{(i)})$ per row. In paritcular, the i-th row contains columns $x_0^{(i)} \in \mathbf{R}, x_1^{(i)} \in \mathbf{R}$ and $y^{(i)} \in \{0, 1\}$. In the subproblems that follow, we will investigate using logistic regression and Gaussian discriminant ananalysis (GDA) to perform binary classifcation on these two dataset

(a) In lecture we saw log-likelihood function for logistic regression:

$$\ell(\theta) = \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

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

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

Recall the log likelihood function of Logistic Regression Model

$$\ell(\theta) = \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)} + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})$$

We take derivitate respect to matrix θ

$$\begin{aligned}
\nabla_\theta \ell(\theta) &= \nabla_\theta \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)} + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}) \\
&= \sum_{i=1}^{m} \nabla_\theta y^{(i)} \log h_\theta(x^{(i)} + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}) \\
&= \sum_{i=1}^{m} y^{(i)} \frac{1}{g(\theta^T x^{(i)})} g(\theta^T x^{(i)})(1 - g(\theta^T x^{(i)}))x^{(i)} + (1 - y^{(i)}) \frac{1}{1 - g(\theta^T x^{(i)})} (-g(\theta^T x^{(i)})(1 - g(\theta^T x^{(i)}))x \\
&= \sum_{i=1}^{m} (y^{(i)} - h_\theta(x^{(i)}))x^{(i)} \\
&= KX^T, \text{ where } K \text{ is matrix } y - h
\end{aligned}$$

After we found the gradient respect to θ. We want to find the Hessian matrix to apply Newton's method

$$\begin{aligned} H &= \nabla_{\theta^2}\ell(\theta) \\ &= \nabla_\theta \sum_{i=1}^{m} (y^{(i)} - h_\theta(x^{(i)}))x^{(i)} \\ &= \sum_{i=1}^{m} \nabla_\theta (y^{(i)} - h_\theta(x^{(i)}))x^{(i)} \\ &= -\sum_{i=1}^{m} h_\theta(x^{(i)})(1 - h_\theta(x^{(i)}))x^{(i)}{x^{(i)}}^T \end{aligned}$$

After we found Hessian matrix, update θ follow:

$$\theta = \theta - H^{-1}\nabla_\theta \ell(\theta)$$

(b) **Coding problem.** Follow the instruction in `src/p01b_logreg.py` to train a logistic regression classifier using Newton's Method. Starting with $\theta = 0$, run Newton's method until the updates to θ are small

```
 class LogisticRegression(LinearModel):
     """Logistic regression with Newton's Method as the solver.

     Example usage:
         > clf = LogisticRegression()
         > clf.fit(x_train, y_train)
         > clf.predict(x_eval)
     """

     def fit(self, x, y):
         """Run Newton's Method to maximize l(theta) for logistic
 regression.

         :param x: Training example inputs. Shape (m, n).
         :param y: Training example labels. Shape (m,).
         """

         def h(theta, x):
             """Vectorized implementation of h_theta(x) = 1 / (1 + exp(-
 theta^T x)).

             :param theta: Shape (n,).
             :param x:     All training examples of shape (m, n).
             :return:      The hypothesis for all training examples.
 Shape (m,).
             """
             return 1 / (1 + np.exp(-np.dot(x, theta)))

         def gradient(theta, x, y):
             """Vectorized implementation of the gradient of l(theta).

```

```
                :param theta: Shape (n,).
                :param x:     All training examples of shape (m, n).
                :param y:     All labels of shape (m,).
                :return:      The gradient of shape (n,).
                """
                m, _ = x.shape
                return np.dot(x.T, (y - h(theta, x)))

            def hessian(theta, x):
                """Vectorized implementation of the Hessian of J(theta).

                :param theta: Shape (n,).
                :param x:     All training examples of shape (m, n).
                :return:      The Hessian of shape (n, n).
                """
                m, _ = x.shape
                h_theta_x = np.reshape(h(theta, x), (-1, 1))
                return - np.dot(x.T, h_theta_x * (1 - h_theta_x) * x)

            def next_theta(theta, x, y):
                """The next theta updated by Newton's Method.

                :param theta: Shape (n,).
                :return:      The updated theta of shape (n,).
                """
                return theta - np.dot(np.linalg.inv(hessian(theta, x)),
    gradient(theta, x, y))

            m, n = x.shape

            # Initialize theta
            if self.theta is None:
                self.theta = np.zeros((n,))

            # Update theta using Newton's Method
            old_theta = self.theta
            new_theta = next_theta(self.theta, x, y)
            while np.linalg.norm(new_theta - old_theta, 1) >= self.eps:
                old_theta = new_theta
                new_theta = next_theta(old_theta, x, y)
            print('Train: ----------100%----------')
            self.theta = new_theta

        def predict(self, x):
            """Make a prediction given new inputs x.

            :param x: Inputs of shape (m, n).
            :return:  Outputs of shape (m,).
            """

            return x @ self.theta >= 0

