Homework #1

CSE 446/546: Machine Learning
Sami Turbeville
Collab: Shabab Ahmed
Due: **Wednesday** October 20, 2021 11:59pm
A: 59 points, **B:** 30 points

Please review all homework guidance posted on the website before submitting to GradeScope. Reminders:

- Make sure to read the "What to Submit" section following each question and include all items.
- Please provide succinct answers and supporting reasoning for each question. Similarly, when discussing experimental results, concisely create tables and/or figures when appropriate to organize the experimental results. All explanations, tables, and figures for any particular part of a question must be grouped together.
- For every problem involving generating plots, please include the plots as part of your PDF submission.
- When submitting to Gradescope, please link each question from the homework in Gradescope to the location of its answer in your homework PDF. Failure to do so may result in deductions of up to *[5 points]* . For instructions, see `https://www.gradescope.com/get_started#student-submission`.
- Please recall that B problems, indicated in boxed text, are only graded for 546 students, and that they will be weighted at most 0.2 of your final GPA (see website for details). In Gradescope there is a place to submit solutions to A and B problems separately. You are welcome to create just a single PDF that contains answers to both, submit the same PDF twice, but associate the answers with the individual questions in Gradescope.
- If you collaborate on this homework with others, you must indicate who you worked with on your homework. Failure to do so may result in accusations of plagiarism.
- For every problem involving code, please include the code as part of your PDF for the PDF submission *in addition to* submitting your code to the separate assignment on Gradescope created for code. Not submitting all code files will lead to a deduction of *[1 point]* .

Not adhering to these reminders may result in point deductions.

Short Answer and "True or False" Conceptual questions

A1. The answers to these questions should be answerable without referring to external materials. Briefly justify your answers with a few words.

---

a. *[2 points]* In your own words, describe what bias and variance are? What is bias-variance tradeoff?

   text

---

b. *[2 points]* What **typically** happens to bias and variance when the model complexity increases/decreases?

As complexity increases bias decreases and variance increases, and vice versa. There is a sweet spot somewhere in between where bias is considerably low and variance is also minimized. There will be some unusual cases where this may not be the

---

c. *[1 point]* True or False: A learning algorithm will always generalize better if we use fewer features to represent our data.

False, with fewer training data points the model will not generalize as well because it doesn't have as many features to detect and it will be biased towards this training data which may not be the best representation of the complete data.

---

d. *[2 points]* True or False: Hyperparameters should be tuned on the test set. Explain your choice and detail a procedure for hyperparameter tuning.

False, never ever ever use the test set. I would use the k-folding cross validation because it is faster even though it is more biased.

---

e. *[1 point]* True or False: The training error of a function on the training set provides an overestimate of the true error of that function.

Maximum Likelihood Estimation (MLE)

A2. You're the Reign FC manager, and the team is five games into its 2021 season. The number of goals scored by the team in each game so far are given below:

$$[2, 4, 6, 0, 1].$$

Let's call these scores $x_1, \cdots, x_5$. Based on your (assumed iid) data, you'd like to build a model to understand how many goals the Reign are likely to score in their next game. You decide to model the number of goals scored per game using a *Poisson distribution*. Recall that the Poisson distribution with parameter λ assigns every non-negative integer $x = 0, 1, 2, \cdots$ a probability given by

$$\mathrm{Poi}(x|\lambda) = e^{-\lambda}\frac{\lambda^x}{x!}.$$

a. *[5 points]* Derive an expression for the maximum-likelihood estimate of the parameter λ governing the Poisson distribution in terms of goal counts for the first n games: $x_1, \cdots, x_n$. (Hint: remember that the log of the likelihood has the same maximizer as the likelihood function itself.)

b. *[2 points]* Give a numerical estimate of λ after the first five games. Given this λ, what is the probability that the Reign score 6 goals in their next game?

c. *[2 points]* Suppose the Reign score 8 goals in their 6th game. Give an updated numerical estimate of λ after six games and compute the probability that the Reign score 6 goals in their 7th game.

