MATH 156 Machine Learning
Lecture Note

Hanbaek Lyu

Department of Mathematics, University of California, Los Angeles, CA 90095

Email address: hlyu@math.ucla.edu

www.hanbaeklyu.com

Supplementary Repository: https://github.com/HanbaekLyu/Math156_UCLA_SP21

Contents

CHAPTER 1

Introduction to Machine Learning

1. Prologue

When do we say we learned something, say, how to ride bikes or how to multiply two fractions? Roughly speaking, this is to *recognizing patterns* between input and output so that we can use the recognized patterns to *predict* the output of fresh new input. *Machine Learning* (ML) consists of techniques that enable computers to this learning process. Below are some key components in ML.

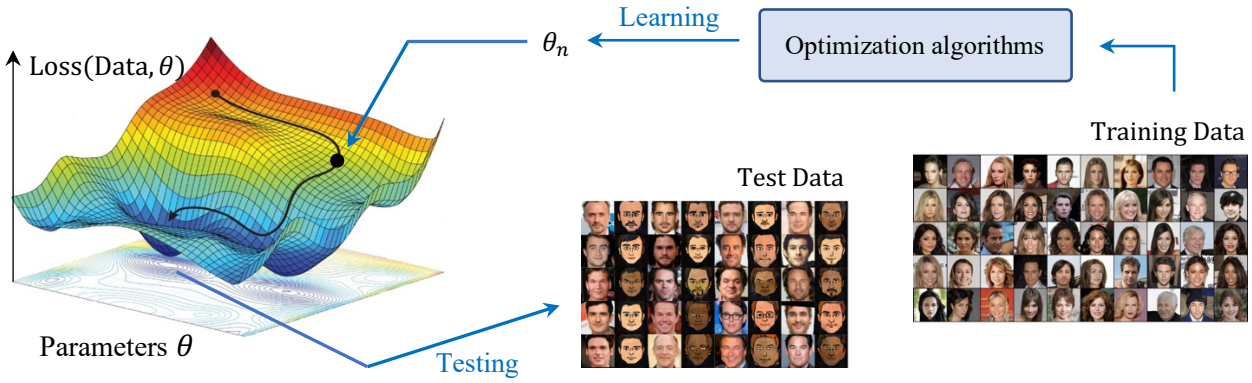


FIGURE 1. Basic scheme of Machine Learning: (1) Select a model to explain the dataset; (2) Fit the parameters to the training dataset using an optimization algorithm; (3) Test the performance of the fitted model to the test dataset.

1. Model Selection: The goal is to predict unknown output $\mathbf{y}$ from a new input $\mathbf{x}$. For this purpose, choose a model $\mathbf{y} \approx \hat{\mathbf{y}}(\mathbf{x}, \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$[1]. Given input $\mathbf{x}$ and parameters $\boldsymbol{\theta}$, the model will output the value $\hat{\mathbf{y}}$ that should be close to the true and unkonwn $\mathbf{y}$.

e.g., Linear regression, Logistic regression, PCA, NMF, feedforward neural networks, CNN, RNN, etc.

2. Traing Data: We need a large number of known input and output pairs $(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_n, \mathbf{y}_N)$, from which we want to recognize patterns between the variables $\mathbf{x}$ and $\mathbf{y}$.

e.g., vectors, matrices, tensors, images, videos, texts, networks, etc.

3. Training the model: We need to find the optimal parameter $\hat{\boldsymbol{\theta}}$ for which the model $\mathbf{x} \mapsto \hat{\mathbf{y}}(\mathbf{x}, \hat{\boldsymbol{\theta}})$ gives a good fit for the training data. More precisely, this is done by solving following optimization problem

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\arg\min}\, \ell(\text{training data}, \boldsymbol{\theta}), \tag{1}$$

where ℓ is a *loss function* that measures the error between the true output y and the predicted output $\hat{\mathbf{y}}$, and $\boldsymbol{\Theta}$ is a set of admissible parameter values. A typical choise of such loss function is

[1]Model parameters are usually denoted by θ in statistics and by w (for 'weights') in machine learning. We will use both interchangably.

the *mean squared error* (MSE)

$$\ell(\text{training data}, \boldsymbol{\theta}) = \frac{1}{N}\sum_{k=1}^{N} \|\mathbf{y}_i - \hat{\mathbf{y}}(\mathbf{x}_i, \boldsymbol{\theta})\|^2, \tag{2}$$

where 'training data' consists of training examples $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)$. Then use *optimizaiton algorithms* to find the optimial parameter $\hat{\boldsymbol{\theta}}$ in (1).

e.g., gradient descent, stochastic gradient descent, expectation-maximization, majorization-minimization, etc.

4. Testing: After an optimial parameter $\hat{\boldsymbol{\theta}}$ is learned, test the 'recognized pattern' $\mathbf{x} \mapsto \hat{\mathbf{y}}(\mathbf{x}, \hat{\boldsymbol{\theta}})$ on *test data* $(\mathbf{x}'_1, \mathbf{y}'_1), \ldots, (\mathbf{x}'_m, \mathbf{y}'_m)$, that were kept hidden during the training step. If the loss $\ell(\text{test data}, \hat{\boldsymbol{\theta}})$ is small, then we have learned the correct pattern succesfully.

2. Polynomial Regression

2.1. Basic theory. Suppose we have a training dataset of pairs of real numbers $(x_1, y_1), \cdots, (x_N, y_N)$. Think of y_k is the price of the stock you are holding and x_k is NASDAQ composite at time k. You want to obtain a simple functional explaining the *dependent variable* y_k in terms of the *independent variable* x_k. While there could be a general relationship between the variables x and y, here we will first study a simple model of *polynomial regression*. Namely, consider the following polynomial function $x \mapsto \hat{y}(x; \mathbf{w})$,

$$\hat{\mathbf{y}}(x; \mathbf{w}) := w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{k=0}^{M} w_k x^M = \boldsymbol{\phi}(x)^T \mathbf{w}, \tag{3}$$

where

$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 & x & \cdots & x^M \end{bmatrix}^T, \qquad \mathbf{w} = \begin{bmatrix} w_0 & w_1 & \cdots & w_M \end{bmatrix}^T \in \mathbb{R}^{M+1}. \tag{4}$$

Here $\mathbf{w} \in \mathbb{R}^{M+1}$ denotes the $(M+1)$-dimensional vector of parameters. Here $\hat{\mathbf{y}}(x; \mathbf{w})$ is a degree M polynomial in x and the components of $\mathbf{w}$ give the coefficients of monomials of x.

Suppose we have training data samples $(x_1, y_1), \cdots, (x_N, y_N)$. Let $\mathbf{Y}_{\text{train}} = [y_1, \cdots, y_N]^T$. We would like to find the optimial parameter $\mathbf{w}$ so that the corresponding polynomial regression (3) gives a good fit for the training data. In order to do so, we first write the vector of n predicted outcomes as the following matrix form

$$\begin{bmatrix} \hat{\mathbf{y}}(x_1, \mathbf{w}) \\ \hat{\mathbf{y}}(x_2, \mathbf{w}) \\ \vdots \\ \hat{\mathbf{y}}(x_N, \mathbf{w}) \end{bmatrix} = \begin{bmatrix} 1 & x_1 & \cdots & x_1^M \\ 1 & x_2 & \cdots & x_2^M \\ & \vdots & & \\ 1 & x_2 & \cdots & x_N^M \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{bmatrix} \quad \Longleftrightarrow \quad \hat{\mathbf{Y}}_{\text{train}} = \mathbf{X}_{\text{train}} \mathbf{w}. \tag{5}$$

Since we want to fit the polynomial regression model in (3) to the training data, we want to choose the parameter $\mathbf{w}$ so that

$$\mathbf{Y}_{\text{train}} \approx \hat{\mathbf{Y}} = \mathbf{X}_{\text{train}} \mathbf{w}. \tag{6}$$

In order to do so, we will use the *method of least squares*. Namely, we seek to choose $\mathbf{w} = \hat{\mathbf{w}}$ where

$$\hat{\mathbf{w}} = \underset{\mathbf{w} \in \mathbb{R}^{M+1}}{\arg\min} \|\mathbf{Y}_{\text{train}} - \mathbf{X}_{\text{train}} \mathbf{w}\|_F^2, \tag{7}$$

where the *matrix Frobenius norm* $\|\cdot\|_F$ is defined by $\|A\|_F = \text{tr}(A^T A)$. When $A = [a_1, \ldots, a_d]^T$ is a column vector, then $A^T A$ is a scalar so that

$$\|A\|_F^2 = \text{tr}(A^T A) = A^T A = a_1^2 + a_2^2 + \cdots + a_d^2, \tag{8}$$

so the Frobenius norm agrees with the Euclidean norm in this case. In Exercise (1.2.1) (with $\mathbf{Y} = \mathbf{Y}_{\text{train}}$, $\mathbf{X} = \mathbf{X}_{\text{train}}$, $\mathbf{B} = \mathbf{w}$, and $\lambda = 0$), we will show that the solution of (7) is given by

$$\hat{\mathbf{w}} = (\mathbf{X}_{\text{train}}^T \mathbf{X}_{\text{train}})^{-1} \mathbf{X}_{\text{train}}^T \mathbf{Y}_{\text{train}}. \tag{9}$$

Hence for any new input x, the optimal polynomial regreesion model will tell us to predict the corresponding ouput y as

$$y \approx \hat{\mathbf{y}}(x;\hat{\mathbf{w}}) = \begin{bmatrix} 1 & x & x^2 & \cdots & x^M \end{bmatrix} \hat{\mathbf{w}}. \tag{10}$$

Exercise 1.2.1 (Method of Least Squares)**.** Suppose we have matrices $\mathbf{Y} \in \mathbb{R}^{d\times n}$ and $\mathbf{X} \in \mathbb{R}^{d\times r}$. We seek to find a matrix $\hat{\mathbf{B}} \in \mathbb{R}^{r\times n}$ where

$$\hat{\mathbf{B}} = \underset{\mathbf{B}\in\mathbb{R}^{r\times n}}{\arg\min} \|\mathbf{Y}-\mathbf{X}\mathbf{B}\|_F^2 + \lambda\|\mathbf{B}\|_F^2. \tag{11}$$

Here $\lambda \geq 0$ is called the L_2-regularization parameter. (This is an instance of unconstrained quadratic optimization problem.)

(i) Show that

$$\|\mathbf{Y}-\mathbf{X}\mathbf{B}\|_F^2 + \lambda\|\mathbf{B}\|_F^2 = \mathrm{tr}(\mathbf{Y}-\mathbf{X}\mathbf{B})^T(\mathbf{Y}-\mathbf{X}\mathbf{B}) + \lambda\mathrm{tr}(\mathbf{B}^T\mathbf{B}) \tag{12}$$

$$= \mathrm{tr}(\mathbf{Y}^T\mathbf{Y}) - 2\mathrm{tr}(\mathbf{Y}^T\mathbf{X}\mathbf{B}) + \mathrm{tr}(\mathbf{B}^T\mathbf{X}^T\mathbf{X}\mathbf{B}) + \lambda\mathrm{tr}(\mathbf{B}^T\mathbf{B}). \tag{13}$$

(ii) Show that (use Exercise 1.2.2)

$$\frac{\partial}{\partial\mathbf{B}}\left(\|\mathbf{Y}-\mathbf{X}\mathbf{B}\|_F^2 + \lambda\|\mathbf{B}\|_F^2\right) = 2(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})\mathbf{B} - 2\mathbf{X}^T\mathbf{Y}, \qquad \frac{\partial^2}{\partial\mathbf{B}^2}\left(\|\mathbf{Y}-\mathbf{X}\mathbf{B}\|_F^2 + \lambda\|\mathbf{B}\|_F^2\right) = 2(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}). \tag{14}$$

(iii) From **(ii)**, conclude that the quadratic function $\mathbf{B} \mapsto \|\mathbf{Y}-\mathbf{X}\mathbf{B}\|_F^2 + \lambda\|\mathbf{B}\|_F^2$ is convex, and and hence its every critical point is a local minimum. (See Ref)

(iv) Suppose $\lambda = 0$ and $\mathbf{X}^T\mathbf{X}$ is invertible[2]. Then from **(ii)** and **(iii)**, conclude that the quadratic function $\mathbf{B} \mapsto \|\mathbf{Y}-\mathbf{X}\mathbf{B}\|_F^2 + \lambda\|\mathbf{B}\|_F^2$ has a unique global minimum $\hat{\mathbf{B}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$ [3]

(v) Suppose $\lambda > 0$. Then argue that $\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}$ is always invertible[4], and the quadratic function $\mathbf{B} \mapsto \|\mathbf{Y}-\mathbf{X}\mathbf{B}\|_F^2 + \lambda\|\mathbf{B}\|_F^2$ has a unique global minimum $\hat{\mathbf{B}} = (\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}$.

Exercise 1.2.2 (Matrix derivatives)**.** Show the following matrix derivatives: (Ref: The Matrix Cookbook)

(i) (First order) $\dfrac{\partial}{\partial\mathbf{X}}\mathrm{tr}(\mathbf{B}\mathbf{X}\mathbf{C}) = \mathbf{B}^T\mathbf{C}^T, \qquad \dfrac{\partial}{\partial\mathbf{X}}\mathrm{tr}(\mathbf{B}\mathbf{X}^T\mathbf{C}) = \mathbf{C}\mathbf{B}.$

(ii) (Second order) $\dfrac{\partial}{\partial\mathbf{X}}\mathrm{tr}(\mathbf{X}^T\mathbf{B}\mathbf{X}) = \mathbf{B}\mathbf{X}+\mathbf{B}^T\mathbf{X}, \qquad \dfrac{\partial}{\partial\mathbf{X}}\mathrm{tr}(\mathbf{B}^T\mathbf{X}^T\mathbf{C}\mathbf{X}\mathbf{B}) = \mathbf{C}^T\mathbf{X}\mathbf{B}\mathbf{B}^T + \mathbf{C}\mathbf{X}\mathbf{B}\mathbf{B}^T.$

2.2. Examples and further discussions. In this subsection, we give an example of polynomial regression on a synthetic data. We will sample various instances of the following random variable

$$Y = \sin(2\pi X) + \varepsilon, \tag{15}$$

where $X \sim \mathrm{Uniform}([0,1])$ and $\varepsilon \sim N(0,\sigma^2)$ with $\sigma = 0.35$, where X and ε are drawn independently. ε represents white noise, where the true signal should lie on the sine curve $y = \sin(2\pi x)$, which is unknown. We can control the signal-noise ratio through the parameter σ, which is the standard deviation of the Gaussian noise ε. For each $N \in \{8, 100\}$, we generate N i.i.d. samples of Y to form the train set, and another N i.i.d. samples of Y forms the test set. The goal is to fit the degree-M polynomial regression model (3) to the train set and learn the optimial coefficients $\hat{\mathbf{w}} \in \mathbb{R}^{M+1}$ by solving the least squares in (7), and use it to predict the test set via (10). Recall the closed-form solution for the optimial parameter $\hat{\mathbf{w}}$ given in (9).

First consider $N = 8$ samples with $M = 1$ degree polynomial regression as shown in Figure 2 left. We are only given the $N = 8$ training points represented by the blue dots. The red line represents the

[2] $\mathbf{X}^T\mathbf{X}$ is symmetric and positive semidefinite, and it is invertible iff the singular values of $\mathbf{X}$ are all nonzero.

[3] The matrix $\mathbf{X}^\dagger := (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is called the *Moore-Penrose pseudo-inverse* of $\mathbf{X}$. If $\mathbf{X}$ is square and invertible, then $\mathbf{X}^\dagger = \mathbf{X}^{-1}(\mathbf{X}^T)^{-1}\mathbf{X}^T = \mathbf{X}^{-1}$. So the the psuedo-inverse can be regarded as a generalization of matrix inverse for non-square matrices.

[4] Hint: Show that $\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}$ is positive definite if $\lambda > 0$. Use the fact that the eigenvalues of a positive definite matrix $\mathbf{A}$ has to be positive (why?), so $\mathbf{A}\mathbf{y} \neq \mathbf{0}$ for any $\mathbf{y}$ (why?) so $\mathbf{A}$ is invertible (why?).

fitted polynomial, which is a linear function being a degree $M = 1$ polynomial. Increasing the degree M of the polynomial model (and hence the degree of freedom of our model) to $M = 4$, we see the quartic polynomial in Figure 2 middle gives a much better fit to the sine curve. However, further increasing M to 9, the polynomial shown in Figure 2 right oscillates more wildly around the sine curve, giving a less favorable explanation of the true signal. While this may seem somewhat counterintuitive as the $M = 9$ polynomial model entails the $M = 4$ polynomial model, what happens here is an issue known as *overfitting*. Indeed, notice that the $M = 9$ polynomial in Figure 2 right passes through every single train data point, while the $M = 4$ polynomial in Figure 2 middle does not. Since the train set contains noise, our model should not be fitted too much to the observed data in the train set as it may learn 'false patterns' misled by the noise.