```

Listing 1: Logistic Regression Implement (Newton's Method)

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

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

where $\phi, \mu_0, \mu_1, \Sigma$ are the parameters of our model

Supposr that we already fit, and now we 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

$$p(y=1|x) = \frac{1}{1+\exp(-(w^Tx+b)}$$

Find the parameter θ We know that if all the assumption is acceptable then the probability will equal to the sigmoid function

$$\begin{aligned} p(y=1 \mid x;\ \phi, \mu_0, \mu_1, \Sigma) &= \frac{p(x \mid y=1;\ \mu_0, \mu_1, \Sigma)\ p(y=1;\ \phi)}{p(x)} \\ &= 1/(1+\frac{p(x \mid y=0;\ \mu_0, \mu_1, \Sigma)\ p(y=0;\ \phi)}{p(x \mid y=1;\ \mu_0, \mu_1, \Sigma)\ p(y=1;\ \phi)}) \\ &= 1/(1+\exp\big(\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1) - \frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\big)\frac{1-\phi}{\phi}) \\ &= 1/(1+\exp\big(-\big((\mu_1-\mu_0)^T\Sigma^{-1}x + (\frac{1}{2}\mu_0^T\Sigma^{-1}\mu_0 - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 - \log\frac{1-\phi}{\phi})\big)\big)) \\ &= 1/(1+\exp\big(-(\theta^Tx+\theta_0)\big)) \end{aligned}$$

where $\theta = \Sigma^{-1}(\mu_1 - \mu_0)$ and $\theta_0 = \frac{1}{2}\mu_0^T\Sigma^{-1}\mu_0 - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 - \log\frac{1-\phi}{\phi}$.

(d) For this part if the problem only, you may assume n (the dimension of x) is 1, so that $\Sigma = [\sigma^2]$ is just a real number, and likewise the determinant of Σ is given by $|\Sigma| = \sigma^2$. Given tha dataset, we claim that the maximum likelihood estimates of the parameters are given by

$$\begin{aligned}
\phi &= \frac{1}{m}\sum_{i=1}^{m} 1\{y^{(i)}=1\} \\
\mu_0 &= \frac{\sum_{i=1}^{m} 1\{y^{(i)}=0\}x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)}=0\}} \\
\mu_1 &= \frac{\sum_{i=1}^{m} 1\{y^{(i)}=1\}x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)}=1\}} \\
\Sigma &= \frac{1}{m}\sum_{i=1}^{m}(x^{(i)}-\mu_{y^{(i)}})(x^{(i)}-\mu_{y^{(i)}})^T
\end{aligned}$$

Maximize log likelihood function To find ϕ, μ_0 and μ_1, recall the log-likelihood:

$$\begin{aligned}
\ell(\phi,\mu_0,\mu_1,\Sigma) &= \log\prod_{i=1}^{m} p(x^{(i)},y^{(i)}) \\
&= \log\prod_{i=1}^{m} p(x^{(i)}\mid y^{(i)})\; p(y^{(i)}) \\
&= \log\prod_{i=1}^{m} \big(p(x^{(i)}\mid y^{(i)}=1)\; p(y^{(i)}=1)\big)^{1\{y^{(i)}=1\}}\big(p(x^{(i)}\mid y^{(i)}=0)\; p(y^{(i)}=0)\big)^{1\{y^{(i)}=0\}}
\end{aligned}$$

First, we take deriviate of ℓ respect to ϕ

$$\begin{aligned}
\frac{\partial}{\partial\phi}\ell(\phi,\mu_0,\mu_1,\Sigma) &= \frac{\partial}{\partial\phi}\log\prod_{i=1}^{m} \big(p(x^{(i)}\mid y^{(i)}=1)\; p(y^{(i)}=1)\big)^{1\{y^{(i)}=1\}}\big(p(x^{(i)}\mid y^{(i)}=0)\; p(y^{(i)}=0)\big)^{1\{y^{(i)}=0\}} \\
&= \frac{\partial}{\partial\phi}\sum_{i=1}^{m} 1\{y^{(i)}=1\}\big(-\frac{1}{2}(x^{(i)}-\mu_1)^T\Sigma^{-1}(x^{(i)}-\mu_1)+\log\phi\big) \\
&\quad + \frac{\partial}{\partial\phi}\sum_{i=1}^{m} 1\{y^{(i)}=0\}\big(-\frac{1}{2}(x^{(i)}-\mu_0)^T\Sigma^{-1}(x^{(i)}-\mu_0)+\log(1-\phi)\big) + \frac{\partial}{\partial\phi}C \\
&= \sum_{i=1}^{m} 1\{y^{(i)}=1\}\frac{1}{\phi} - \sum_{i=1}^{m} 1\{y^{(i)}=0\}\frac{1}{1-\phi} \\
&= \frac{1}{\phi}\sum_{i=1}^{m} 1\{y^{(i)}=1\} - \frac{1}{1-\phi}\sum_{i=1}^{m} 1\{y^{(i)}=0\} \\
&= \frac{1}{\phi}\sum_{i=1}^{m} 1\{y^{(i)}=1\} - \frac{1}{1-\phi}(m-\sum_{i=1}^{m} 1\{y^{(i)}=1\})
\end{aligned}$$