Solutions

a. Let's start by finding the probability of getting this combination of scores $x_1, ..., x_5$

$$\begin{aligned}
P(x_1 \cdots x_n) &= P(x_1)P(x_2)\cdots P(x_n) && \text{since scores are iid} \\
&= \prod e^{-\lambda}\frac{\lambda^x}{x!} && \text{by definition} \\
ln(P(x_1 \cdots x_n)) &= ln(\prod e^{-\lambda}\frac{\lambda^{x_i}}{x_i!}) && \text{taking natural log of both sides} \\
&= \sum ln(e^{\lambda}\frac{\lambda^{x_i}}{x_i!}) && \text{since log turns product to sum} \\
&= \sum -\lambda + x_i ln(\lambda) - ln(x_i!) \\
&= -n\lambda + ln(\lambda)\sum(x_i) - \sum(ln(x_i!))
\end{aligned}$$

Then we differentiate to get the maximum.

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}\lambda} ln(P(x_1 \cdots x_n)) &= \frac{\mathrm{d}}{\mathrm{d}\lambda}[-n\lambda + ln(\lambda)\sum(x_i) - \sum(ln(x_i!))] \\
&= -n + \frac{1}{\lambda}\sum(x_i) - 0 \\
0 &= -n + \frac{1}{\lambda}\sum(x_i) && \text{maximized when differential is 0} \\
n &= \frac{1}{\lambda}\sum(x_i) && \text{solve for } \lambda \\
\lambda &= \frac{1}{n}\sum(x_i)
\end{aligned}$$

Thus,

$$\boxed{\lambda = \mathrm{mean}(x_i, ..., x_n)}$$

b. After the first five games,

$$\lambda = \frac{1}{5}(2+4+6+0+1)$$

$$\boxed{\lambda = \frac{13}{5} = 2.6}$$

The probability that Reign score 6 goals in their next game is

$$Poi(x|\lambda) = e^{-\lambda}\frac{\lambda^x}{x!}$$
$$Poi(6|2.6) = e^{-2.6}\frac{2.6^6}{6!}$$

$$\boxed{Poi(6|2.6) = 0.0319}$$

c. After the sixth game if Reign scores 8 goals, then

$$\lambda = \frac{1}{6}(2+4+6+0+1+8)$$

$$\boxed{\lambda = \frac{21}{6} = 3.5}$$

The probability that Reign score 6 goals in their next game is

$$Poi(x|\lambda) = e^{-\lambda}\frac{\lambda^x}{x!}$$
$$Poi(6|3.5) = e^{-3.5}\frac{3.5^6}{6!}$$

$$\boxed{Poi(6|3.5) = 0.0770983}$$

A3. *[10 points]* *(Optional Background)* In World War 2, the Allies attempted to estimate the total number of tanks the Germans had manufactured by looking at the serial numbers of the German tanks they had destroyed. The idea was that if there were n total tanks with serial numbers $\{1, \cdots, n\}$ then its reasonable to expect the observed serial numbers of the destroyed tanks constituted a uniform random sample (without replacement) from this set. The exact maximum likelihood estimator for this so-called *German tank problem* is non-trivial and quite challenging to work out (try it!). For our homework, we will consider a much easier problem with a similar flavor.

Let $x_1, \cdots, x_n$ be independent, uniformly distributed on the continuous domain $[0, \theta]$ for some θ. What is the Maximum likelihood estimate for θ?

Solution

- The pdf for the iid on the continuous domain from 0 to θ is

$$f(x_1 \cdots x_n) = \frac{1}{\theta}$$

for $x_i \in [0, \theta]$. Because they are iid, we can do the following

$$f(x_1 \cdots x_n) = f(x_1)f(x_2) \cdots f(x_n) = \frac{1}{\theta^n}$$

The maximum θ is found by taking the derivative which we be the same as the derivative of the log.

$$ln((\frac{1}{\theta})^n) = -nln(\theta)$$

$$\frac{\mathrm{d}}{\mathrm{d}\theta} - nln(\theta) = -\frac{n}{\theta}$$

Since n and θ must be non-negative, then

$$-\frac{n}{\theta} \leq 0$$

So θ is maximized for the largest x_i. So,

$$\boxed{\text{MLE}(\theta) = \max(x_1, ..., x_n)}$$

Polynomial Regression

Relevant Files[1]

- **polyreg.py**
- `linreg_closedform.py`
- `test_polyreg_univariate.py`
- `test_polyreg_learningCurve.py`
- `data/polydata.dat`

A4. *[10 points]* Recall that polynomial regression learns a function $h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_d x^d$, where d represents the polynomial's highest degree. We can equivalently write this in the form of a linear model with d features

$$h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 \phi_1(x) + \theta_2 \phi_2(x) + \ldots + \theta_d \phi_d(x) \ , \tag{1}$$

using the basis expansion that $\phi_j(x) = x^j$. Notice that, with this basis expansion, we obtain a linear model where the features are various powers of the single univariate x. We're still solving a linear regression problem, but are fitting a polynomial function of the input.