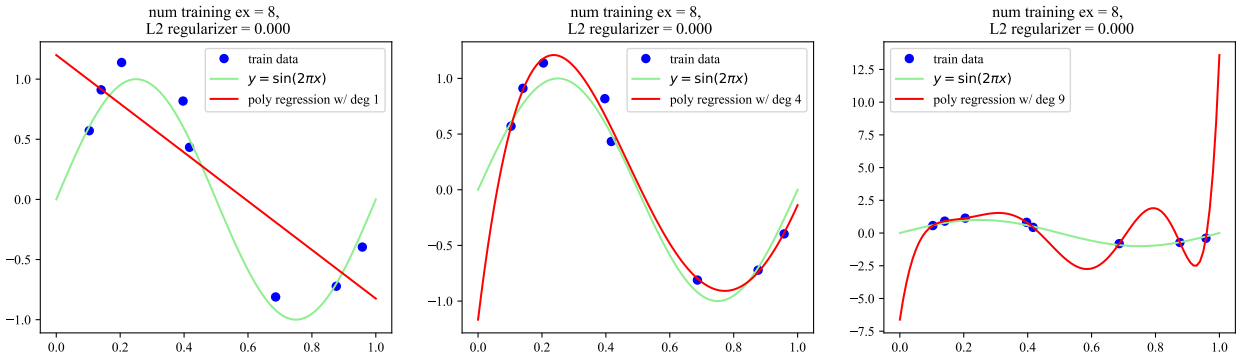


FIGURE 2. Example of polynomial regression fitting. $N = 8$ samples for the train set and $\lambda = 0$ L_2-regularization parameter.

One way to overcome the issue of overfitting is to collect more training data. Indeed, if we use $N = 100$ data points for the train set, the $M = 9$ polynomial regression in Figure 3 right becomes more accurate to the true sine curve than the $M = 4$ case in Figure 3 middle. Since the Gaussian noise in $N = 100$ train samples are all independent, it will be extremely unlikely that many of the points lie on any given $M = 9$ degree polynomial. Even though our synthetic data was built this way, this intuition is often useful in real-world data analysis tasks – noise in nature is close to Gaussian, and fitting to large train set learns 'average patterns', which should be close to the patterns in the original signal.

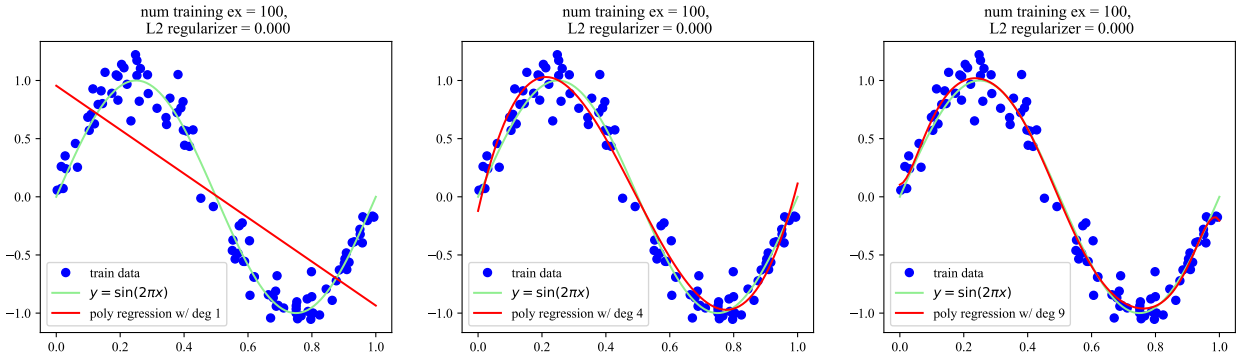


FIGURE 3. Example of polynomial regression fitting. $N = 100$ samples for the train set and $\lambda = 0$ L_2-regularization parameter.

Another way to reduce overfitting is to use *regularization*. Namely, this is to put some penalization on the parameters of the model so that they cannot be too large. For example, consider the following

optimization problem

$$\hat{\mathbf{w}} = \underset{\mathbf{w}\in\mathbb{R}^{M+1}}{\arg\min} \|\mathbf{Y}_{\text{train}} - \mathbf{X}_{\text{train}}\mathbf{w}\|_F^2 + \lambda\|\mathbf{w}\|_F^2, \tag{16}$$

where we have added the 'L_2-regularization term' $\lambda\|\mathbf{w}\|_F^2$ to the squared loss $\|\mathbf{Y}_{\text{train}} - \mathbf{X}_{\text{train}}\mathbf{w}\|_F^2$. Here $\lambda \geq 0$ is called the *L_2-regularizer*, which is a new hyperparameter that we introduced to the model (previously M was the only hyperparameter). Due to this additional penalization term, the coefficients of $\hat{\mathbf{w}}$ in (16) cannot be too large. This technique is also called *shrinkage methods* in statistics as it shrinks the magnitude of the coefficients, and *weight decay* in the context of neural networks. The particular case of a quadratic regularizer as in (16) is called *ridge regression* [HK70]. If we instead use the L_1-regularization $\lambda\|\mathbf{w}\|_1$, then this is known as the (Lagrangian form of) the LASSO regression [Tib96], which better promotes 'sparsity' of $\hat{\mathbf{w}}$. An advantage of using the L_2-regularization is that we still have a closed-form solution for (16) by Exercise 1.2.1:

$$\hat{\mathbf{w}} = (\mathbf{X}_{\text{train}}^T\mathbf{X}_{\text{train}} + \lambda\mathbf{I})^{-1}\mathbf{X}_{\text{train}}^T\mathbf{Y}_{\text{train}}. \tag{17}$$

Indeed, using the L_2-regularizer $\lambda = 0.001$ in the $M = 9$ polynomial regression improves the performance for $N = 8$ training data points, as shown in Figure 4 right.

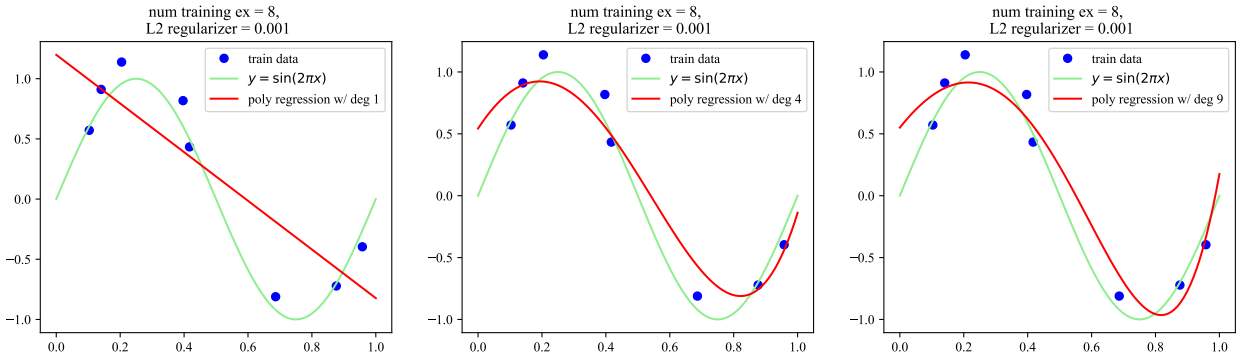


FIGURE 4. Example of polynomial regression fitting. $N = 8$ samples for the train set and $\lambda = 0.001$ L_2-regularization parameter.

Lastly, we evalute the performance of our learned polynomial model on the N test data points. We use the *mean squared error* (MSE)

$$\text{MSE}(\hat{\mathbf{w}}) := \frac{1}{N}\|\mathbf{Y}_{\text{test}} - \hat{\mathbf{Y}}_{\text{test}}\|_F^2, \tag{18}$$

where $\hat{\mathbf{Y}}_{\text{test}} = \mathbf{X}_{\text{test}}\hat{\mathbf{w}}$ and $\hat{\mathbf{w}}$ is given by (17). We use a similar MSE for the train set.

In Figure 5, we show the MSE for train and test sets against the degree M for three L_2-regularization parameters $\lambda \in \{0, 0.001, 0.01\}$ for $N \in \{8, 100\}$ data points both for the test and train sets. While the train error (in blue) mostly decreases as we increase the degree M in all cases, the test error start to increase rapidly at $M = 6$ with no regularization $\lambda = 0$ (Figure 5 left). This indicates severe overfitting occurs past $M = 6$. The gap between the test and train error is also known as the *generalization error*[5]. In Figures 5 middle and right, we see the use of moderate L_2-regularizer $\lambda = 0.001$ could significantly reduce overfitting (and hence the generalization error), but larger values of relgularizer $\lambda = 0.01$ hinders both the test and train errors. In general, the right value of the regularization parameter can be estimated by *cross-valiation*. However, the performance of our model on hyperparameter becomes less sensitive when we have a large traning set ($N = 100$), as we see from Figure 6.

[5]We learn patterns from the observed test set. If the learned patterns capture the generic features of the underlying signal, then they should apply to the unobserved test set and yield a small test error; otherwise, then the pattern we learn does not 'generalize' beyond the test set.

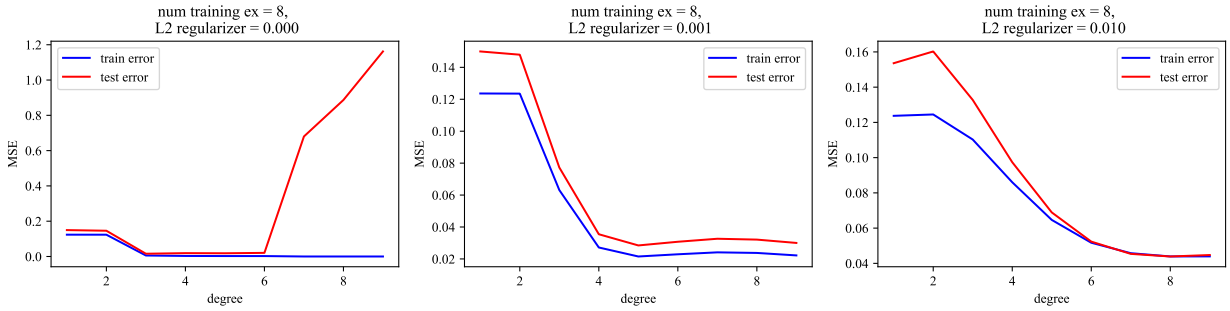


FIGURE 5. Plot of test and train error with respect to the degree M for three L_2-regularization parameters. $N = 8$ points in both train and test sets.

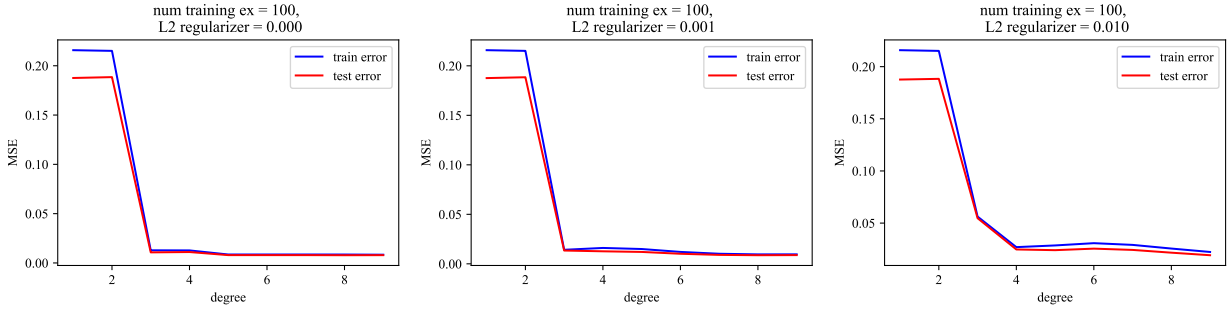


FIGURE 6. Plot of test and train error with respect to the degree M for three L_2-regularization parameters. $N = 100$ points in both train and test sets.

Exercise 1.2.3. Using the Jupyter notebook provided in the course repository, reproduce the Figures 2 - 6, where the data are independently generated from the following random variable

$$Y = \cos(2\pi X) + \varepsilon, \tag{19}$$

where $X \sim \text{Uniform}([0,1])$ and $\varepsilon \sim N(0, 0.16)$ are independent.

2.3. Polynomial Regression by Maximum Likelihood. In this subsection, we consider the polynomial regression with *Gaussian noise*. Namely, given an input x, the model predicts the ouput as a *random variable* (RV) $\hat{\mathbf{y}}(x; \mathbf{w}, \sigma) = \hat{\mathbf{y}}(x; \mathbf{w}) + \varepsilon$, where $\hat{\mathbf{y}}(x; \mathbf{w})$ is the polynomial function given in (3) and $\varepsilon \in N(0, \sigma^2)$ is an independent Gaussian noise with mean 0 and variance σ^2. Note that the predicted output $\hat{\mathbf{y}}(x; \mathbf{w}, \sigma)$ itself is a Gaussian RV with mean $\hat{\mathbf{y}}(x; \mathbf{w})$ and variance σ^2 (see (10)):

$$\hat{\mathbf{y}}(x; \mathbf{w}, \sigma) \sim N(\hat{\mathbf{y}}(x; \mathbf{w}), \sigma^2), \qquad \hat{\mathbf{y}}(x; \mathbf{w}) = \sum_{k=0}^{M} w_k x^k = \boldsymbol{\phi}(x)^T \mathbf{w}. \tag{20}$$

This is a probabilistic model of polynomial regression with parameters $\mathbf{w} = [w_0, \ldots, w_M]^T$ for the coefficients of the unknown polynomial and σ for the standard deviation of the Gaussian noise.

Suppose we are given with a sequence of training examples $[(x_1, y_1), \ldots, (x_N, y_N)]$. In order to fit the probabilistic polynomial regression (20) to the training dataset, we assume that the observed outputs $y_1, \ldots, y_N$ are realizations of the independent RVs $\hat{\mathbf{y}}_1(x_1; \mathbf{w}, \sigma), \ldots, \hat{\mathbf{y}}_N(x_N; \mathbf{w}, \sigma)$ satisfying (20). According to the method of *maximum likelihood* (see Section 6), we will have to find values of the parameters $\mathbf{w}$

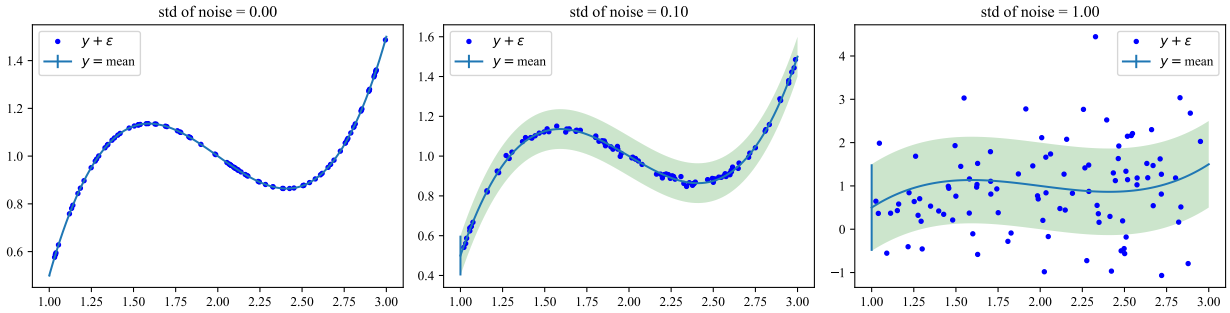


FIGURE 7. Plot of sample points of $Y = -6 + (23/2)x - 6x^2 + x^3 + \varepsilon$, $\varepsilon \sim N(0, \sigma^2)$. Shaded regions represents $\pm$ 1 standard deviation of the noise.

and σ for which the joint likelihood $L(y_1, \ldots, y_N; \mathbf{w}, \sigma)$ of observing the tuple $(y_1, \ldots, y_N)$ from the tuple of independent RVs $\hat{\mathbf{y}}_1(x_1; \mathbf{w}, \sigma), \ldots, \hat{\mathbf{y}}_N(x_N; \mathbf{w}, \sigma)$ is maximized. Namely, the joint likelihood is given by

$$L(y_1, \ldots, y_N; \mathbf{w}, \sigma) = \prod_{i=1}^{n} f_{\hat{\mathbf{y}}(x_i; \mathbf{w}, \sigma)}(y_i) \tag{21}$$

$$= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y_i - \hat{\mathbf{y}}(x_i; \mathbf{w}))^2}{2\sigma^2} \right] \tag{22}$$

$$= (2\pi\sigma^2)^{-N/2} \exp\left[-\frac{1}{2\sigma^2} \sum_{k=1}^{N} (y_i - \hat{\mathbf{y}}(x_i; \mathbf{w}))^2 \right]. \tag{23}$$