Set the equation above to 0, we easily get

$$\phi = \frac{1}{m}\sum_{i=1}^{m} 1\{y^{(i)} = 1\}$$

Second, we compute gradient respect to μ_0

$$\begin{aligned}
\nabla_{\mu_0}\ell(\phi,\mu_0,\mu_1,\Sigma) &= \nabla_{\mu_0}\sum_{i=1}^{m} 1\{y^{(i)}=1\}\big(-\frac{1}{2}(x^{(i)}-\mu_1)^T\Sigma^{-1}(x^{(i)}-\mu_1)+\log\phi\big) \\
&\quad + \sum_{i=1}^{m} 1\{y^{(i)}=0\}\big(-\frac{1}{2}(x^{(i)}-\mu_0)^T\Sigma^{-1}(x^{(i)}-\mu_0)+\log(1-\phi)\big) + C \\
&= \nabla_{\mu_0}\sum_{i=1}^{m} 1\{y^{(i)}=0\}\big(-\frac{1}{2}(x^{(i)}-\mu_0)^T\Sigma^{-1}(x^{(i)}-\mu_0)+\log(1-\phi)\big) \\
&= \sum_{i=1}^{m} 1\{y^{(i)}=0\}\nabla_{\mu_0}\big(-\frac{1}{2}(x^{(i)}-\mu_0)^T\Sigma^{-1}(x^{(i)}-\mu_0)+\log(1-\phi)\big) \\
&= \sum_{i=1}^{m} 1\{y^{(i)}=0\}\nabla_{\mu_0}\big(\mu_0^T\Sigma^{-1}x^{(i)}-\frac{1}{2}\mu_0^T\Sigma^{-1}\mu_0\big) \\
&= \sum_{i=1}^{m} 1\{y^{(i)}=0\}\Sigma^{-1}x^{(i)}-\frac{1}{2}(\mu_0^T\Sigma^{-1}+\mu_0^T\Sigma^{-1})^T \\
&= \sum_{i=1}^{m} 1\{y^{(i)}=0\}\Sigma^{-1}(x^{(i)}-\mu_0)
\end{aligned}$$

Set the equation above equal to 0, we easily get

$$\mu_0 = \frac{\sum_{i=1}^{m} 1\{y^{(i)}=0\}x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)}=0\}}$$

Similarly for μ_1

Lastly, we compute the co-variance matrix Σ of the distribution
Simplify ℓ w.r.t Σ^{-1}:

$$\begin{aligned}
\ell(\phi,\mu_0,\mu_1,\Sigma) &= -\frac{m}{2}\log|\Sigma| - \frac{1}{2}\sum_{i=1}^{m}(x^{(i)}-\mu_{y^{(i)}})^T\Sigma^{-1}(x^{(i)}-\mu_{y^{(i)}}) + C \\
&= \frac{m}{2}\log|\Sigma^{-1}| - \frac{1}{2}\sum_{i=1}^{m}\Sigma^{-1}(x^{(i)}-\mu_{y^{(i)}})(x^{(i)}-\mu_{y^{(i)}})^T + C
\end{aligned}$$

We can derive the same estimate by solving:

$$\frac{\partial}{\partial \Sigma^{-1}} \ell(\phi, \mu_0, \mu_1, \Sigma) = \frac{m}{2}\Sigma - \frac{1}{2}\sum_{i=1}^{m}(x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$$

For the last time, set equattion above to 0, and we find the co-variance matrix that maximize the log-likelihood function

$$\begin{aligned}\Sigma &= \frac{1}{m}\sum i = 1^m (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T \\ &= \frac{1}{m}RR^T\end{aligned}$$