Implement regularized polynomial regression in `polyreg.py`. You may implement it however you like, using gradient descent or a closed-form solution. However, I would recommend the closed-form solution since the data sets are small; for this reason, we've included an example closed-form implementation of linear regression in `linreg_closedform.py` (you are welcome to build upon this implementation, but make CERTAIN you understand it, since you'll need to change several lines of it). You are also welcome to build upon your implementation from the previous assignment, but you must follow the API below. Note that all matrices are actually 2D numpy arrays in the implementation.

- `__init__(degree=1, regLambda=1E-8)` : constructor with arguments of d and λ
- `fit(X,Y)`: method to train the polynomial regression model
- `predict(X)`: method to use the trained polynomial regression model for prediction
- `polyfeatures(X, degree)`: expands the given $n \times 1$ matrix X into an $n \times d$ matrix of polynomial features of degree d. Note that the returned matrix will not include the zero-th power.

Note that the `polyfeatures(X, degree)` function maps the original univariate data into its higher order powers. Specifically, X will be an $n \times 1$ matrix ($X \in \mathbb{R}^{n \times 1}$) and this function will return the polynomial expansion of this data, a $n \times d$ matrix. Note that this function will **not** add in the zero-th order feature (i.e., $x_0 = 1$). You should add the x_0 feature separately, outside of this function, before training the model.

By not including the x_0 column in the matrix `polyfeatures()`, this allows the `polyfeatures` function to be more general, so it could be applied to multi-variate data as well. (If it did add the x_0 feature, we'd end up with multiple columns of 1's for multivariate data.)

Also, notice that the resulting features will be badly scaled if we use them in raw form. For example, with a polynomial of degree $d = 8$ and $x = 20$, the basis expansion yields $x^1 = 20$ while $x^8 = 2.56 \times 10^{10}$ – an absolutely huge difference in range. Consequently, we will need to standardize the data before solving linear regression. Standardize the data in `fit()` after you perform the polynomial feature expansion. You'll need to apply the same standardization transformation in `predict()` before you apply it to new data.

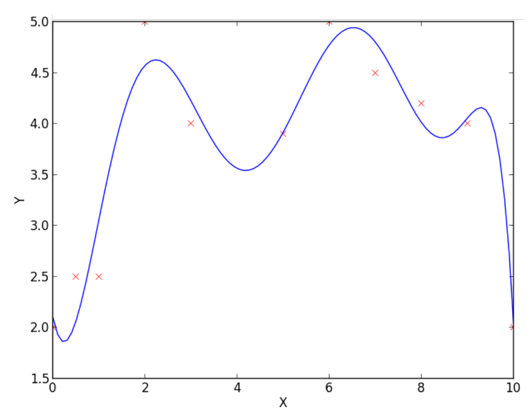

Figure 1: Fit of polynomial regression with $\lambda = 0$ and $d = 8$

Run `test_polyreg_univariate.py` to test your implementation, which will plot the learned function. In this case, the script fits a polynomial of degree $d = 8$ with no regularization $\lambda = 0$. From the plot, we see that the function fits the data well, but will not generalize well to new data points. Try increasing the amount of regularization, and in 1-2 sentences, describe the resulting effect on the function (you may also provide an additional plot to support your analysis).

[1]**Bold text** indicates files or functions that you will need to complete; you should not need to modify any of the other files.

Ridge Regression on MNIST

A5. In this problem we will implement a regularized least squares classifier for the MNIST data set. The task is to classify handwritten images of numbers between 0 to 9.

You are **NOT** allowed to use any of the pre-built classifiers in `sklearn`. Feel free to use any method from `numpy` or `scipy`. **Remember:** if you are inverting a matrix in your code, you are probably doing something wrong (Hint: look at `scipy.linalg.solve`).