A key observation to make is the following: Denoting $\mathbf{Y}_{\text{train}} := [y_1, \ldots, y_N]^T$,

$$\sum_{k=1}^{N} (y_i - \hat{\mathbf{y}}(x_i; \mathbf{w}))^2 = \|\mathbf{Y}_{\text{train}} - \mathbf{X}_{\text{train}}\mathbf{w}\|_F^2, \tag{24}$$

where $\mathbf{X}_{\text{train}}$ is defined in (5) (see Exercise 1.2.4). Thus we can write the log likelihood function as

$$\log L(y_1, \ldots, y_N; \mathbf{w}, \sigma) = -\frac{N}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \|\mathbf{Y}_{\text{train}} - \mathbf{X}_{\text{train}}\mathbf{w}\|_F^2. \tag{25}$$

Noting that log is strictly increasing function, we can find the *maximum likelihood estimate* (MLE) of the parameters $\mathbf{w}$,σ by the following optimization problem

$$(\hat{\mathbf{w}}, \hat{\sigma}) := \underset{\mathbf{w} \in \mathbb{R}^{M+1}, \sigma \geq 0}{\operatorname{argmax}} \; L(y_1, \ldots, y_N; \mathbf{w}, \sigma) \tag{26}$$

$$= \underset{\mathbf{w} \in \mathbb{R}^{M+1}, \sigma \geq 0}{\operatorname{argmax}} \; \log L(y_1, \ldots, y_N; \mathbf{w}, \sigma) \tag{27}$$

$$= \underset{\mathbf{w} \in \mathbb{R}^{M+1}, \sigma \geq 0}{\operatorname{argmax}} \left(-\frac{N}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \|\mathbf{Y}_{\text{train}} - \mathbf{X}_{\text{train}}\mathbf{w}\|_F^2 \right). \tag{28}$$

Since the first term in the log likelihood function in (25) does not depend on $\mathbf{w}$, we see that $\hat{\mathbf{w}}$ is the maximizer of $-\frac{1}{2\sigma^2} \|\mathbf{Y}_{\text{train}} - \mathbf{X}_{\text{train}}\mathbf{w}\|_F^2$. It follows that

$$\hat{\mathbf{w}} = \underset{\mathbf{w} \in \mathbb{R}^{M+1}}{\operatorname{argmin}} \|\mathbf{Y}_{\text{train}} - \mathbf{X}_{\text{train}}\mathbf{w}\|_F^2 = (\mathbf{X}_{\text{train}}^T \mathbf{X}_{\text{train}})^{-1} \mathbf{X}_{\text{train}}^T \mathbf{Y}_{\text{train}}, \tag{29}$$

where the second equality follows from Exercise 1.2.1. Note that this agrees with the optimal parameter $\hat{\mathbf{w}}$ in (10) for the polynomial regression without noise.

On the other hand, in order to find $\hat{\sigma}$, we take the first and the second partial derivatives of (25) in σ:

$$\frac{\partial}{\partial\sigma}\log L(y_1,\dots,y_N;\mathbf{w},\sigma) = -\frac{N}{\sigma}+\frac{1}{\sigma^3}\|\mathbf{Y}_{\text{train}}-\mathbf{X}_{\text{train}}\mathbf{w}\|_F^2, \tag{30}$$

$$\frac{\partial^2}{\partial\sigma^2}\log L(y_1,\dots,y_N;\mathbf{w},\sigma) = -\frac{N}{\sigma^2}-\frac{3}{\sigma^4}\|\mathbf{Y}_{\text{train}}-\mathbf{X}_{\text{train}}\mathbf{w}\|_F^2 \le 0. \tag{31}$$

According to the second derivative, the log likelihood is a concave function in σ. According to the first derivative, the log likelihood has a unique critical point. Since $(\hat{w},\hat{\sigma})$ jointly maximizes the log likelihood, we deduce

$$\hat{\sigma} = \frac{1}{N}\|\mathbf{Y}_{\text{train}}-\mathbf{X}_{\text{train}}\hat{\mathbf{w}}\|_F^2, \tag{32}$$

where $\hat{\mathbf{w}}$ is given in (29).

Hence for any new input x, the optimal polynomial regression model will tell us to predict the corresponding ouput y as the following RV

$$\hat{Y}(x;\hat{\mathbf{w}},\hat{\sigma}) \sim N(\boldsymbol{\phi}(x)^T\hat{\mathbf{w}},\hat{\sigma}^2), \tag{33}$$

where the optimal parameters $\hat{\mathbf{w}}$ and $\hat{\sigma}$ are given in (29) and (32), respectively.

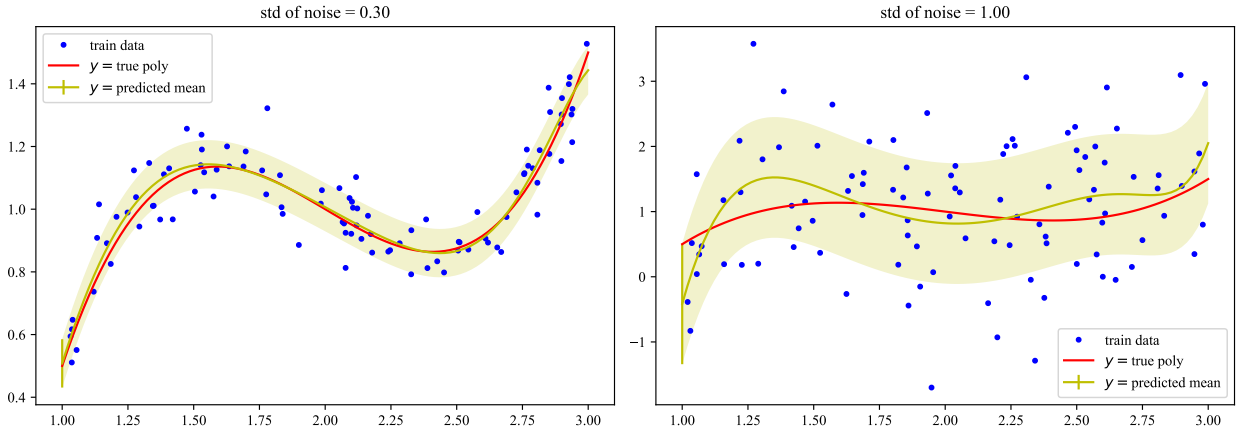


FIGURE 8. Plot of sample points of $Y = -6+(23/2)x-6x^2+x^3+\varepsilon$, $\varepsilon\sim N(0,\sigma^2)$ in blue for $\sigma = 0.3$ (left) and $\sigma = 1$ (right). Degree $M = 9$ polynomial regression with Gaussian noise is fitted by maximum likelihood. The predicted mean is shown in the yellow curve with ± 1 predicted standard deviation ($\hat{\sigma}$) shown in yellow shade.

Exercise 1.2.4. Fix $\mathbf{w} = [w_0, w_1,\dots,w_M]\in\mathbb{R}^{M+1}$ and $\sigma\ge 0$. Let $\hat{Y}_1,\dots,\hat{Y}_N$ be independent Gaussian RVs where $\hat{Y}(x_i;\mathbf{w},\sigma)\sim N(\boldsymbol{\phi}(\mathbf{x})^T\mathbf{w},\sigma^2)$ for $i\in\{1,\dots,N\}$, where $\boldsymbol{\phi}(x) = [1,x,\dots,x^M]^T$. Show that the joint likelihood function for observing the values $y_1,\dots,y_N$ is given by

$$L(y_1,\dots,y_N;\mathbf{w},\sigma) = (2\pi\sigma^2)^{-N/2}\exp\left[-\frac{1}{2\sigma^2}\|\mathbf{Y}-\mathbf{X}\mathbf{w}\|_F^2\right], \tag{34}$$

where

$$\mathbf{Y} = [y_1,\dots,y_N]^T\in\mathbb{R}^N, \qquad \mathbf{X} = \begin{bmatrix} 1 & x_1 & \cdots & x_1^M \\ & \vdots & & \\ 1 & x_2 & \cdots & x_N^M \end{bmatrix}\in\mathbb{R}^{N\times M}. \tag{35}$$

2.4. Bayesian Polynomial Regression. In this subsection, we take a Bayesian approach of the polynomial regression by regarding the $\mathbf{w} = [w_0, \dots, w_M]^T \in \mathbb{R}^{M+1}$ as a (vector-valued) RV. For simplicity, we assume σ in (20) is known.

We assume that the components of $\mathbf{w}$ are i.i.d. Gaussian RV with distribution $N(0, \tau^2)$ for some $\tau \geq 0$. Equivalently, $\mathbf{w}$ follows the multivariate Gaussian distribution $N(\mathbf{0}, \tau\mathbf{I})$ with mean vector $\mathbf{0} = [0, \dots, 0]^T \in \mathbb{R}^{M+1}$ and covariance matrix τI, where $\mathbf{I}$ is the $(M+1)\times(M+1)$ identity matrix. Its probability distribution function (PDF) is given by

$$p(\mathbf{x}; \mathbf{0}, \tau\mathbf{I}) = (2\pi\tau^2)^{-(M+1)/2} \exp\left[-\frac{1}{2\tau^2}\mathbf{x}^T\mathbf{x}\right]. \tag{36}$$

See Example 3.2.2 for the general multivariate Gaussian distribution.

As brefiely introduced in Section 7, Bayesian approach assumes parameters are random and continuously updates a hypothetical probability distribution on the parameters (a.k.a. *prior*) to the new distribution (a.k.a. *posterior*) with respect to the newly observed data according to Bayes' Theorem A.7.1. Namely, as in Subsection 2.3, suppose we are given with training examples $[(x_1, y_1), \dots, (x_N, y_N)]$. Accoding to Bayes' Theorem, Exericse 1.2.4, and (36), the posterior distribution of the coefficients $\mathbf{w}$ is given by the following proportionality relation

$$\overbrace{p(\mathbf{w}|\text{Data})}^{\text{posterior}} \propto \overbrace{L(\text{Data}|\mathbf{w})}^{\text{likelihood}}\overbrace{p(\mathbf{w})}^{\text{prior}} \tag{37}$$

$$= (2\pi\sigma^2)^{-N/2} \exp\left[-\frac{1}{2\sigma^2}\|\mathbf{Y} - \mathbf{X}\mathbf{w}\|_F^2\right] (2\pi\tau^2)^{-(M+1)/2} \exp\left[-\frac{1}{2\tau^2}\mathbf{w}^T\mathbf{w}\right] \tag{38}$$

$$= (2\pi\sigma^2)^{-N/2}(2\pi\tau^2)^{-(M+1)/2} \exp\left[-\frac{1}{2\sigma^2}\|\mathbf{Y} - \mathbf{X}\mathbf{w}\|_F^2 - \frac{1}{2\tau^2}\|\mathbf{w}\|_F^2\right]. \tag{39}$$

Later, we will show that the posterior distribution is in fact a multivarite Gaussian

$$p(\mathbf{w}|\text{Data}) = N(\mathbf{m}, \mathbf{S}), \tag{40}$$

where

$$\mathbf{m} := \sigma^{-2}\mathbf{S}\mathbf{X}_{\text{train}}^T\mathbf{Y}_{\text{train}} \qquad \in \mathbb{R}^{M+1}, \tag{41}$$

$$\mathbf{S}^{-1} := \tau^{-2}\mathbf{I} + \sigma^{-2}\mathbf{X}_{\text{train}}^T\mathbf{X}_{\text{train}} \qquad \in \mathbb{R}^{(M+1)\times(M+1)}. \tag{42}$$

Also, notice that the above posterior distribution is maximized when the exponent is maximized. Hence

$$\hat{\mathbf{w}}_{\text{MAP}} := \underset{\mathbf{w}\in\mathbb{R}^{M+1}}{\arg\max}\, p(\mathbf{w}; \text{Data}) = \underset{\mathbf{w}\in\mathbb{R}^{M+1}}{\arg\min}\left(\|\mathbf{Y} - \mathbf{X}\mathbf{w}\|_F^2 + \frac{\sigma^2}{\tau^2}\|\mathbf{w}\|_F^2\right). \tag{43}$$

The quantity in the left hand side is called *maximum a posteriori* (MAP) estimate of $\mathbf{w}$. Note that the minimization problem in the right hand side is equivalent to (16) with the L_2-regularization parameter $\lambda = (\sigma/\tau)^2$.

Finally, for any new input x, the Bayesian polynomial regression model above will tell us to predict the corresponding ouput y as the following RV

$$\hat{Y}(x; \text{Data}, \sigma) \sim N(\boldsymbol{\phi}(x)^T\mathbf{w}_{\text{pos}}, \sigma^2), \tag{44}$$

where $\mathbf{w}_{\text{pos}}$ is a *random* parameter following the posterior distribution given in (39). In fact, it can be shown that the distribution of $\hat{Y}(x; \text{Data}, \sigma)$ is a (univariate) Gaussian:

$$\hat{Y}(x; \text{Data}, \sigma) \sim N(\boldsymbol{\phi}(x)^T\mathbf{m}, \sigma^2 + \boldsymbol{\phi}(x)^T\mathbf{S}\boldsymbol{\phi}(x)). \tag{45}$$

These claims will be justified later.

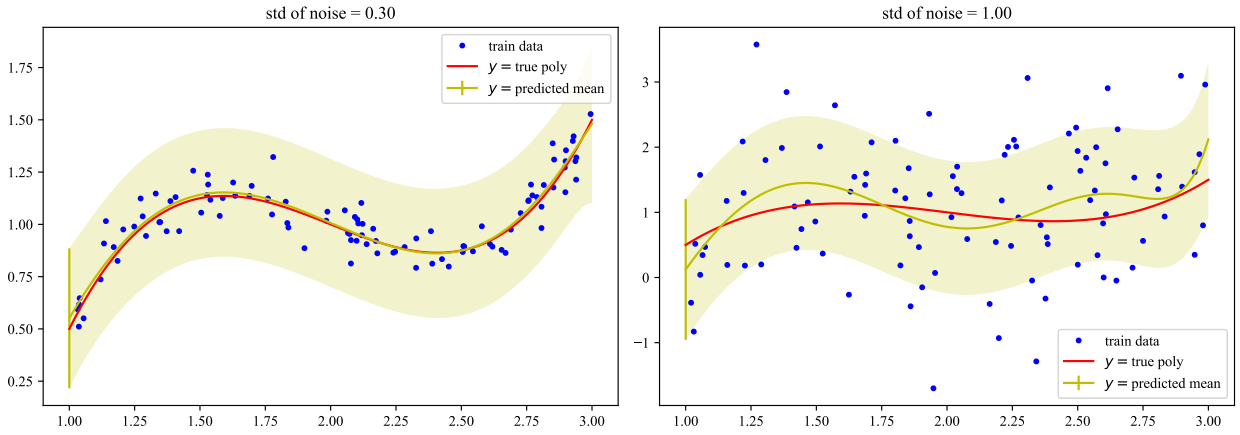


FIGURE 9. Plot of sample points of $Y = -6 + (23/2)x - 6x^2 + x^3 + \varepsilon$, $\varepsilon \sim N(0, \sigma^2)$ in blue for $\sigma = 0.3$ (left) and $\sigma = 1$ (right). Degree $M = 9$ Bayesian polynomial regression with prior $\mathbf{w} \sim N(\mathbf{0}, \tau^2\mathbf{I})$, $\tau = 10$ is fitted. The predicted mean is shown in the yellow curve with ± 1 predicted standard deviation ($\hat{\sigma}$) shown in yellow shade.

Exercise 1.2.5. Using the Jupyter notebook provided in the course repository, reproduce the Figures 8 - 9, where the data are independently generated from the following random variable

$$Y = \cos(2\pi X) + \varepsilon, \tag{46}$$

where $X \sim \text{Uniform}([0,1])$ and $\varepsilon \sim N(0, 0.16)$ are independent. Compare the results of the maximum likelihood and the Bayesian polynomial regression.

3. Decision Theory

Suppose we have an input vector $\mathbf{x} \in \mathscr{X}$ together with a corresponding output vector $\mathbf{y} \in \mathscr{Y}$. Our goal is to predict the output $\mathbf{y}$ given a new value for $\mathbf{x}$, which will take a continuous value for regression problems and discrete value (class label) for classification problems. More formally, a *decision algorithm* is a function $\mathscr{A} : \mathscr{X} \to \mathscr{Y}$ that maps a new input vector $\mathbf{x} \in \mathscr{X}$ to a predicted output $\mathbf{y} \in \mathscr{Y}$.