, where R is the row matrix about distance of each input with its mean

(e) **Coding Problem**. In `src/p01e_gda.py`, fill in the code to calculate these parameters to derive θ and use the resulting GDA model to make prediction on the validation set.

```
    class GDA(LinearModel):
        """Gaussian Discriminant Analysis.

        Example usage:
            > clf = GDA()
            > clf.fit(x_train, y_train)
            > clf.predict(x_eval)
        """

        def fit(self, x, y):
            """Fit a GDA model to training set given by x and y.

            :param x: Training example inputs. Shape (m, n).
            :param y: Training example labels. Shape (m,).
            """

            m, n = x.shape

            phi = np.sum(y) / m
            mu_0 = np.dot(x.T, 1 - y) / np.sum(1 - y)
            mu_1 = np.dot(x.T, y) / np.sum(y)

            # Reshape y to compute pairwise product with mu
            y_reshaped = np.reshape(y, (m, -1))

            # Matrix comprises mu_0 and mu_1 based on the value of y. Shape
   (m, n)
            mu_x = y_reshaped * mu_1 + (1 - y_reshaped) * mu_0

            x_centered = x - mu_x

            sigma = np.dot(x_centered.T, x_centered) / m
            sigma_inv = np.linalg.inv(sigma)

            # Compute theta and theta_0 according to the conclusion from part (c)
            theta = np.dot(sigma_inv, mu_1 - mu_0)
            theta_0 = 1 / 2 * mu_0 @ sigma_inv @ mu_0 - 1 / 2 * mu_1 @ sigma_inv @ mu_1 - np.log((1 - phi) / phi)

            self.theta = np.insert(theta, 0, theta_0)

        def predict(self, x):
            """Make a prediction given new inputs x.

            :param x: Inputs of shape (m, n).
            :return:  Outputs of shape (m,).
            """

            # Add x_0 = 1 convention to make predictions using theta^T x >= 0
            return util.add_intercept(x) @ self.theta >= 0

```

Listing 2: GDA Implemenation

2 Incomplete Positive-Only Labels

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

That is, we assume a dataset{ $(x^{(i)}, y^{(i)}, t^{(i)})_{i=1}^{m}$}, where $t^{(i)} \in \{0, 1\}$ is the "true" label, and where

$$y^{(i)} = \begin{cases} 1 & x^{(i)} \text{ is labeled} \\ 0 & \text{otherwise} \end{cases}$$

All labeles examples are positive, which is to say $p(t^{(i)} = 1 \mid y^{(i)} = 1) = 1$, but unlabeled examples may be positive or negative . Our goal in the problem is to conctruct a binary classifier h such that $h(x^{(i)}) \approx p(t^{(i)} = 1 \mid x^{(i)})$ as closely as possible using only x and y.

(a) Suppose that each $y^{(i)}$ and $x^{(i)}$ are conditionally independent given $t^{(i)}$

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

Prove that

$$p(t^{(i)} = 1 \mid x^{(i)}) = p(y^{(i)} = 1 \mid x^{(i)})/\alpha, \ \alpha \in \mathbf{R}$$

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

(b) Suppose we want to estimate α using a training classifier h and held-out validation set V, Let V_+ be the set of labeled (and hence positive) examples in V. Assuming that $h(x^{(i)}) \approx p(y^{(i)} = 1 \mid x^{(i)})$ for all examples $x^{(i)}$, show that

$$h(x^{(i)}) \approx \alpha, \text{ for all } x^{(i)} \in V_+$$

$$\begin{aligned} \langle(x^{(i)}) &\approx p(y^{(i)} = 1 \mid x^{(i)}) \\ &\approx \alpha \ p(t^{(i)} = 1 \mid x^{(i)}) \\ &\approx \alpha \end{aligned}$$

(c) **Coding problem.** The following three problem will deal with a dataset which we have provided i the following files

`data3_{train, valid, test}.cvs`

First we will consider the ideal case, where we have access to the true t-labels for training. In `src02_posonly.py`, write a logistic regression classidier that uses x_1, x_2 as input features, and train it uisng the $t - labels$ (you can ignore the $y - labels$ for this part). Out put the trained model's predictions on the test set to the file specified in the code

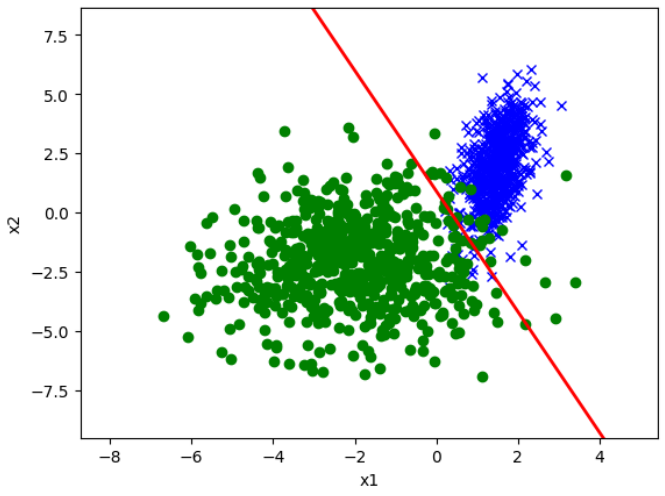


Figure 1: Output boundary using true label

(d) **Coding problem.** We now consider the case that $t - labels$ are unavailable, so you only hace acess to the $y - labels$ at training time. Add to your code in `p02cde_posonly.py` to re-train the classifier, but using the $y - labels$ only