Each example has features $x_i \in \mathbb{R}^d$ (with $d = 28 * 28 = 784$) and label $z_j \in \{0, ..., 9\}$. You can visualize a single example x_i with `imshow` after reshaping it to its original 28×28 image shape (and noting that the label z_j is accurate). We wish to learn a predictor $\widehat{f}$ that takes as input a vector in $\mathbb{R}^d$ and outputs an index in $\{0, \cdots, 9\}$. We define our training and testing classification error on a predictor f as

$$\widehat{\epsilon}_{\text{train}}(f) = \frac{1}{N_{\text{train}}} \sum_{(x,z)\in\text{Training Set}} \mathbf{1}\{f(x) \neq z\}$$
$$\widehat{\epsilon}_{\text{test}}(f) = \frac{1}{N_{\text{test}}} \sum_{(x,z)\in\text{Test Set}} \mathbf{1}\{f(x) \neq z\}$$

We will use one-hot encoding of the labels: for each observation (x, z), the original label $z \in \{0, \ldots, 9\}$ is mapped to the standard basis vector e_{z+1} where e_i is a vector of size k containing all zeros except for a 1 in the i^{th} position (positions in these vectors are indexed starting at one, hence the $z + 1$ offset for the digit labels). We adopt the notation where we have n data points in our training objective with features $x_i \in \mathbb{R}^d$ and label one-hot encoded as $y_i \in \{0, 1\}^k$. Here, $k = 10$ since there are 10 digits.

a. *[10 points]* In this problem we will choose a linear classifier to minimize the regularized least squares objective:

$$\widehat{W} = \text{argmin}_{W\in\mathbb{R}^{d\times k}} \sum_{i=1}^{n} \|W^T x_i - y_i\|_2^2 + \lambda\|W\|_F^2$$

Note that $\|W\|_F$ corresponds to the Frobenius norm of W, i.e. $\|W\|_F^2 = \sum_{i=1}^{d} \sum_{j=1}^{k} W_{i,j}^2$. To classify a point x_i we will use the rule $\arg\max_{j=0,\cdots,9} e_{j+1}^T \widehat{W}^T x_i$. Note that if $W = \begin{bmatrix} w_1 & \cdots & w_k \end{bmatrix}$ then

$$\begin{aligned}
\sum_{i=1}^{n} \|W^T x_i - y_i\|_2^2 + \lambda\|W\|_F^2 &= \sum_{j=1}^{k} \left[\sum_{i=1}^{n} (e_j^T W^T x_i - e_j^T y_i)^2 + \lambda\|W e_j\|^2 \right] \\
&= \sum_{j=1}^{k} \left[\sum_{i=1}^{n} (w_j^T x_i - e_j^T y_i)^2 + \lambda\|w_j\|^2 \right] \\
&= \sum_{j=1}^{k} \left[\|X w_j - Y e_j\|^2 + \lambda\|w_j\|^2 \right]
\end{aligned}$$

where $X = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix}^\top \in \mathbb{R}^{n\times d}$ and $Y = \begin{bmatrix} y_1 & \cdots & y_n \end{bmatrix}^\top \in \mathbb{R}^{n\times k}$. Show that

$$\widehat{W} = (X^T X + \lambda I)^{-1} X^T Y$$

b. *[10 points]*

- Implement a function `train` that takes as input $X \in \mathbb{R}^{n\times d}$, $Y \in \{0, 1\}^{n\times k}$, $\lambda > 0$ and returns $\widehat{W} \in \mathbb{R}^{d\times k}$.
- Implement a function `one_hot` that takes as input $Y \in \{0, ..., k - 1\}^n$, and returns $Y \in \{0, 1\}^{n\times k}$.

- Implement a function `predict` that takes as input $W \in \mathbb{R}^{d \times k}$, $X' \in \mathbb{R}^{m \times d}$ and returns an m-length vector with the ith entry equal to $\arg\max_{j=0,\cdots,9} e_j^T W^T x_i'$ where $x_i' \in \mathbb{R}^d$ is a column vector representing the ith example from X'.
- Using the functions you coded above, train a model to estimate $\widehat{W}$ on the MNIST training data with $\lambda = 10^{-4}$, and make label predictions on the test data. This behavior is implemented in `main` function provided in zip file. **What is the training and testing error?** Note that they should both be about 15%.

What to Submit:

- **Part A:** Derivation of expression for $\widehat{W}$
- **Part B:** Values of training and testing errors
- **Code** on Gradescope through coding submission

Administrative

A6.

a. *[2 points]* About how many hours did you spend on this homework? There is no right or wrong answer :)