Method of learning decision algorithms can be roughly divided into 'discriminative' and 'generative' ones. *Discriminative* methods learn decision algorithms directly from training dataset without any probabilistic modling of the relationship between the input $\mathbf{x}$ and the output $\mathbf{y}$. For instance, the polynomial regession model $\mathbf{x} \mapsto \hat{y}(\mathbf{x}; \mathbf{w}) = \boldsymbol{\phi}(x)^T\mathbf{w}$ in (10) is a classical example of discriminative decision algorithms for the case of regression problems. On the other hand, a classical example of discriminative binary classification algorithm is the following *perceptron algorithm*:

$$\mathbf{x} \longmapsto f(\boldsymbol{\phi}(\mathbf{x})^T\mathbf{w}) \in \{+1, -1\}, \tag{47}$$

where $\boldsymbol{\phi}(\mathbf{x})$ is a *feature vector* of the input $\mathbf{x}$ (e.g., $\boldsymbol{\phi}(x) = [1, x, \ldots, x^M]^T$ in the case of polynomial regression) and f is the *activation function* of the form

$$f(x) = \mathbf{1}(x \geq \theta) - \mathbf{1}(x < \theta) = \begin{cases} +1 & x \geq \theta \\ 1 & x < \theta \end{cases}, \tag{48}$$

where $\theta \geq 0$ is a threshold parameter.

In the rest of this section, we take a closer look at the generative methods of learning decision algorithms. This is roughly a two-step process involving the 'inference problem' and the 'decision problem'.

1. Inference problem: We model the unknown input and output as the random vector $(\mathbf{X}, \mathbf{Y})$ following the joint distribution $(\mathbf{x}, \mathbf{y}) \mapsto p(\mathbf{x}, \mathbf{y})$:

$$\mathbb{P}(\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{y}) = p(\mathbf{x}, \mathbf{y}). \tag{49}$$

This describes our complete understanding on the pattern between $\mathbf{x}$ and $\mathbf{y}$, where randomness represents the uncertainty associated to these variables. Conditioning on the input value $\mathbf{x}$, we can also deduce that the output $\mathbf{Y}$ given the observed input $\mathbf{x}$ has the following conditional distribution

$$p(\mathbf{y}\,|\,\mathbf{x}) := \mathbb{P}(\mathbf{Y} = \mathbf{y}\,|\,\mathbf{X} = \mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{x})}, \tag{50}$$

where the marginal distribution of $\mathbf{X}$ is given by

$$p(\mathbf{x}) = \int_{\mathscr{Y}} p(\mathbf{x}, \mathbf{y})\, d\mathbf{y}. \tag{51}$$

It is possible that one only compute the conditional distribution $p(\mathbf{y}\,|\,\mathbf{x})$ in the inference step and proceed to the decision problem (e.g., logistic regression).

2. Decision problem: Given a conditional probability distribution $\mathbf{y} \mapsto p(\mathbf{y}\,|\,\mathbf{x})$ and given a new value of input $\mathbf{x}$, decide (predict) the value of corresponding output $\mathbf{y}$. A general principle to find an optimal decision algorithm using the conditional distribution is the following *(conditional) expected loss minmization* (ELM):

$$\mathbf{x} \longmapsto \mathscr{A}(\mathbf{x}) := \operatorname*{arg\,min}_{\hat{\mathbf{y}} \in \mathscr{Y}} \mathbb{E}\left[\ell(\mathbf{Y}, \hat{\mathbf{y}})\,|\,\mathbf{X} = \mathbf{x}\right], \tag{52}$$

where $(\mathbf{y}, \hat{\mathbf{y}}) \mapsto \ell(\mathbf{y}, \hat{\mathbf{y}}) \in \mathbb{R}$ is a chosen *loss function* that measures the 'goodness' of our prediction $\hat{\mathbf{y}}$ with respect to the true output $\mathbf{y}$.

Below are some instances of decision algorithms found by ELM:

Classicifaion: The prediction $\hat{\mathbf{y}}$ is the maximizer of the conditional distribution $p(\mathbf{y}\,|\,\mathbf{x})$:

$$\hat{\mathbf{y}} = \operatorname*{arg\,max}_{\mathbf{y} \in \mathscr{Y}} p(\mathbf{y}\,|\,\mathbf{x}) = \text{'most probable output given } \mathbf{x}\text{'}. \tag{53}$$

Regression: The prediction $\hat{\mathbf{y}}$ is the conditional expectation of the random output $\mathbf{Y}$ given $\mathbf{x}$:

$$\hat{\mathbf{y}} = \mathbb{E}\,[\mathbf{Y}\,|\,\mathbf{X} = \mathbf{x}] = \text{'expected output given } \mathbf{x}\text{'}. \tag{54}$$

The inference problem is more demanding both theoretically and computationally than solving the decision problem. In practical applications, we must often make a specific prediction for the value of $\mathbf{y}$ in order to take a suitable action based on our current best knowledge of the problem.

Example 1.3.1 (Detecting cancer from x-ray images)**.** Consider the situation where the input is an x-ray image of a new patient, and we would like to determine the output $\mathbf{y}$ between two classes `normal` and `cancer`. Our decision algorithm $\mathscr{A}$ maps the patient's x-ray image $\mathbf{x}$ to a class label $\mathbf{y} \in \{\texttt{normal}, \texttt{cancer}\}$. There are four possible outcomes depending on whether the patient has cancer or not and how she has been diagnosed by our algorithm. Since misclassifying a patient who has cancer may be far more detrimental than misclassifying a patient who does not have cancer, it would be reasonable to consider a loss function ℓ of the following form:

$$\ell(\text{true}, \text{prediction}) = \begin{array}{lcc} \text{true}\backslash\text{prediction} & \texttt{normal} & \texttt{cancer} \\ \texttt{normal} & 0 & 1 \\ \texttt{cancer} & 1000 & 0 \end{array}. \tag{55}$$

Then the decision algorithm $\mathscr{A}$ that minimizes the corresponding expected loss would be more aggressive in looking for cues of cancer and classifying as `cancer` in order to avoid the huge loss incurred when missing patient with cancer. ▲

Example 1.3.2 (Maxizing conditional probabilities for classification)**.** Consider the case of classification problem, where the output variable $\mathbf{Y}$ is a discrete random variable (taking values of class labels). Consider the following 'indicator loss function'

$$\ell(\mathbf{y},\hat{\mathbf{y}}) = \mathbf{1}(\mathbf{y}\neq\hat{\mathbf{y}}). \tag{56}$$

Namely, we gain a unit loss whenever our prediction $\hat{y}$ differs from the true output $\mathbf{y}$. Then note that

$$\mathbb{E}\left[\ell(\mathbf{Y},\hat{\mathbf{y}})\,|\,\mathbf{X}=\mathbf{x}\right] = \mathbb{E}\left[\mathbf{1}(\mathbf{y}\neq\hat{\mathbf{y}})\,|\,\mathbf{X}=\mathbf{x}\right] = \mathbb{P}(\mathbf{Y}\neq\hat{\mathbf{y}}\,|\,\mathbf{X}=\mathbf{x}) = 1-\mathbb{P}(\mathbf{Y}=\hat{\mathbf{y}}\,|\,\mathbf{X}=\mathbf{x}) = 1-p(\hat{\mathbf{y}}\,|\,\mathbf{x}). \tag{57}$$

Thus, the decision algorithm $\mathscr{A}$ given by the ELM (52) in this case reduces to

$$\mathbf{x}\longmapsto\mathscr{A}(\mathbf{x}) = \underset{\hat{\mathbf{y}}\in\mathscr{Y}}{\arg\min}\ \mathbb{E}\left[\ell(\mathbf{Y},\hat{\mathbf{y}})\,|\,\mathbf{X}=\mathbf{x}\right] = \underset{\hat{\mathbf{y}}\in\mathscr{Y}}{\arg\max}\ p(\hat{\mathbf{y}}\,|\,\mathbf{x}). \tag{58}$$

In words, we predict the true class label $\mathbf{Y}$ given $\mathbf{x}$ as the most probable class label given $\mathbf{x}$. ▲

Exercise 1.3.3 (Minimizing squared loss for regression)**.** Consider the case of regression problem, where the output variable $\mathbf{Y}\in\mathbb{R}^d$ is a continuous d-dimensional random vector (e.g., stock prices in a portfolio tomorrow). Consider the squared loss:

$$\ell(\mathbf{y},\hat{\mathbf{y}}) = \|\mathbf{y}-\hat{\mathbf{y}}\|_F^2 = (\mathbf{y}-\hat{\mathbf{y}})^T(\mathbf{y}-\hat{\mathbf{y}}). \tag{59}$$

In this exercise, we will show that the decision algorithm $\mathscr{A}$ given by the ELM (52) in this case reduces to

$$\mathbf{x}\longmapsto\mathscr{A}(\mathbf{x}) = \underset{\hat{\mathbf{y}}\in\mathscr{Y}}{\arg\min}\ \mathbb{E}\left[\ell(\mathbf{Y},\hat{\mathbf{y}})\,|\,\mathbf{X}=\mathbf{x}\right] = \mathbb{E}\left[\mathbf{Y}\,|\,\mathbf{X}=\mathbf{x}\right]. \tag{60}$$

(i) Write $\mathbf{Y} = [Y_1, Y_2, \ldots, Y_d]^T$. Show that

$$\mathbb{E}\left[\|\mathbf{Y}\|_F^2\,\middle|\,\mathbf{X}=\mathbf{x}\right] - \|\mathbb{E}\left[\mathbf{Y}\,|\,\mathbf{X}=\mathbf{x}\right]\|_F^2 = \sum_{i=1}^{d}\mathrm{Var}(Y_i\,|\,\mathbf{X}=\mathbf{x}). \tag{61}$$

(ii) Justify each equality in the following 'completing squares' computation:

$$\mathbb{E}\left[\|\mathbf{Y}-\mathscr{A}(\mathbf{x})\|_F^2\,\middle|\,\mathbf{X}=\mathbf{x}\right] = \mathbb{E}\left[\mathbf{Y}^T\mathbf{Y} - 2\mathscr{A}(\mathbf{x})^T\mathbf{Y} + \mathscr{A}(\mathbf{x})^T\mathscr{A}(\mathbf{x})\,\middle|\,\mathbf{X}=\mathbf{x}\right] \tag{62}$$

$$= \mathbb{E}\left[\|\mathbf{Y}\|_F^2\,\middle|\,\mathbf{X}=\mathbf{x}\right] - 2\mathscr{A}(\mathbf{x})^T\mathbb{E}\left[\mathbf{Y}\,|\,\mathbf{X}=\mathbf{x}\right] + \|\mathscr{A}(\mathbf{x})\|_F^2 \tag{63}$$

$$= \|\mathscr{A}(\mathbf{x}) - \mathbb{E}\left[\mathbf{Y}\,|\,\mathbf{X}=\mathbf{x}\right]\|_F^2 + \sum_{i=1}^{d}\mathrm{Var}(Y_i\,|\,\mathbf{X}=\mathbf{x}). \tag{64}$$

(iii) From **(ii)**, deduce that the expected squared loss $\mathbb{E}\left[\ell(\mathbf{Y},\hat{\mathbf{y}})\,|\,\mathbf{X}=\mathbf{x}\right]$ of the decision algorithm $\mathbf{x}\mapsto\mathscr{A}(\mathbf{x})$ is minimized when $\mathscr{A}(\mathbf{x}) = \mathbb{E}\left[\mathbf{Y}\,|\,\mathbf{X}=\mathbf{x}\right]$ and the global minimum equals $\sum_{i=1}^{d}\mathrm{Var}(Y_i\,|\,\mathbf{X}=\mathbf{x})$.

In the following proposition, we show that minimizing 'total expected loss' and 'conditional expected loss' give the same decision algorithm.

Proposition 1.3.4 (Total expected loss minimization)**.** *Consider the following decision algorithm $\mathscr{B}$: $\mathscr{X}\to\mathscr{Y}$ given by the following total expected loss minimization:*

$$\mathscr{B} = \underset{f:\mathscr{X}\to\mathscr{Y}}{\arg\min}\ \mathbb{E}\left[\ell(\mathbf{Y},f(\mathbf{X}))\right], \tag{65}$$

where $(\mathbf{y},\hat{\mathbf{y}})\mapsto\ell(\mathbf{y},\hat{\mathbf{y}})$ is a chosen loss function. Let $\mathscr{A}$ denote the decision algorithm given by the (conditional) expected loss minimization:

$$\mathbf{x}\longmapsto\mathscr{A}(\mathbf{x}) = \underset{\hat{\mathbf{y}}\in\mathscr{Y}}{\arg\min}\ \mathbb{E}\left[\ell(\mathbf{Y},\hat{\mathbf{y}})\,|\,\mathbf{X}=\mathbf{x}\right]. \tag{66}$$

Then it holds that $\mathscr{B}=\mathscr{A}$.

PROOF. It suffices to show that $\mathscr{A}$ achieves the minimum total expected loss:

$$\mathbb{E}\left[\ell(\mathbf{Y},\mathscr{A}(\mathbf{X}))\right] = \min_{f:\mathscr{X}\to\mathscr{Y}} \mathbb{E}\left[\ell(\mathbf{Y},f(\mathbf{X}))\right] = \mathbb{E}\left[\ell(\mathbf{Y},\mathscr{B}(\mathbf{X}))\right], \tag{67}$$

where the second equality follows from the definition of $\mathscr{B}$. First, since $\mathscr{B}$ minimizes the total expected loss and $\mathscr{A}$ is one of the possible decision algorithms,

$$\mathbb{E}\left[\ell(\mathbf{Y},\mathscr{A}(\mathbf{X}))\right] \geq \mathbb{E}\left[\ell(\mathbf{Y},\mathscr{B}(\mathbf{X}))\right]. \tag{68}$$

On the other hand, by the minimality of $\mathscr{A}$, we have for each $\mathbf{x}\in\mathscr{X}$,

$$\mathbb{E}_{\mathbf{Y}}\left[\ell(\mathbf{Y},\mathscr{B}(\mathbf{X}))\,|\,\mathbf{X}=\mathbf{x}\right] \geq \mathbb{E}\left[\ell(\mathbf{Y},\mathscr{A}(\mathbf{x}))\,|\,\mathbf{X}=\mathbf{x}\right] \tag{69}$$

By taking expectation over $\mathbf{X}$ and using iterated expectation,

$$\mathbb{E}\left[\ell(\mathbf{Y},\mathscr{B}(\mathbf{X}))\right] = \mathbb{E}_{\mathbf{X}}\left[\mathbb{E}_{\mathbf{Y}}\left[\ell(\mathbf{Y},\mathscr{B}(\mathbf{X}))\,|\,\mathbf{X}\right]\right] \geq \mathbb{E}_{\mathbf{X}}\left[\mathbb{E}_{\mathbf{Y}}\left[\ell(\mathbf{Y},\mathscr{A}(\mathbf{X}))\,|\,\mathbf{X}\right]\right] = \mathbb{E}\left[\ell(\mathbf{Y},\mathscr{A}(\mathbf{X}))\right]. \tag{70}$$

Hence by combining (68) and (70), we conclude (67), as desired. □

CHAPTER 2

Algorithms for Classification

1. Introduction

A particular type of pattern reconition problem that machine learning methods have made significant progress over the past decades is *classiciation problems*. Here are some motivating examples:

1. Classify patients' chest x-ray images into classes '`normal`' and '`pneumonia`'.
2. Classify dog and cat images into classes '`dog`' and '`cat`'.
3. Classify an image of handwritten digit into the correct digit. (see Figure 10)
4. Classify incoming emails into '`spam`' and '`not spam`'.
5. Classify given user behavior on a social network into '`churn`' or '`human`'

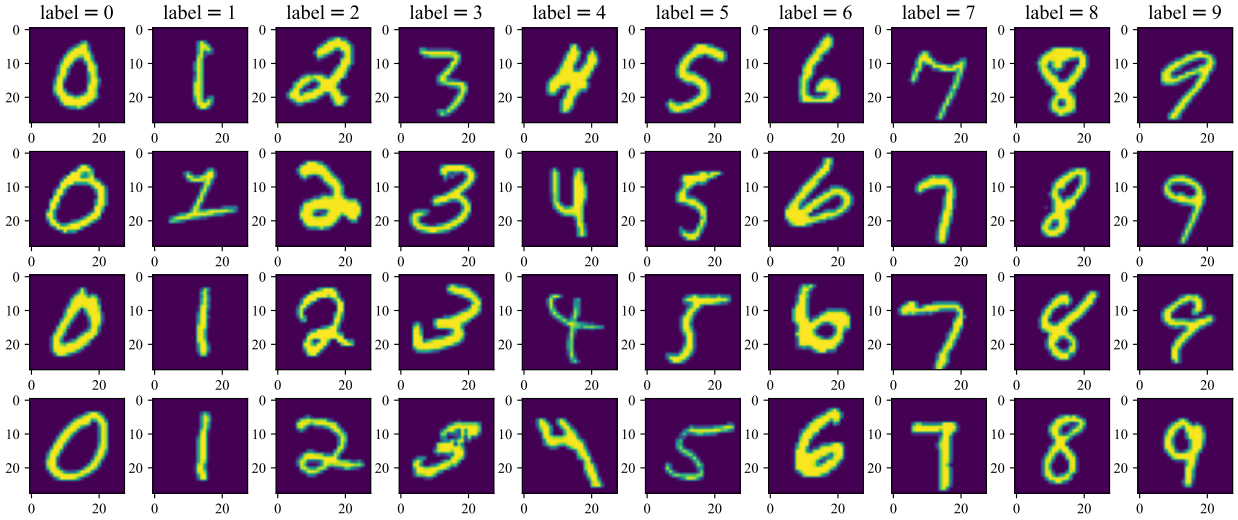


FIGURE 10. Examples of MNIST dataset of handwritten digits by Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. There are total 70000 images of size 28×28 pixels.