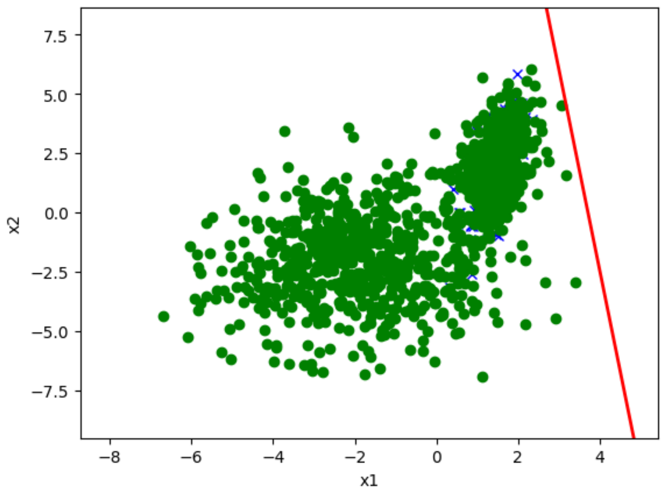


Figure 2: Output using y label

(e) **Coding problem.** Using the validation set, estimate the constant $\alpha$by averaging your classifier's predictions over all labeled examples in the validation set:

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

To plot the decision boundary found by solving $\frac{1}{\alpha}\frac{1}{1+\exp(-\theta^T x)} = \frac{1}{2}$, we can equivalently transform the equation to the form of $\theta'^T x = 0$ and solve for θ'. By simplifying the equation, we obtain $\theta^T x - \log(\frac{2}{\alpha} - 1) = 0$. The left-hand side is in fact adding $\log(\frac{2}{\alpha} - 1)$ to θ_0.

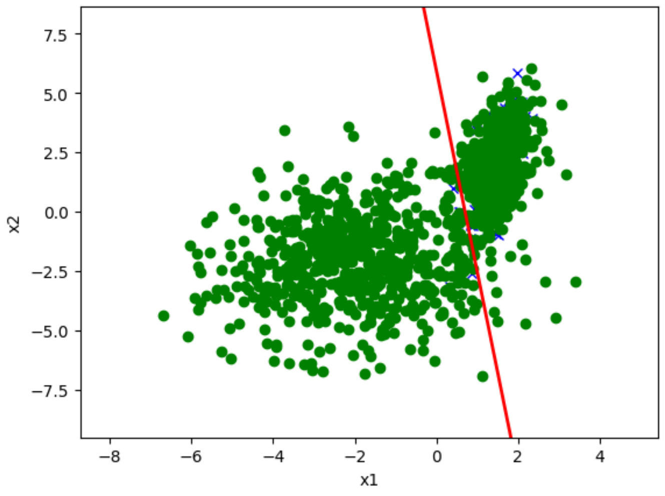


Figure 3: Caption

3 Poisson Regression

(a) Consider the Poisson distribution parameterized by λ

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

Show that the Poisson distribution is in the exponential family, and clearly the values for $b(y), \eta, T(y), a(\eta)$.

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

So...

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

(b) Consider performing regression using a GLM model with a Poisson response variable. What is the canonical response function for the family?

$$\begin{aligned} h_\theta(x) &= E[y \mid x;\theta] \\ &= \lambda \\ &= e^{\eta} \\ &= e^{\theta^T x} \end{aligned}$$

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

$$\begin{aligned}
\ell(\theta) &= \log p(y^{(i)} \mid x^{(i)}; \theta) \\
&= \log \frac{e^{-\lambda}\ \lambda^y}{y!} \\
&= -e^{\eta} + \eta y^{(i)} - \log y! \\
&= -e^{\theta^T x^{(i)}} + \theta^T x^{(i)} y^{(i)} - \log y!
\end{aligned}$$

Let's take derivitate respect to θ_j

$$\begin{aligned}
\frac{\partial}{\partial \theta_j}\ell(\theta) &= \frac{\partial}{\partial \theta_j} - e^{\theta^T x^{(i)}} + \theta^T x^{(i)} y^{(i)} - \log y! \\
&= -x_j^{(i)} e^{\theta^T x} + x_j^{(i)} \\
&= (y^{(i)} - e^{\theta^T x^{(i)}}) x_j^{(i)}
\end{aligned}$$

Learning update rule:

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

(d) **Coding problem.** Consider a website that wants to predict its daily traffic. the website owners have collected a dataset of past traffic to their webiste, along with some features which they think are useful in predicting the number of visitor per dat. The dataset is split into train/valid/test sets