In general, for classification problem, we seek to learn a decision algorithm $\mathscr{A} : \mathscr{X} \to \mathscr{Y} = \{\mathscr{C}_1, \ldots, \mathscr{C}_\kappa\}$, which maps each object $\mathbf{x} \in \mathscr{X}$ into one of κ possible classes $\mathscr{C}_1, \ldots, \mathscr{C}_\kappa$. This problem is called a *binary classification* if $\kappa = 2$ and a *multi-class classification* if $\kappa \geq 3$. For instance, the MNST dataset consists of total 70000 black-and-white images of size 28×28 pixels. Viewing each image as a matrix in $\mathbb{R}^{28\times 28}$, the classification problem for the MNST dataset $\mathscr{X}_{\text{MNIST}}$ can be thought of as a multi-class classification problem where one wants to learn a decision algorithm $\mathscr{A} : \mathbb{R}^{28\times 28} \supseteq \mathscr{X}_{\text{MNIST}} \to \{0, 1, 2, \ldots, 9\}$. Equivalently, by vectorizing each image into a $784 = 28^2$ dimensional vector, this is also to learn a decision algorithm $\mathscr{A} :\subseteq \mathbb{R}^{784} \supseteq \mathscr{X}_{\text{MNIST}} \to \{0, 1, 2, \ldots, 9\}$.

Some of the popular classification algorithms are: Logistic Regression, k-Nearest Neighbors, Decision Trees, Support Vector Machine, and Naive Bayes. We will take a closer look at some of the algorithms in this chapter.

2. Logistic Regression

2.1. The model formulation. We begin our discussion for the simplest case of binary classification. Without loss of generality, we can assume that the two classes are 0 and 1. Say we have training examples $(\boldsymbol{\phi}(\mathbf{x}_1), y_1), \ldots, (\boldsymbol{\phi}(\mathbf{x}_N), y_N)$, where

- $\mathbf{x}_1, \ldots, \mathbf{x}_N$: Input data (e.g., collection of all medical records of each patient)
- $\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_N) \in \mathbb{R}^p$: Features (e.g., some useful (derived) information for each patient)
- $y_1, \ldots, y_n \in \{0, 1\}$: Binary class labels (e.g., '`normal`' or '`pneumonia`').

The basic idea of logistic regression is to model the output y as a Bernoulli (0-1) random variable $Y \sim \text{Bernoulli}(p)$ with success probability p that depends on the observed feature $\boldsymbol{\phi}(\mathbf{x})$. More specifically,

$$Y \mid \boldsymbol{\phi}(\mathbf{x}) \sim \text{Bernoulli}(p), \qquad p = \sigma(\boldsymbol{\phi}(\mathbf{x})^T \mathbf{w}) := \frac{\exp(\boldsymbol{\phi}(\mathbf{x})^T \mathbf{w})}{1 + \exp(\boldsymbol{\phi}(\mathbf{x})^T \mathbf{w})} \tag{71}$$

Here, $\sigma : x \mapsto e^x/(1+e^x) = 1/(1+e^{-x})$ is called a *sigmoid* (or *logistic*) function, and $\mathbf{w} \in \mathbb{R}^p$ is a p-dimensional parameter vector. Notice that the sigmoid function maps the real line onto the interval $[0, 1]$ (see Figure 11). Hence one can think of it as a function that 'links' real scalar with an associated probability. So in logistic regression, we model the 'success probability' p given the feature $\boldsymbol{\phi}(\mathbf{x})$ as the dot product $\boldsymbol{\phi}(\mathbf{x})^T \mathbf{w} \in \mathbb{R}$ fed into the sigmoid function σ in order to make it a probability $\in [0, 1]$.

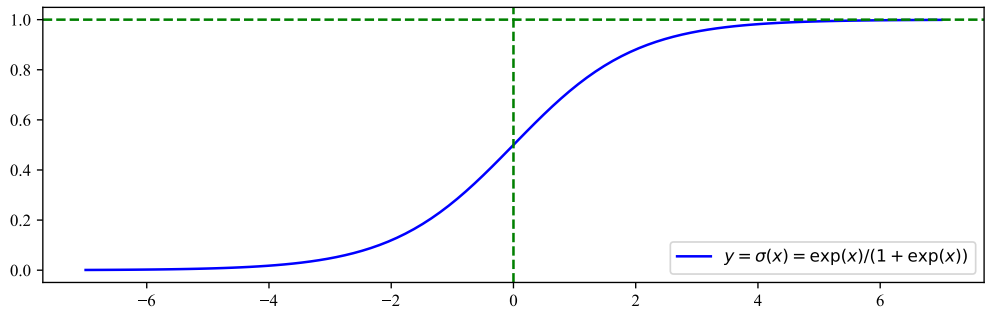


FIGURE 11. Plot of sigmoid function.

Next, let's discuss how to train logistic regression model. Since it is a probabilistic model, we will use the method of maximum likelihood. For that, we compute the joint likelihood of observing the output values $(y_1, \ldots, y_N)$ from the tuple of independent Bernoulli variables $(Y_1, \ldots, Y_N)$ with success probabilities $p_1 := \sigma(\boldsymbol{\phi}(\mathbf{x}_1)^T \mathbf{w}), \ldots, p_N := \sigma(\boldsymbol{\phi}(\mathbf{x}_N)^T \mathbf{w})$:

$$L(y_1, \ldots, y_N; \mathbf{w}) = \mathbb{P}(Y_1 = y_1, \ldots, Y_N = y_N; \mathbf{w}) = \prod_{i=1}^{N} p_i^{y_i} (1 - p_i)^{1 - y_i}. \tag{72}$$

Hence the optimial parameter $\hat{\mathbf{w}} \in \mathbb{R}^p$ is given by

$$\hat{\mathbf{w}} = \operatorname*{argmax}_{\mathbf{w} \in \mathbb{R}^p} \log L(y_1, \ldots, y_N; \mathbf{w}) \tag{73}$$

$$= \operatorname*{argmax}_{\mathbf{w} \in \mathbb{R}^p} \sum_{i=1}^{N} y_i \log p_i + (1 - y_i) \log(1 - p_i) \tag{74}$$

$$= \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^p} \sum_{i=1}^{N} y_i \log(p_i^{-1} - 1) - \log(1 - p_i) \tag{75}$$

$$= \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^p} \sum_{i=1}^{N} -y_i \boldsymbol{\phi}(\mathbf{x}_i)^T \mathbf{w} + \log\left(1 + \exp(\boldsymbol{\phi}(\mathbf{x}_i)^T \mathbf{w})\right), \tag{76}$$

where we have used the following observations

$$p_i^{-1} - 1 = \exp(-\boldsymbol{\phi}(\mathbf{x}_i)^T \mathbf{w}), \qquad (1-p_i)^{-1} = 1 + \exp(\boldsymbol{\phi}(\mathbf{x}_i)^T \mathbf{w}). \tag{77}$$

Using a matrix formulation, we have

$$\hat{\mathbf{w}} = \underset{\mathbf{w}\in\mathbb{R}^p}{\arg\min} \left[\ell_{LR}(\mathbf{w}) := \left(\sum_{i=1}^{N} \log\left(1 + \exp(\boldsymbol{\phi}(\mathbf{x}_i)^T \mathbf{w})\right) \right) - \mathbf{Y}^T \boldsymbol{\Phi}^T \mathbf{w} \right], \tag{78}$$

where

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \in \{0,1\}^N, \qquad \boldsymbol{\Phi} = \begin{bmatrix} \uparrow & & \uparrow \\ \boldsymbol{\phi}(\mathbf{x}_1) & \cdots & \boldsymbol{\phi}(\mathbf{x}_N) \\ \downarrow & & \downarrow \end{bmatrix} \in \mathbb{R}^{p\times N}. \tag{79}$$

Here $\boldsymbol{\Phi}$ is also called the *design matrix*. The loss function $\ell_{LR}(\mathbf{w})$ in (78) is called the *logistic regression loss*.

2.2. Minimizing the logistic regression loss. In the previous section, we have formulated the logistic regression model and derived an optimization problem (78) for its training. Namely, we need to minimize the logistic regression loss ℓ_{LR} defined in (78) in order to find the optimal (maximum likelihood) parameter $\hat{\mathbf{w}}$. Recall that in the case of polynomial regression, the quadratic optimization problem (7) had a closed-form solution for the optimal $\hat{\mathbf{w}}$. In the case of the logistic regression, we will observe that there is no closed-form solution for the minimizer, but it is still a *convex optimization problem*, which can be solved effectively by an iterative algorithm (e.g., gradient descent, Newton-Ralphson).

The following exercise provides the gradient, the Hessian, and convexity of the logistic regression loss ℓ_{LR}.

Exercise 2.2.1 (Convexity of the logistic regression loss)**.** Let $\ell_{LR}(\mathbf{w})$ denote the logistic regression loss defined in (78). Let $\mathbf{Q} = \mathbf{Q}(\mathbf{w}) = [p_1, \ldots, p_N]^T \in \mathbb{R}^N$ where $p_i = \sigma(\boldsymbol{\phi}(\mathbf{x}_i)^T \mathbf{w})$ with $\sigma(x) = e^x/(1+e^x)$.

(i) Show that $\frac{\partial \sigma(x)}{x} = \sigma(x)(1-\sigma(x))$.

(ii) Show that

$$\frac{\partial \mathbf{Q}^T}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial}{\partial w_1} \\ \vdots \\ \frac{\partial}{\partial w_p} \end{bmatrix} \begin{bmatrix} p_1 & \cdots & p_N \end{bmatrix} = \begin{bmatrix} \uparrow & & \uparrow \\ p_1(1-p_1)\boldsymbol{\phi}(\mathbf{x}_1) & \cdots & p_N(1-p_N)\boldsymbol{\phi}(\mathbf{x}_N) \\ \downarrow & & \downarrow \end{bmatrix} = \boldsymbol{\Phi}\mathbf{D}, \tag{80}$$

where $\mathbf{D}$ is the $N \times N$ diagonal matrix with diagonal entries $\mathbf{D}_{ii} = p_i(1-p_i)$.

(iii) Show that

$$\nabla_{\mathbf{w}} \ell_{LR}(\mathbf{w}) = \boldsymbol{\Phi}(\mathbf{Q} - \mathbf{Y}) \in \mathbb{R}^p, \qquad \mathbf{H} := \frac{\partial^2 \ell_{LR}(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^T} = \nabla_{\mathbf{w}} \mathbf{Q}^T \boldsymbol{\Phi}^T = \boldsymbol{\Phi}\mathbf{D}\boldsymbol{\Phi}^T \in \mathbb{R}^{p\times p}. \tag{81}$$

(iv) Argue that the Hessian $\mathbf{H}$ in **(iii)** is positive semi-definite. Conclude that the logistic regression loss ℓ_{LR} is convex. (Hint: $\mathbf{D}$ is a diagonal matrix with positive diagonal entries.)

Now, we will introduce two convex optimization algorithms for logistic regression. The algorithms are iterative in nature, meaning that they apply the same update rule until a certain convergence criterion is satisfied. The first algorithm belongs to the class of *gradient descent*:

$$\mathbf{w}_{t+1} \longleftarrow \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} \ell_{LR}(\mathbf{w}_t) = \mathbf{w}_t - \eta_t \boldsymbol{\Phi}(\mathbf{Q}(\mathbf{w}_t) - \mathbf{Y}), \tag{82}$$

where $(\eta_t)_{t\geq 0}$ is a sequence of positive numbers called *step sizes*. Recall that the gradient $\nabla_{\mathbf{w}} \ell_{LR}$ is the direction of change in $\mathbf{w}$ that increases the loss function ℓ_{LR} most rapidly. Since we want to minimize the loss function, we would want to incrementally change the current iterate $\mathbf{w}_t$ to the opposite direction of the gradient. The remaining degree of freedom is the step size that tells us how far we follow such

direction $-\nabla_{\mathbf{w}}\ell_{LR}(\mathbf{w}_t)$ of the steepest descent[1]. A particular choice that is guaranteed to converge to a global minimum for convex optimization problems is $\eta_t = \gamma(\log t)/\sqrt{t}$, where $\gamma > 0$ is a constant[2].

The second algorithm also belongs to the class of gradient descent but uses the Hessian $\mathbf{H} = \nabla^2\ell_{LR}(\mathbf{w}_t)$, which is called the *Newton-Ralphson method*. It takes the following form:

$$\mathbf{w}_{t+1} \longleftarrow \mathbf{w}_t - \mathbf{H}^{-1}\nabla_{\mathbf{w}}\ell_{LR}(\mathbf{w}_t). \tag{83}$$

One can think of the above algorithm as a vresion of gradient descent where we used the Hessian inverse $\mathbf{H}^{-1}$ as a 'matrix-valued step size'[3]. The use of second-order derivatives often accelerates the speed of convergence but suffers from higher computational cost per iteration. In our current case of logistic regression, the Newton-Ralphson algorithm in (83) takes the following simple form

$$\mathbf{w}_{t+1} \longleftarrow \mathbf{w}_t - (\mathbf{\Phi}\mathbf{D}\mathbf{\Phi}^T)^{-1}\mathbf{\Phi}(\mathbf{Q}-\mathbf{Y}) \tag{84}$$

$$= (\mathbf{\Phi}\mathbf{D}\mathbf{\Phi}^T)^{-1}\big(\mathbf{\Phi}\mathbf{D}\mathbf{\Phi}^T\mathbf{w}_t - \mathbf{\Phi}(\mathbf{Q}-\mathbf{Y})\big) \tag{85}$$

$$= (\mathbf{\Phi}\mathbf{D}\mathbf{\Phi}^T)^{-1}\mathbf{\Phi}\mathbf{D}\mathbf{z}, \tag{86}$$

where $\mathbf{D} \in \mathbb{R}^{N\times N}$ is the diagonal matrix defined in Exercise 2.6.1 and we have denoted $\mathbf{z} := \mathbf{\Phi}^T\mathbf{w}_t - \mathbf{D}^{-1}(\mathbf{Q}-\mathbf{Y}) \in \mathbb{R}^N$. From the analysis in Exercise 1.2.1, one can observe that the last expression is the solution of the following 'weighted least sqaures problem'

$$\tilde{\mathbf{w}} := \underset{\mathbf{w}\in\mathbb{R}^p}{\arg\min}\, \|\mathbf{z} - \sqrt{\mathbf{D}}\mathbf{\Phi}^T\mathbf{w}\|_F^2. \tag{87}$$

However, both the target vector $\mathbf{z}$ and the weight matrix $\sqrt{\mathbf{D}}$ change when we update $\mathbf{w}_t$. Hence one has to apply the above formula iteratively. For these reasons, the Newton-Ralphson algorithm (83) for logistic regression is also called *iterative reweighted least squares* (IRWS).

Exercise 2.2.2. The gradient descent (GD) algorithm for logistic regression training in (82) is implemented as the python function 'fit_LR_GD' in the Jupyter notebook provided in the course repository.

```
def fit_LR_GD(Y, H, W0=None, sub_iter=100, stopping_diff=0.01):
        '''
        Convex optimization algorithm for Logistic Regression using Gradient Descent
        Y = (n x 1), H = (p x n) (\Phi in lecture note), W = (p x 1)
        Logistic Regression: Y ~ Bernoulli(Q), Q = sigmoid(H.T @ W)
        MLE -->
        Find \hat{W} = argmin_W ( sum_j ( log(1+exp(H_j.T @ W) ) - Y.T @ H.T @ W ) )
        '''
        if W0 is None:
            W0 = np.random.rand(H.shape[0],1) #If initial coefficients W0 is None, randomly initialize

        W1 = W0.copy()
        i = 0
        dist = 1
        grad = np.ones(W0.shape)
        while (i < sub_iter) and (np.linalg.norm(grad) > stopping_diff):
            Q = 1/(1+np.exp(-H.T @ W1))  # probability matrix, same shape as Y
            # grad = H @ (Q - Y).T + alpha * np.ones(W0.shape[1])
            grad = H @ (Q - Y)
            W1 = W1 - (np.log(i+1) / (((i + 1) ** (0.5)))) * grad
            i = i + 1
            # print('iter %i, grad_norm %f' %(i, np.linalg.norm(grad)))
        return W1
```

FIGURE 12. A python implementation of the gradient descent algorithm for logistic regression training (82).