`data/ds4_{train, valid}.csv`

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. In `src/p03d_poisson.py`, implement Poisson regression for this dataset and use gradient ascent to maximize the log-likelihood of θ

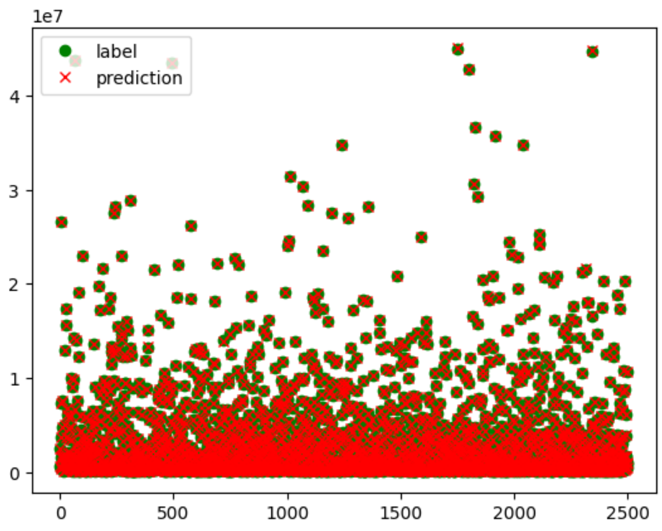


Figure 4: Caption

4 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.
To recap, an exponential family distribution is one whose probability density can be represented

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

(a) Derive an expression for the mean of the distribution. Show that $\mathbb{E}[Y\ |X;\theta]$ can be represented as the gradient of the lofg-partition function a with respect to the natural parameter η

$$\begin{aligned}\frac{\partial}{\partial\eta}p(y;\ \eta) &= \frac{\partial}{\partial\eta}\big(b(y)\exp(\eta y - a(\eta))\big)\\ &= b(y)\exp(\eta y - a(\eta))(y - \frac{\partial}{\partial\eta}a(\eta))\\ &= y\ p(y;\ \eta) - p(y;\ \eta)\frac{\partial}{\partial\eta}a(\eta)\end{aligned}$$

From the equation above, we have:

$$y\ p(y;\ \eta) = \frac{\partial}{\partial\eta}p(y;\ \eta) + p(y;\ \eta)\frac{\partial}{\partial\eta}a(\eta)$$

$$\begin{aligned}
\mathbb{E}[\,Y;\ \eta\,] &= \mathbb{E}[\,Y \mid X;\ \eta\,] \\
&= \int y\ p(y;\ \eta)\ dy \\
&= \int \frac{\partial}{\partial\eta} p(y;\ \eta) + p(y;\ \eta)\frac{\partial}{\partial\eta}a(\eta)\ dy \\
&= \int \frac{\partial}{\partial\eta} p(y;\ \eta)\ dy + \int p(y;\ \eta)\frac{\partial}{\partial\eta}a(\eta)\ dy \\
&= \frac{\partial}{\partial\eta}\int p(y;\ \eta)\ dy + \frac{\partial}{\partial\eta}a(\eta)\int p(y;\ \eta)\ dy \\
&= \frac{\partial}{\partial\eta}\cdot 1 + \frac{\partial}{\partial\eta}a(\eta)\cdot 1 \\
&= 0 + \frac{\partial}{\partial\eta}a(\eta) \\
&= \frac{\partial}{\partial\eta}a(\eta)
\end{aligned}$$

(b) Next, derive an expression for the variance of the distribution. In particular show that $Var(Y \mid X;\theta)$ can b expressed as the derivative of the mean w.r.t η

$$\begin{aligned}
\frac{\partial^2}{\partial\eta^2}p(y;\ \eta) &= \frac{\partial}{\partial\eta}\big(y\ p(y;\ \eta) - p(y;\ \eta)\frac{\partial}{\partial\eta}a(\eta)\big) \\
&= y^2 p(y;\ \eta) - 2\ y\ p(y;\ \eta)\frac{\partial}{\partial\eta}a(\eta) + p(y;\ \eta)\big(\frac{\partial}{\partial\eta}a(\eta)\big)^2 - p(y;\ \eta)\frac{\partial^2}{\partial\eta^2}a(\eta) \\
&= -p(y;\ \eta)\frac{\partial^2}{\partial\eta^2}a(\eta) + \big(y - \frac{\partial}{\partial\eta}a(\eta)\big)^2 p(y;\ \eta)
\end{aligned}$$

So, we have:

$$\big(y - \frac{\partial}{\partial\eta}a(\eta)\big)^2 p(y;\ \eta) = \frac{\partial^2}{\partial\eta^2}p(y;\ \eta) + p(y;\ \eta)\frac{\partial^2}{\partial\eta^2}a(\eta)$$