[1] See Figure 1: The trajectory on the graph of the loss function can be viewed as that of a gradient descent algorithm.

[2] This is an instance of dinimishing step sizes, where the step sizes are required to be non-summable $\sum_{t=1}^{\infty}\eta_t = \infty$ but square-summable $\sum_{t=1}^{\infty}\eta_t^2 < \infty$. See [SSY18].

[3] The logic behind this algorithm is to minimize the second-order Taylor approximation of the objective ℓ_{LR}. Namely, $\ell_{LR}(\mathbf{w}) = \ell_{LR}(\mathbf{w}_t) + \nabla\ell_{LR}(\mathbf{w}_t)^T(\mathbf{w}-\mathbf{w}_t) + \frac{1}{2}(\mathbf{w}-\mathbf{w}_t)^T\nabla^2\ell_{LR}(\mathbf{w}_t)(\mathbf{w}-\mathbf{w}_t) + O(\|\mathbf{w}-\mathbf{w}_t\|_F^3)$, and the minimizer of the quadratic function here is exactly (83).

(i) Modify the code (or make your own function) so that each gradient descent step (82), the current value of objective function ℓ_{LR} in (78) is printed. Test it on the MNIST example with digits ['0','1']. Does the loss monotonically decrease?

(ii) The implemented version of GD for (82) uses the step size $\eta_t = \gamma \log(t+1)/\sqrt{(t+1)}$ with $\gamma = 1$. Test the same step sizes for $\gamma = 10$ and also $\eta_t = 1$ and $\eta_t = 1/t$. Test it on the MNIST example with digits ['0','1']. Is there any difference in the way the loss function decrease?

(iii) If we use the step size of the form $\eta_t = \gamma t^{-\delta}$ for $0 \le \delta \le 1$, what would be your optimal choice of the hyperparameters γ and δ? (There is no answer and trying out a few choices and make your observation.)

Exercise 2.2.3. See the Jupyter notebook provided in the course repository.

```
def fit_LR_NR(Y, H, W0=None, sub_iter=100, stopping_diff=0.01):
    '''
    Convex optimization algorithm for Logistic Regression using Newton-Ralphson algorithm.
    Y = (n x 1), H = (p x n) (\Phi in lecture note), W = (p x 1)
    Logistic Regression: Y ~ Bernoulli(Q), Q = sigmoid(H.T @ W)
    MLE -->
    Find \hat{W} = argmin_W ( sum_j ( log(1+exp(H_j.T @ W) ) - Y.T @ H.T @ W ) )
    '''
    ### Implement by yourself.
```

FIGURE 13. A python implementation of the Newton-Ralphson algorithm for logistic regression training (83) to be completed.

(i) Implement the Newton-Ralphson algorithm for logistic regression training in (83) in the same notebook, and reproduce Figure 23. How does the result compare with two optimization algorithms?

(ii) Compare with the gradient descent algorithm. Which one do you think is faster? Which one do you think is more efficient? (Do some experiments make your own argument. There is no single answer.)

3. Binary classification of MNIST by Logistic Regression

3.1. Training. Recall that the MNIST dataset (see Figure 10) consists of total 70000 black-and-white images of size 28×28 pixels, corresponding to one of the 10 digits from $\{0, 1, \ldots, 9\}$. In this subsection, we will apply logistic regression to classify the MNIST images of chosen two digits.

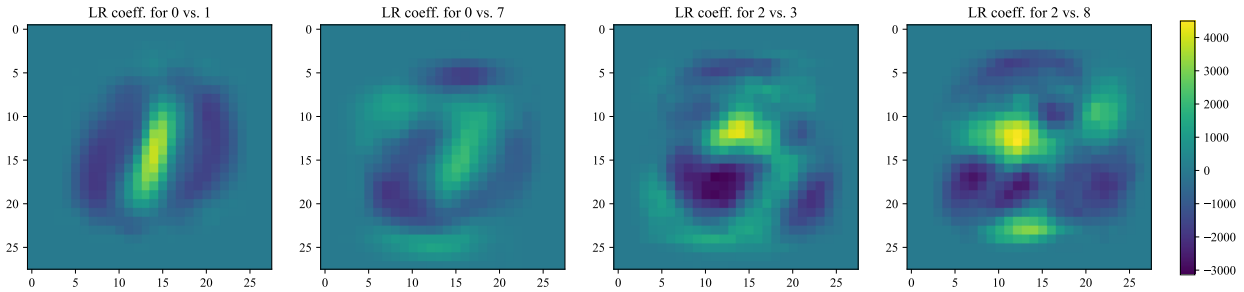


FIGURE 14. Examples of the fitted logistic regression coefficients to the MNIST dataset with chosen two digits. Gradient descent algorithm is used for training.

For instance, suppose we choose the images for digits '0' and '1'. There are total 6903 and 7877 images of digits '0' and '1', respectively. Viewing each image as a matrix in $\mathbb{R}^{28\times 28}$, the binary classification problem now is to learn a decision algorithm $\mathscr{A} : \mathbb{R}^{28\times 28} \supseteq \{\text{Images of '0' or '1'}\} \to \{0, 1\}$, where class label 1 means the image is classified as digit '1' and label 0 for digit '0'. The goal is to learn the optimal logistic regression parameter $\hat{\mathbf{w}} \in \mathbb{R}^{28\times 28}$ by solving the optimization problem (78). The meaning of the (i, j) entry $\hat{\mathbf{w}}[i, j]$ is the positive association of the (i, j) pixel of all 28×28 images in the train set with class label 1.

The first subplot in Figure 23 displays $\hat{\mathbf{w}}$ for digits '0′ and '1′, learned by the gradient descent algorithm in (82) with $\gamma = 1$. One can see that the pixels lying on the overall shape of digit '1' have large regression coefficients, meaning that they contribute to the image being classified as '1' with a large probability. On the other hand, one can also see that the pixels lying on the overall shape of digit '0' have small regression coefficients, meaning that they discount the probability of the image being classified as '1' and hence encourage it to be classified as '0'. The results for other digit pairs ['0′, '7′], ['2′, '3′], and ['2′, '8′] are also shown in Figure 23.

Notice such a pixel-based regression for image classification would make sense only if the images are well-aligned. For example, if the images for digit '1' show up in all different locations and orientations and even with various scaling, then it won't be possible to attribute any pixel (location in the image) as an important feature of being class 1.

Exercise 2.3.1. See the Jupyter notebook provided in the course repository.

(i) Reproduce the experiments in Figure 23 for all pairs of digits. (There will be $\binom{10}{2} = 55$ pairs. Use either convex optimization algorithms.)

(ii) By looking at the plot of the optimal regression coefficients, for each digit i, make a guess on digits j that would be the easiest and the hardest for the logistic regression model would distinguish.

3.2. Testing. Now let's discuss testing our trained logistic regression model for the MNIST dataset. We first need to discuss how we measure the accuracy of binary classification. Suppose we have target and predicted binary vectors $y_{\text{test}}, y_{\text{pred}} \in \{0,1\}^N$. For each test example i, there are four possibilities: $(y_{\text{test}}[i], y_{\text{pred}}[i]) \in \{(T,T),(T,F),(F,T),(F,F)\}$, where T stands for 'true' or 'positive' and F stands for 'false' or 'negative'. These four cases have the following specific names:

- *True Positive*: $(y_{\text{test}}[i], y_{\text{pred}}[i]) = (T,T)$
- *False Positive* $(y_{\text{test}}[i], y_{\text{pred}}[i]) = (F,T)$ (e.g., 'patient diagnosed to have cancer but does not')
- *False Negative* $(y_{\text{test}}[i], y_{\text{pred}}[i]) = (T,F)$ (e.g., 'patient diagnosed healthy but has cancer')
- *True Negative*$(y_{\text{test}}[i], y_{\text{pred}}[i]) = (F,F)$

One of the above instances hold for each test example i. The table in Figure 15 summaries various statistics associated to predicting y_{test} as y_{pred}.

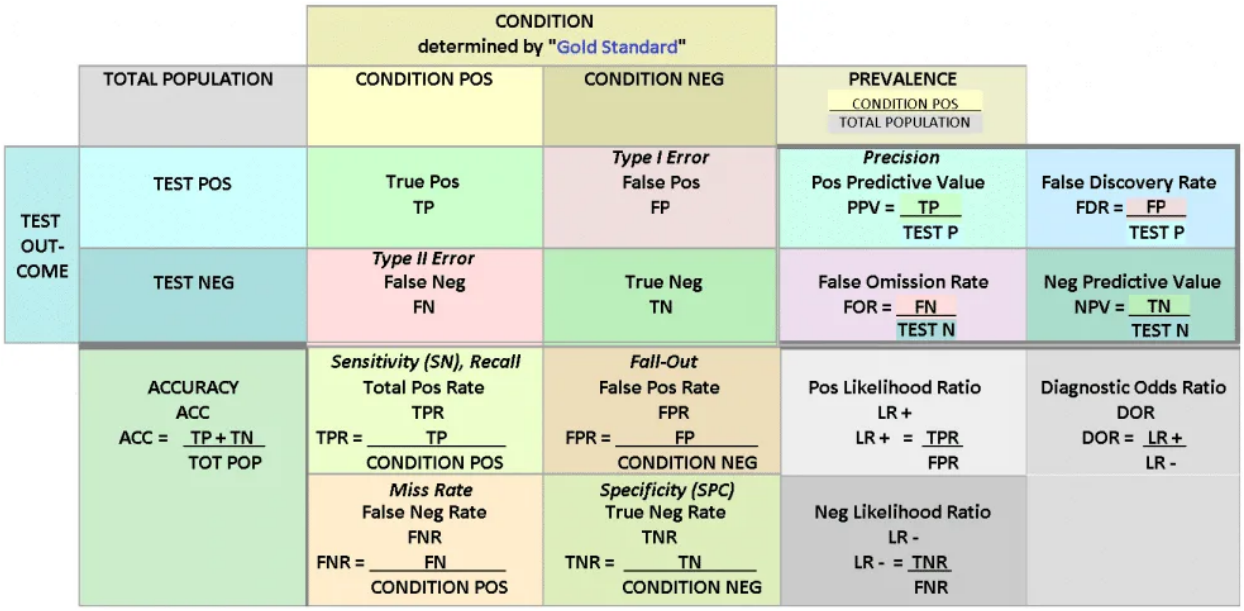


FIGURE 15. Various statistics of binary classification testing. Figure excerpted from link.

Here is a list of a few commonly used statistics:

- *Accuracy* $= \frac{\#\text{ predicted correctly}}{\text{size of the test set}}$ (e.g. $= \frac{\#\text{ predicted correctly as cancer or healthy}}{\#\text{ total subjects}}$)
- *Sensitivity* $= \frac{\#\text{ predicted positive correctly}}{\#\text{ positive test examples}}$ (e.g. $= \frac{\#\text{ predicted correctly as cancer}}{\#\text{ cancer patients}}$)
- *Specificity* $= \frac{\#\text{ predicted negative correctly}}{\#\text{ negative test examples}}$ (e.g. $= \frac{\#\text{ predicted correctly as healthy}}{\#\text{ healthy subjects}}$)
- *Precision* $= \frac{\#\text{ predicted positive correctly}}{\#\text{ predicted positive}}$ (e.g. $= \frac{\#\text{ predicted correctly as cancer}}{\#\text{ predicted as cancer}}$)

Next, before we proceed to the running example of LR on MNIST, we need to say a bit more on how we get the binary vector of predictions y_{pred}. Namely, recall that LR is a probabilistic model, where one models the output as a Bernoulli RV Y with success probability $p = \sigma(\boldsymbol{\phi}(\mathbf{x})^T\mathbf{w})$ as in (71). The result of fitting LR to the training set is the optimal parameter $\hat{\mathbf{w}}$ defined in (78), so we can compute the predictive probability $p = p(\mathbf{y}|\boldsymbol{\phi}(\mathbf{x}))$ for the unknown output. Suppose we have $p = 0.8$. This means that our prediction on the output has probability 0.8 to be 1 and 0.2 to be 0, so it would be reasonable to predict the output as 1. More precisely, for the current example, we use the following decision algorithm

$$\mathbf{x} \longmapsto \hat{\mathbf{y}} = \mathscr{A}(\mathbf{x}) = \begin{cases} 1 & \text{if } p = \sigma(\boldsymbol{\phi}(\mathbf{x})^T\hat{\mathbf{w}}) \geq 0.5 \\ 0 & \text{if } p = \sigma(\boldsymbol{\phi}(\mathbf{x})^T\hat{\mathbf{w}}) < 0.5. \end{cases} \tag{88}$$

Simply speaking, we are predicting $\hat{\mathbf{y}}$ to be more likely class label predicted by the predicted probability, with the classification threshold being 0.5[4]. This is an instance of the more general decision algorithm discussed in Exercise 1.3.2.

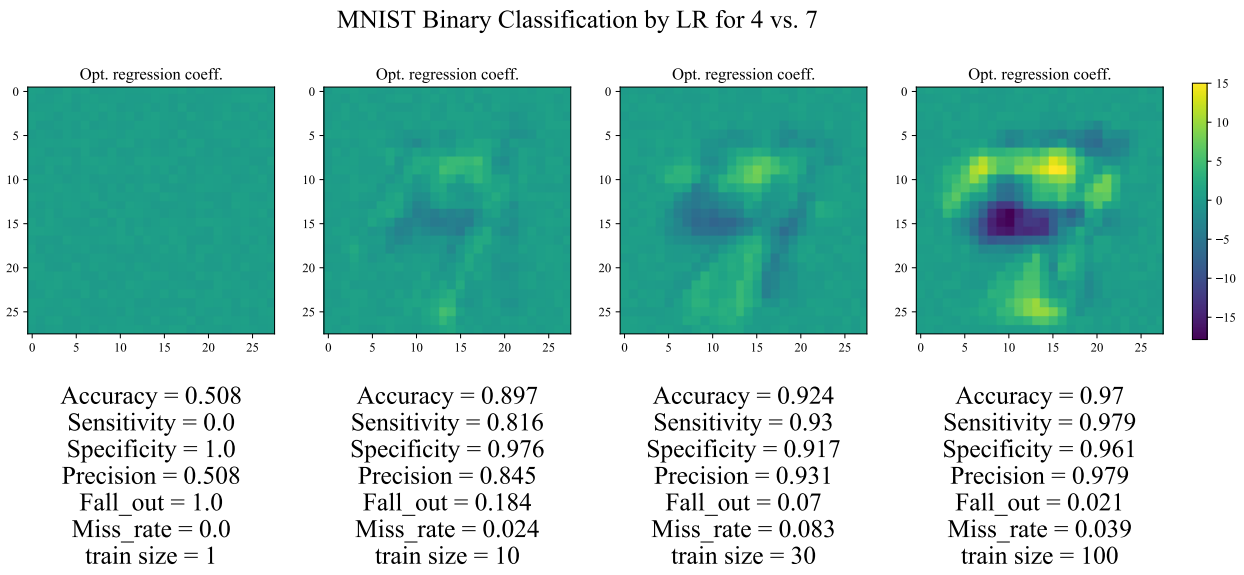


FIGURE 16. Examples of the training and testing logistic regression coefficients to the MNIST dataset with chosen two digits. Gradient descent algorithm is used for training. Training datasets of size in $\{1, 10, 30, 100\}$ are subsampled and used for fitting the model. Various statistics for binary classification are also shown.

Now we discuss testing LR on the MNIST dataset. MNIST is such a clean dataset so that we only need a few training examples to achieve very good classification accuracies. We test our LR classifier trained on randomly subsampled test sets of sizes 1, 10, 30, and 100 for classifying digits '4' vs. '7'. The test set contains about 20% of examples in the dataset, which is about 14,000 images of digits '4' and '7'. In Figure

[4]The use of threshold 0.5 is not necessary. For highligy imbalanced training datasets, the predicted probabilities could be skewed either to 0 or 1, in which case we would need to use some other threshold $\theta \in [0,1]$. The 'overall accuracy' of binary classification using arbitrary threshold θ is measured by other means, such as the *area-under-curve* (AUC) score for the corresponding *receiver operation characteristic* (ROC)

16, we have examples of trained regression coefficients (shown as a heat map on 28×28 image) as well as various statistics of binary classification we discussed before. Observe that as we increase the size of the train set, the optimal regression coefficients become more aware of the shapes of '7' as positive (yellow) and '4' as negative (blue), and the classification metrics improve in general. See Figure 17 for a similar example with digits '3' and '8' instead, which seems to be a bit more difficult case than distinguishing between '4' and '7'.