Hence,

$$\begin{aligned}
\mathrm{Var}[\ Y;\ \eta\ ] &= \int (y - \mathbb{E}[\ Y;\ \eta\ ])^2 p(y;\ \eta)\ dy \\
&= \int \big(y - \frac{\partial}{\partial \eta} a(\eta)\big)^2 p(y;\ \eta)\ dy \\
&= \int \frac{\partial^2}{\partial \eta^2} p(y;\ \eta) + p(y;\ \eta) \frac{\partial^2}{\partial \eta^2} a(\eta)\ dy \\
&= \int \frac{\partial^2}{\partial \eta^2} p(y;\ \eta)\ dy + \int p(y;\ \eta) \frac{\partial^2}{\partial \eta^2} a(\eta)\ dy \\
&= \frac{\partial^2}{\partial \eta^2} \int p(y;\ \eta)\ dy + \frac{\partial^2}{\partial \eta^2} a(\eta) \int p(y;\ \eta)\ dy \\
&= \frac{\partial^2}{\partial \eta^2} a(\eta)
\end{aligned}$$

5 Locally weighted linear regression

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

$$\mathcal{J}(\theta) = \frac{1}{2}\sum_{i=1}^{m} w^{(i)}\Big(\theta^T x^{(i)} - y^{(i)}\Big)$$

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 ell generalize some of those ideas to the weighted setting.

i Show that $\mathcal{J}(\theta)$ can be also written

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

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

$$\begin{aligned}\mathcal{J}(\theta) &= \frac{1}{2}\sum_{i=1}^{m} w^{(i)}\Big(\theta^T x^{(i)} - y^{(i)}\Big)\\ &= \frac{1}{2}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})w^{(i)}(\theta^T x^{(i)} - y^{(i)})\\ &= \frac{1}{2}(X\theta - y)^T W (X\theta - y)\end{aligned}$$

ii If all the $w^{(i)}$ equal 1, then we saw in class that normal equation is

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

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

To find closed from of parameters that minimizes $\mathcal{J}(\theta)$, we compute gradient of $\mathcal{J}(\theta)$ respect to θ

$$\begin{aligned}\nabla_\theta \mathcal{J}(\theta) &= \nabla_\theta \frac{1}{2}(x\theta - y)^T W (X\theta - y)\\ &= \nabla_\theta (X\theta - y)\Big((X\theta - y)W + (X\theta - y)W^T\Big)^T\\ &= \frac{1}{2}X^T(W + W)(X\theta - y)\\ &= X^T W (X\theta - y) = 0 \qquad \Rightarrow \theta = (X^T W X)^{-1} X^T W y\end{aligned}$$

iii Suppose we have a dataset $\{(x^{(i)}, y^{(i)}\}_{i=1}^{m}$ of m independent examples, but we model the $y^{(i)}$'s as a drawn from conditional distributions with diffirent levels of variance $(\sigma^{(i)})^2$. Specifically, assume the model

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

That is each $y^{(i)}$ is drawn from a Gaussian distribution with mean $\theta^T x^{(i)}$ and variance $(\sigma^{(i)})^2$. 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 $\sigma^{(i)\prime}s$

$$\begin{aligned}
\ell(\theta) &= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi}\sigma^{(i)}} \exp\big(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2(\sigma^{(i)})^2} \big) \\
&= -m \log \sqrt{2\pi}\sigma^{(i)} - \sum_{i=1}^{m} \frac{(y^{(i)} - \theta^T x^{(i)})^2}{2(\sigma^{(i)})^2} \\
&= -\frac{1}{2} \sum_{i=1}^{m} \frac{1}{(\sigma^{(i)})^2} (\theta^T x^{(i)} - y^{(i)})^2 - m \log \sqrt{2\pi}\sigma^{(i)}
\end{aligned}$$

Thus, maximizing $\ell(\theta)$ is equivalent to minimizing

$$\frac{1}{2} \sum_{i=1}^{m} \frac{1}{(\sigma^{(i)})^2} (\theta^T x^{(i)} - y^{(i)})^2$$

By setting $w^{(i)} = 1/(\sigma^{(i)})^2$, finding the maximum likelihood estimate of θ reduces to minimizing $J(\theta)$.

(b) **Coding problem.** We will now consider the following dataset (the formatting matches that of Dataset 1-4, except $x^{(i)}$ is 1-dimensional)

`data/ds5_{train, valid, test}.csv`

In `src/p05b_lwr.py` implement locally weighted linear regression using the normal equations you derived in Part (a) and using

$$w^{(i)} = \exp\Bigg(-\frac{\|x^{(i)} - x\|^2}{2\tau^2} \Bigg)$$

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

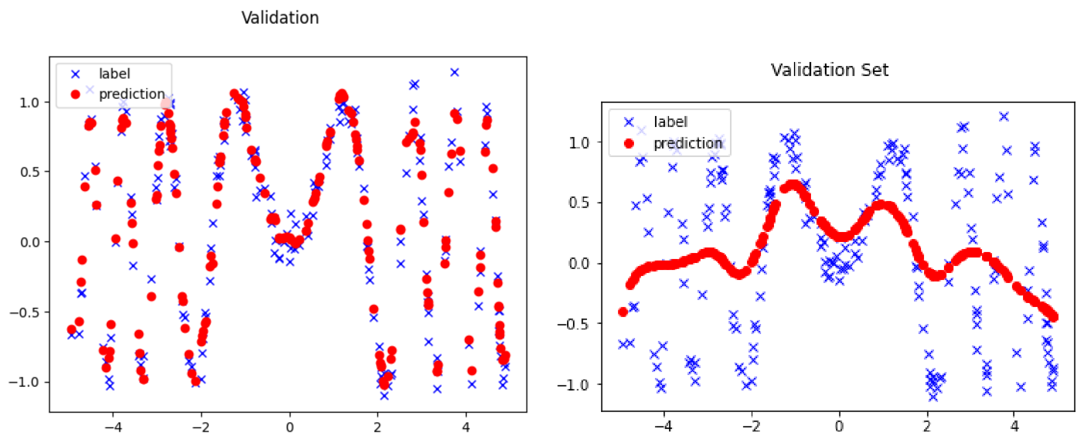
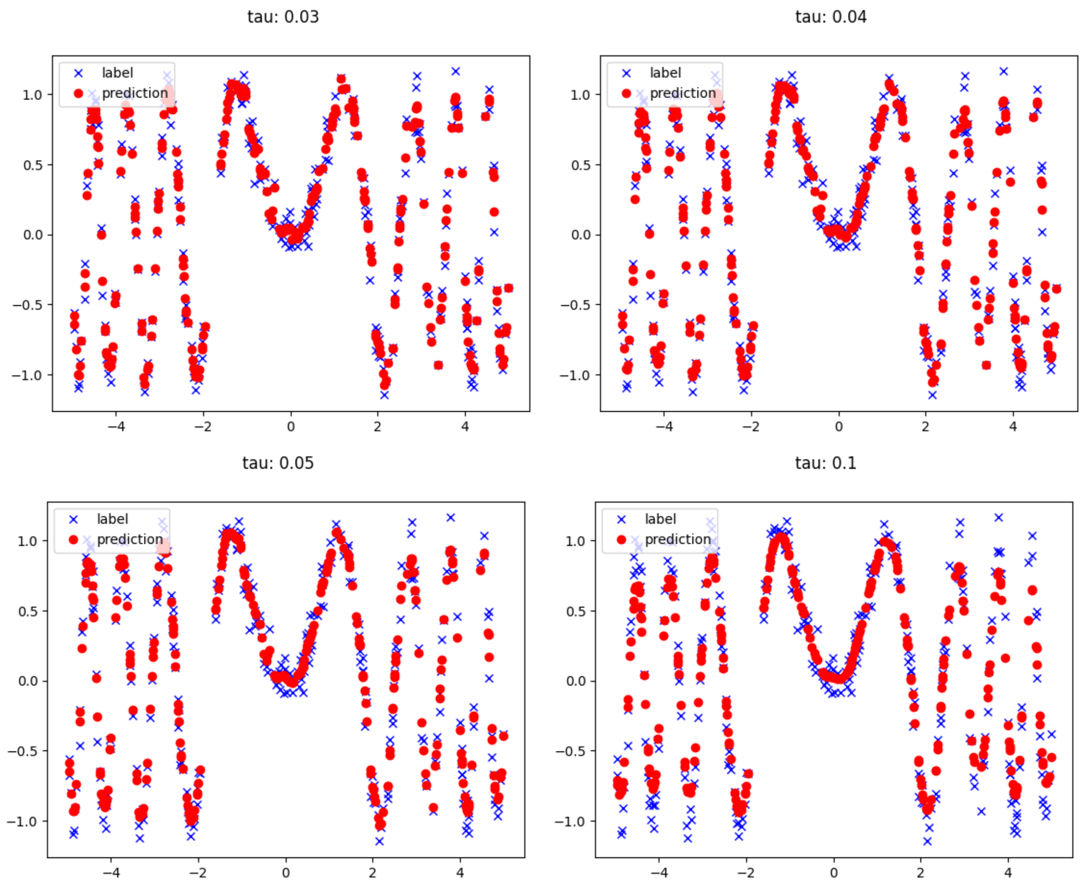


Figure 5: Tau: 0.5

(c) **Coding problem.** We will now tune the hyperparameter τ. For each τ , plot your model's predictions on the validation set in the format described in part (b). Report the value of which achieves the lowest MSE on the valid split, and finally report the MSE on the test split using this τ-value



6 Github Repository