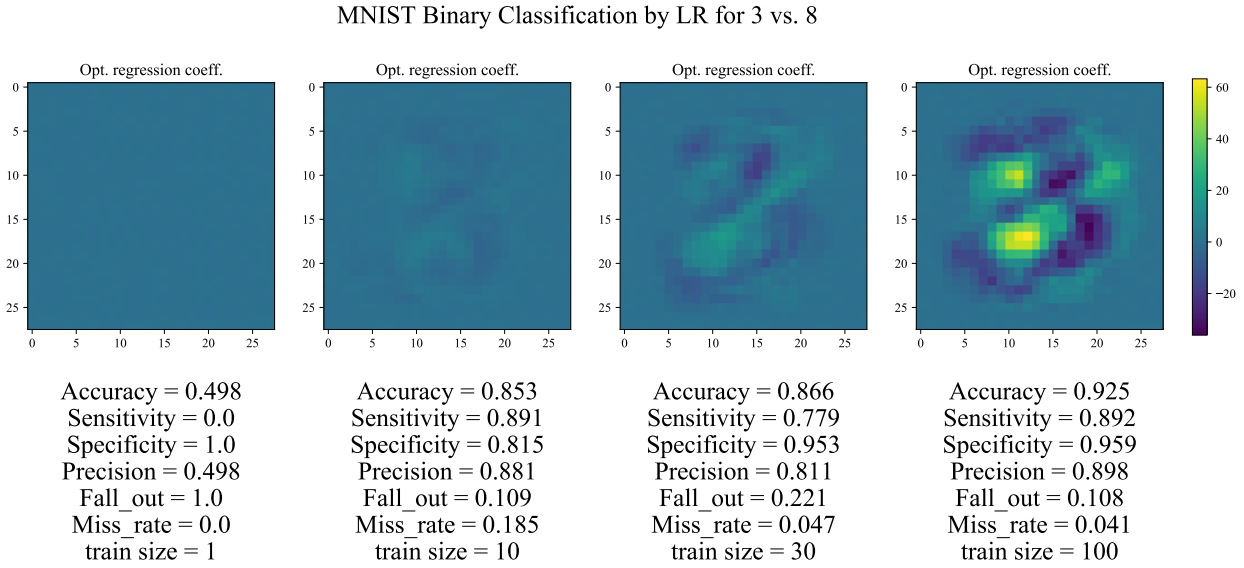


FIGURE 17. Examples of the training and testing logistic regression coefficients to the MNIST dataset with chosen two digits. Gradient descent algorithm is used for training. Training datasets of size in $\{1, 10, 30, 100\}$ are subsampled and used for fitting the model. Various statistics for binary classification are also shown.

3.3. Multiclass Logistic Regression. Next, we extend our logistic regression model to the multiclass classification, where the class labels now assume $\kappa \geq 2$ distinct values. This model is called the *multiclass logistic regression* (MLR). Without loss of generality, we can assume that the κ classes are the integers in $\{1, 2, \ldots, \kappa\}$. Say we have training examples $(\boldsymbol{\phi}(\mathbf{x}_1), y_1), \ldots, (\boldsymbol{\phi}(\mathbf{x}_N), y_N)$, where

- $\mathbf{x}_1, \ldots, \mathbf{x}_N$: Input data (e.g., collection of all medical records of each patient)
- $\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_N) \in \mathbb{R}^p$: Features (e.g., some useful (derived) information for each patient)
- $y_1, \ldots, y_n \in \{1, 2, \ldots, \kappa\}$: κ class labels (e.g., digits from 0 to 9).

The basic idea of multiclass logistic regression is to model the output y as a discrete random variable Y with probability mass function (PMF) $p = [p_1, \ldots, p_\kappa]$ that depends on the observed feature $\boldsymbol{\phi}(\mathbf{x})$. More specifically,

$$\mathbb{P}\left(Y = i \,|\, \boldsymbol{\phi}(\mathbf{x})\right) = p_i, \qquad p_i \propto \exp(\boldsymbol{\phi}(\mathbf{x})^T \mathbf{w}_i) \quad \text{for } i = 1, \ldots, \kappa. \tag{89}$$

More precisely, the predictive probability distribution $p = [p_1, \ldots, p_\kappa]$ can be written as

$$p_i = \frac{\exp(\boldsymbol{\phi}(\mathbf{x})^T \mathbf{w}_i)}{\sum_{j=1}^{\kappa} \exp(\boldsymbol{\phi}(\mathbf{x})^T \mathbf{w}_j)}, \qquad \text{for } i = 1, \ldots, \kappa. \tag{90}$$

A probability distribution of the form $p_i \propto \exp(a_i)$ is called the *softmax distribution* with *activation* $\mathbf{a} = [a_1, \ldots, a_\kappa]$. Hence in the case of multinomial LR, the predictive probability distribution $p = [p_1, \ldots, p_\kappa]$

is the softmax distribution with 'linear activation' $\boldsymbol{\phi}(\mathbf{x})^T\mathbf{w}_1 \ldots, \boldsymbol{\phi}(\mathbf{x})^T\mathbf{w}_\kappa$ of the feature vector $\boldsymbol{\phi}(\mathbf{x})$. Notice that this model has parameter vectors $\mathbf{w}_1, \ldots, \mathbf{w}_\kappa$, one for each of the κ class labels[5].

Next, let's discuss how to train MLR. We will use the following matrix notations

$$\mathbf{Y} = \begin{bmatrix} \mathbf{1}(y_1 = 1) & \cdots & \mathbf{1}(y_1 = \kappa) \\ \vdots & & \vdots \\ \mathbf{1}(y_N = 1) & \cdots & \mathbf{1}(y_N = \kappa) \end{bmatrix} \in \{0,1\}^{N\times\kappa}, \qquad \boldsymbol{\Phi} = \begin{bmatrix} \uparrow & & \uparrow \\ \boldsymbol{\phi}(\mathbf{x}_1) & \cdots & \boldsymbol{\phi}(\mathbf{x}_N) \\ \downarrow & & \downarrow \end{bmatrix} \in \mathbb{R}^{p\times N}, \tag{92}$$

$$\mathbf{W} = \begin{bmatrix} \uparrow & & \uparrow \\ \mathbf{w}_1 & \cdots & \mathbf{w}_\kappa \\ \downarrow & & \downarrow \end{bmatrix} \in \mathbb{R}^{p\times\kappa}. \tag{93}$$

Again since it is a probabilistic model, we use the method of maximum likelihood. For that, we compute the joint likelihood of observing the output values $(y_1, \ldots, y_N)$ from the tuple of independent RVs $(Y_1, \ldots, Y_N)$ with PMFs $p_{ij} \propto \exp(\boldsymbol{\phi}(\mathbf{x}_j)^T\mathbf{w}_j)$ where $p_{ij} = \mathbb{P}(Y_i = j)$:

$$L(y_1, \ldots, y_N; \mathbf{W}) = \mathbb{P}(Y_1 = y_1, \ldots, Y_N = y_N; \mathbf{W}) = \prod_{i=1}^{N}\prod_{j=1}^{\kappa}(p_{ij})^{\mathbf{1}(y_i=j)} \tag{94}$$

Hence the optimial parameter matrix $\hat{\mathbf{W}} = [\hat{\mathbf{w}}_1, \ldots, \hat{\mathbf{w}}_\kappa] \in \mathbb{R}^{p\times\kappa}$ by

$$\hat{\mathbf{W}} = \underset{\mathbf{W}\in\mathbb{R}^{p\times\kappa}}{\arg\max} \log L(y_1, \ldots, y_N; \mathbf{W}) \tag{95}$$

$$= \underset{\mathbf{W}\in\mathbb{R}^{p\times\kappa}}{\arg\max} \sum_{i=1}^{N}\sum_{j=1}^{\kappa}\mathbf{1}(y_i = j)\log p_{ij} \tag{96}$$

$$= \underset{\mathbf{W}\in\mathbb{R}^{p\times\kappa}}{\arg\min} \sum_{i=1}^{N}\sum_{j=1}^{\kappa}\mathbf{1}(y_i = j)\left[-\boldsymbol{\phi}(\mathbf{x}_i)^T\mathbf{w}_j + \log\left(\sum_{q=1}^{\kappa}\exp(\boldsymbol{\phi}(\mathbf{x}_i)^T\mathbf{w}_q)\right)\right] \tag{97}$$

$$= \underset{\mathbf{W}\in\mathbb{R}^{p\times\kappa}}{\arg\min} \left[-\sum_{i=1}^{N}\sum_{j=1}^{\kappa}\mathbf{1}(y_i = j)\boldsymbol{\phi}(\mathbf{x}_i)^T\mathbf{w}_j + \sum_{i=1}^{N}\log\left(\sum_{q=1}^{\kappa}\exp(\boldsymbol{\phi}(\mathbf{x}_i)^T\mathbf{w}_q)\right)\right], \tag{98}$$

where we have used the following observation

$$1 = \sum_{j=1}^{\kappa}\mathbf{1}(y_i = j) \quad \text{for all } i = 1, \ldots, N. \tag{99}$$

Using a matrix formulation, we have

$$\hat{\mathbf{W}} = \underset{\mathbf{W}\in\mathbb{R}^{p\times\kappa}}{\arg\min} \left[\ell_{MLR}(\mathbf{W}) := \left(\sum_{i=1}^{N}\log\left(\sum_{q=1}^{\kappa}\exp(\boldsymbol{\phi}(\mathbf{x}_i)^T\mathbf{w}_q)\right)\right) - \operatorname{tr}\left(\mathbf{Y}^T\boldsymbol{\Phi}^T\mathbf{W}\right)\right], \tag{100}$$

The loss function $\ell_{MLR}(\mathbf{w})$ in (78) is called the *multiclass logistic regression loss.*

In Exercise 2.3.2, we show that the above is a convex optimization problem and also compute the gradient and Hessian. As in the logistic regression case, we can solve Exercise 2.3.2 iteratively either by gradient descent or Newton-Ralphson. For gradient descent algorithm, we use

$$\mathbf{W}_{t+1} \longleftarrow \mathbf{W}_t - \eta_t\nabla_{\mathbf{W}}\ell_{MLR}(\mathbf{W}_t) = \mathbf{W}_t - \eta_t\boldsymbol{\Phi}(\mathbf{Q}(\mathbf{W}_t) - \mathbf{Y}) \in \mathbb{R}^{p\times\kappa}, \tag{101}$$

where we can use the same step sizes $\eta_t = \gamma(\log t)/\sqrt{t}$ for some $\gamma > 0$. The Newton-Ralphson algorithm (83) can also be implemented for the case of MLR by using the Hessian computed in Exercise 2.3.2.

[5] In fact, it is enough to determine $\kappa - 1$ p-dimensional vectors $\mathbf{w}_2 - \mathbf{w}_1, \ldots, \mathbf{w}_\kappa - \mathbf{w}_1$ since by dividing both the numerator and the denominator by $\exp(\boldsymbol{\phi}(\mathbf{x})^T\mathbf{w}_1)$, we can write the MLR predictive distribution in (90) as

$$p_i = \frac{\exp(\boldsymbol{\phi}(\mathbf{x})^T(\mathbf{w}_i - \mathbf{w}_1))}{\sum_{j=1}^{\kappa}\exp(\boldsymbol{\phi}(\mathbf{x})^T(\mathbf{w}_j - \mathbf{w}_1))}, \qquad \text{for } i = 1, \ldots, \kappa \tag{91}$$

Exercise 2.3.2 (Convexity of the multiclass logistic regression loss)**.** Let $\ell_{MLR}(\mathbf{W})$ denote the multiclass logistic regression loss defined in (100). Let $\mathbf{Q} = \mathbf{Q}(\mathbf{W}) = (p_{ij})_{i,j} \in \mathbb{R}^{N\times\kappa}$ where $p_{ij} \propto \exp(\boldsymbol{\phi}(\mathbf{x}_i)^T \mathbf{w}_j)$.

(i) Show that

$$\nabla_{\mathbf{W}} \ell_{MLR}(\mathbf{W}) = \boldsymbol{\Phi}(\mathbf{Q} - \mathbf{Y}) \in \mathbb{R}^{p\times\kappa}. \tag{102}$$

(ii) For each $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_\kappa] \in \mathbb{R}^{p\times\kappa}$, let $\mathbf{W}^{\text{vec}} := [\mathbf{w}_1^T, \ldots, \mathbf{w}_\kappa^T]^T \in \mathbb{R}^{p\kappa}$ denote its vectorization. Identify $\mathbf{W}$ with $\mathbf{W}^{\text{vec}}$ so that $\ell_{MLR} : \mathbb{R}^{p\kappa} \to \mathbb{R}$ is a function that maps a vector to a scalar. Then its Hessian $\mathbf{H} := \frac{\partial^2 \ell_{MLR}(\mathbf{W})}{\partial \mathbf{W} \partial \mathbf{W}^T}$ is a $p\kappa \times p\kappa$ matrix. Show that

$$\nabla_{\mathbf{W}} \ell_{MLR}(\mathbf{W}) = (\boldsymbol{\Phi}(\mathbf{Q} - \mathbf{Y}))^{\text{vec}} = \overbrace{\begin{bmatrix} \boldsymbol{\Phi} & O & O & \cdots & O \\ O & \boldsymbol{\Phi} & O & \cdots & O \\ \vdots & \vdots & & & \vdots \\ O & \cdots & O & \boldsymbol{\Phi} & O \\ O & O & \cdots & O & \boldsymbol{\Phi} \end{bmatrix}}^{\kappa \times \kappa \text{ block matrix}} (\mathbf{Q}^{\text{vec}} - \mathbf{Y}^{\text{vec}}) \in \mathbb{R}^{p\kappa}, \tag{103}$$

$$\mathbf{H} = \frac{\partial^2 \ell_{MLR}(\mathbf{W})}{\partial \mathbf{W} \partial \mathbf{W}^T} = \nabla_{\mathbf{W}} \mathbf{Q}^T \boldsymbol{\Phi}^T = \begin{bmatrix} \boldsymbol{\Phi}\mathbf{D}_1\boldsymbol{\Phi}^T & O & \cdots & O \\ O & \boldsymbol{\Phi}\mathbf{D}_2\boldsymbol{\Phi}^T & \cdots & O \\ \vdots & \vdots & & \vdots \\ O & \cdots & O & \boldsymbol{\Phi}\mathbf{D}_\kappa\boldsymbol{\Phi}^T \end{bmatrix} \in \mathbb{R}^{p\kappa\times p\kappa}, \tag{104}$$

where for $j = 1, \ldots, \kappa$, $\mathbf{D}_j$ is the $N \times N$ diagonal matrix with diagonal entries $\mathbf{D}_{ii} = p_{ij}(1 - p_{ij})$.

(iii) Argue that the Hessian $\mathbf{H}$ in **(ii)** is positive semi-definite. Conclude that the logistic regression loss ℓ_{LR} is convex. (Hint: Each $\mathbf{D}_j$ is a diagonal matrix with positive diagonal entries.)

4. Classification of MNIST by Multiclass Logistic Regression

4.1. Training. Recall that the MNIST dataset (see Figure 10) consists of total 70000 black-and-white images of size 28×28 pixels, corresponding to one of the 10 digits from $\{0, 1, \ldots, 9\}$. In this subsection, we will apply multiclass logistic regression to classify the MNIST images of for more then two digits.

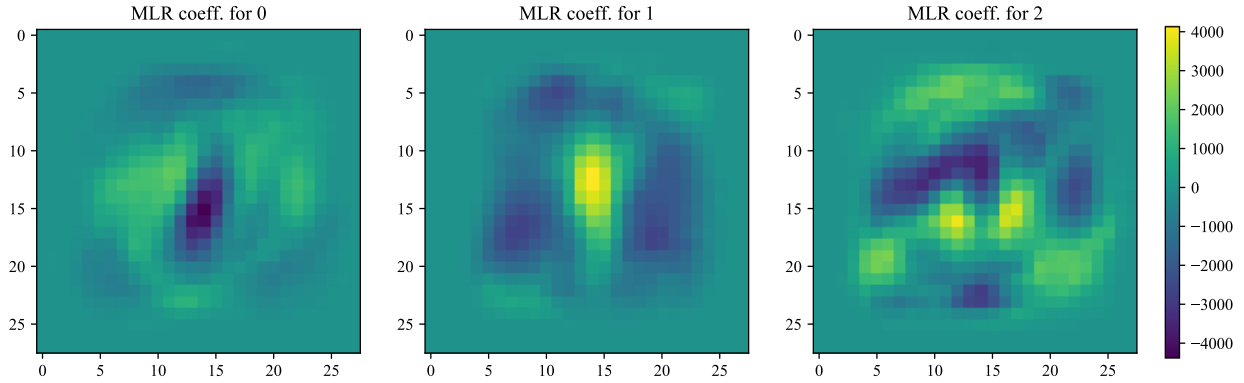


FIGURE 18. Examples of the fitted logistic regression coefficients to the MNIST dataset with digits in $['0', '1', '2']$. Gradient descent algorithm is used for training.

For instance, suppose we choose the images for digits '0', '1', and '2'. There are total 6903 , 7877, and 6990 images of digits '0', '1', and '2', respectively. Viewing each image as a matrix in $\mathbb{R}^{28\times28}$, the 3-class classification problem now is to learn a decision algorithm $\mathscr{A} : \mathbb{R}^{28\times28} \supseteq \{\text{Images of '0' or '1' or '2'}\} \to \{0, 1, 2\}$, where class label i means the image is classified as digit 'i' for $i \in \{0, 1, 2\}$. The goal is to learn the optimal logistic regression parameter $\hat{\mathbf{W}} \in \mathbb{R}^{28\times28\times3}$ by solving the optimization problem (100). The

meaning of the (i,j,k) entry $\hat{\mathbf{W}}[i,j,k]$ is the positive association of the (i,j) pixel of all 28×28 images in the train set with class label k.

The first subplot in Figure 18 displays $\hat{\mathbf{W}}[:,:,0]$ learned by the gradient descent algorithm in (101) with $\gamma = 1$. One can see that the pixels lying on the overall shape of digit '0' have large (yellow) regression coefficients, meaning that they contribute to the image being classified as '0' with a large probability. On the other hand, one can also see that the pixels lying on the overall shape of digits '1' or '2' have small regression coefficients, meaning that they discount the probability of the image being classified as '0'. The other subplots show $\hat{\mathbf{W}}[:,:,1]$ and $\hat{\mathbf{W}}[:,:,2]$.

4.2. Testing. Here we discuss testing our trained multiclass logistic regression model for the MNIST dataset. Suppose we have target and predicted vectors $y_{\text{test}}, y_{\text{pred}} \in \{1,2,\dots,\kappa\}^N$. We measure the overall accuracy of this prediction by the *confusion matrix*, which is a $\kappa\times\kappa$ matrix whose (i,j) entry means the number of instances of true class i classified as class j. Note that in the ideal situation of perfect prediction $y_{\text{pred}} = y_{\text{test}}$, the confusion matrix will be N times the $\kappa\times\kappa$ identity matrix.

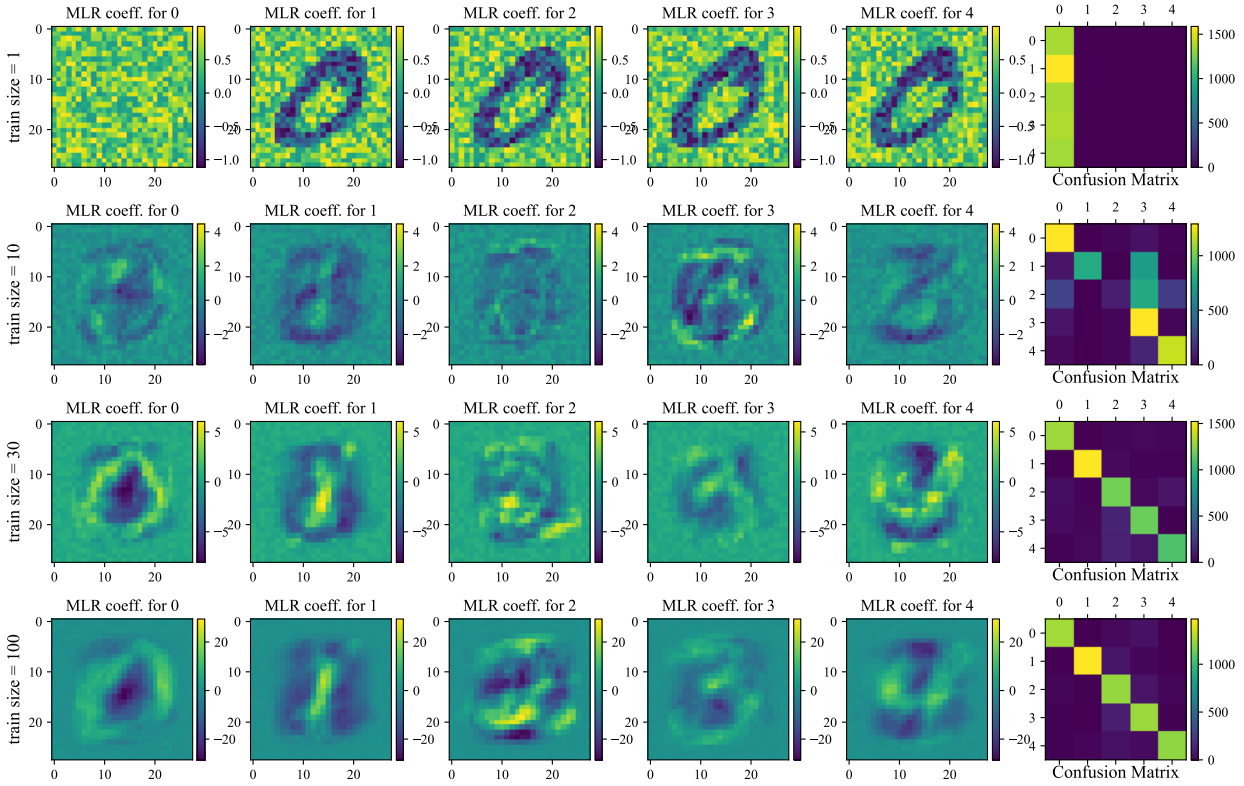


FIGURE 19. Examples of the fitted logistic regression coefficients to the MNIST dataset with digits in $['0','1','2','3','4']$. Gradient descent algorithm is used for training.

Next, recall that MLR is a probabilistic model, where one models the output as a discrete RV Y with PMF $p = [p_1,\dots,p_\kappa]$, $p_i \propto \exp(\boldsymbol{\phi}(\mathbf{x})^T\mathbf{w}_i)$ as in (89). The result of fitting MLR to the training set is the optimal parameter $\hat{\mathbf{W}} = [\hat{\mathbf{w}}_1,\dots,\hat{\mathbf{w}}_\kappa]$ defined in (100), so we can compute the predictive PMF $p = p(\mathbf{y}|\boldsymbol{\phi}(\mathbf{x}))$ for the unknown output. We will use the following *maximum a posteriori* decision algorithm

$$\mathbf{x} \longmapsto \hat{\mathbf{y}} = \mathscr{A}(\mathbf{x}) = \operatorname*{argmax}_{i\in\{1,2,\dots,\kappa\}} p(i\,|\,\boldsymbol{\phi}(\mathbf{x})) \tag{105}$$

$$= \operatorname*{argmax}_{i\in\{1,2,\dots,\kappa\}} \exp(\boldsymbol{\phi}(\mathbf{x})^T\hat{\mathbf{w}}_i) \tag{106}$$

$$= \underset{i\in\{1,2,\ldots,K\}}{\operatorname{argmax}} \boldsymbol{\phi}(\mathbf{x})^T \hat{\mathbf{w}}_i. \tag{107}$$

Simply speaking, we are predicting $\hat{\mathbf{y}}$ to be more likely class label predicted by the predicted PMF.

In Figure 19, we test our MLR classifier trained on randomly subsampled test sets of sizes 1, 10, 30, and 100 for classifying digits between '0', '1', ..., '4'. The test set contains about 20% of examples in the dataset, which is about 35,000 images of the five digits. In Figure 19, we have examples of trained regression coefficients (shown as a heat map on 28×28 image) as well as the confusion matrices. Observe that as we increase the size of the train set, the optimal regression coefficients become more aware of the shapes of the correct digits, and the confusion matrix becomes closer to a diagonal matrix, meaning the the algorithm gets less 'confused'.

5. Probit Regression

5.1. Model Formulation. In this section, we discuss a probabilistic regression (PR) model bor binary classification similar to logistic regression. The setting is identical, but the 'link function' is not the sigmoid function but something called 'probit' (probability + unit) function. Probit regression is widely used in Economics.

As before, we assume that the two classes are 0 and 1. Let $(\boldsymbol{\phi}(\mathbf{x}_1), y_1), \ldots, (\boldsymbol{\phi}(\mathbf{x}_N), y_N)$ be the training examples, where

- $\mathbf{x}_1, \ldots, \mathbf{x}_N$: Input data (e.g., collection of all medical records of each patient)
- $\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_N) \in \mathbb{R}^p$: Features (e.g., some useful (derived) information for each patient)
- $y_1, \ldots, y_n \in \{0, 1\}$: Binary class labels (e.g., '`normal`' or '`pneumonia`').

The basic idea of probit regression is to model the output y as a Bernoulli (0-1) random variable Y that takes value 1 iff the linear activation $\boldsymbol{\phi}(x)^T\mathbf{w}$ is at leaset a threshold θ, where $\mathbf{w} \in \mathbb{R}^p$ is a p-dimensional parameter vector. The point here is to randomize the θ as an independent Gaussian RV. More specifically,

$$Y \mid \boldsymbol{\phi}(\mathbf{x}) \sim \text{Bernoulli}(p), \qquad p = \mathbb{P}\left(\boldsymbol{\phi}(\mathbf{x})^T\mathbf{w} \geq Z\right) =: \Psi(\boldsymbol{\phi}(\mathbf{x})^T\mathbf{w}), \tag{108}$$

where $Z \sim N(0,1)$ is an independent standard Gaussian RV. We can also write the probit model more compactly as

$$Y \mid \boldsymbol{\phi}(\mathbf{x}) \sim \mathbf{1}\left(\boldsymbol{\phi}(\mathbf{x})^T\mathbf{w} \geq Z\right). \tag{109}$$

Here, the function $\Psi : x \mapsto \mathbb{P}(x \geq Z) = \mathbb{P}(Z \leq x)$ is called the *probit* function, which is exactly the CDF of standard Gaussian distribution. Simmilarly as sigmoid function does, the probit function also maps the real line onto the interval $[0,1]$. As in Figure 20, we see the probit link maps real numbers to the extreme probabilities 0 and 1 more aggressively than the sigmoid link.

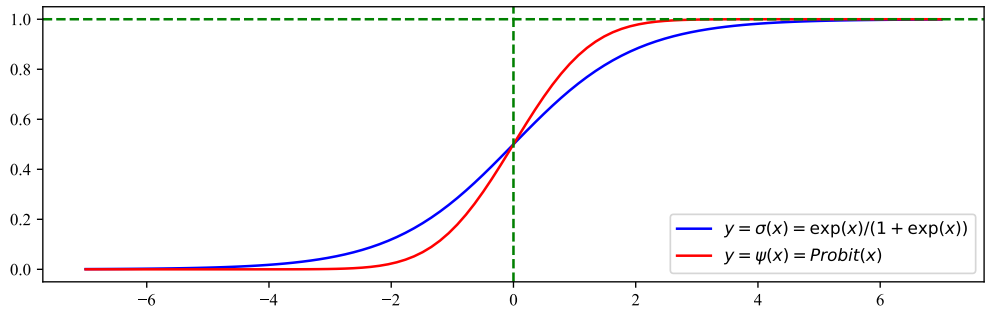


FIGURE 20. Plot of probit and sigmoid function.

Next, let's discuss how to train probit regression model. As before, we will use the method of maximum likelihood. For that, we compute the joint likelihood of observing the output values $(y_1, \ldots, y_N)$

from the tuple of independent Bernoulli variables $(Y_1,\dots,Y_N)$ with success probabilities $p_1 := \Psi(\boldsymbol{\phi}(\mathbf{x}_1)^T\mathbf{w})$, $\dots$, $p_N := \Psi(\boldsymbol{\phi}(\mathbf{x}_N)^T\mathbf{w})$:

$$L(y_1,\dots,y_N;\mathbf{w}) = \mathbb{P}(Y_1 = y_1,\dots,Y_N = y_N;\mathbf{w}) = \prod_{i=1}^{N} p_i^{y_i}(1-p_i)^{1-y_i}. \tag{110}$$

Hence the optimial parameter $\hat{\mathbf{w}} \in \mathbb{R}^p$ is given by

$$\begin{aligned}
\hat{\mathbf{w}} &= \operatorname*{arg\,max}_{\mathbf{w}\in\mathbb{R}^p} \log L(y_1,\dots,y_N;\mathbf{w}) && (111)\\
&= \operatorname*{arg\,max}_{\mathbf{w}\in\mathbb{R}^p} \sum_{i=1}^{N} y_i \log p_i + (1-y_i)\log(1-p_i) && (112)\\
&= \operatorname*{arg\,max}_{\mathbf{w}\in\mathbb{R}^p} \left[\sum_{i=1}^{N} y_i \log \Psi(\boldsymbol{\phi}(\mathbf{x}_i)^T\mathbf{w}) + (1-y_i)\log(1-\Psi(\boldsymbol{\phi}(\mathbf{x}_i)^T\mathbf{w}))\right] && (113)\\
&= \operatorname*{arg\,min}_{\mathbf{w}\in\mathbb{R}^p} \left[\ell_{PR}(\mathbf{w}) := -\sum_{i=1}^{N} y_i \log \Psi(\boldsymbol{\phi}(\mathbf{x}_i)^T\mathbf{w}) + (1-y_i)\log \Psi(-\boldsymbol{\phi}(\mathbf{x}_i)^T\mathbf{w})\right], && (114)
\end{aligned}$$

where we have use the property $\Psi(-x) = 1-\Psi(x)$ of standard normal CDF. The loss function $\ell_{PR}(\mathbf{w})$ in (111) is called the *probit regression loss*.

In Exercise 2.5.1, we show that the above is a convex optimization problem and also compute the gradient and Hessian. As in the logistic regression case, we can solve Exercise 2.5.1 iteratively either by gradient descent or Newton-Ralphson. For gradient descent algorithm, we use

$$\mathbf{w}_{t+1} \longleftarrow \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} \ell_{PR}(\mathbf{w}_t) \in \mathbb{R}^p, \tag{115}$$

where we can use the same step sizes $\eta_t = \gamma(\log t)/\sqrt{t}$ for some $\gamma > 0$. The gradient $\nabla_{\mathbf{w}}\ell_{PR}(\mathbf{w}_t)$ of the probit loss is computed in Exercise 2.5.1. The Newton-Ralphson algorithm can also be implemented for the case of probit regression similarly by using the Hessian computed in Exercise 2.5.1.

Exercise 2.5.1 (Convexity of the probit regression loss)**.** Let $\ell_{PR}(\mathbf{w})$ denote the logistic regression loss defined in (111). Let $\psi(x) = (2\pi)^{-1/2}\exp(-x^2/2)$ denote the PDF of standard normal distribution and let $\Psi(x) = \int_{-\infty}^{x}\psi(t)\,dt$ denote the corresponding CDF. Let $\boldsymbol{\Phi} \in \mathbb{R}^{p\times N}$ is the design matrix defined in (79).

(i) Show that

$$\begin{aligned}
\nabla_{\mathbf{w}}\ell_{PR}(\mathbf{w}) &= \sum_{i=1}^{N} \psi(\boldsymbol{\phi}(\mathbf{x}_i)^T\mathbf{w}) \left[\frac{1-y_i}{\Psi(-\boldsymbol{\phi}(\mathbf{x}_i)^T\mathbf{w})} - \frac{y_i}{\Psi(\boldsymbol{\phi}(\mathbf{x}_i)^T\mathbf{w})}\right]\boldsymbol{\phi}(\mathbf{x}_i) && (116)\\
&= \boldsymbol{\Phi}(\mathbf{Q}_1 - \mathbf{Q}_2), && (117)
\end{aligned}$$

where

$$\mathbf{Q}_1 = \left[\frac{(1-y_1)\psi(\boldsymbol{\phi}(\mathbf{x}_1)^T\mathbf{w})}{\Psi(-\boldsymbol{\phi}(\mathbf{x}_1)^T\mathbf{w})} \quad \cdots \quad \frac{(1-y_N)\psi(\boldsymbol{\phi}(\mathbf{x}_N)^T\mathbf{w})}{\Psi(-\boldsymbol{\phi}(\mathbf{x}_N)^T\mathbf{w})}\right]^T, \quad \mathbf{Q}_2 = \left[\frac{y_1\psi(\boldsymbol{\phi}(\mathbf{x}_1)^T\mathbf{w})}{\Psi(\boldsymbol{\phi}(\mathbf{x}_1)^T\mathbf{w})} \quad \cdots \quad \frac{y_N\psi(\boldsymbol{\phi}(\mathbf{x}_N)^T\mathbf{w})}{\Psi(\boldsymbol{\phi}(\mathbf{x}_N)^T\mathbf{w})}\right]^T \in \mathbb{R}^N. \tag{118}$$

(ii) Show the following properties of PDF and CDF of standard normal distribution:

$$\begin{aligned}
&\psi(x) = \psi(-x), \qquad \Psi(-x) = 1-\Psi(x), \qquad \psi'(x) = -x\psi(x), && (119)\\
&\frac{\partial}{\partial x}\frac{\psi(x)}{\Psi(x)} = \frac{\psi(x)\left[-x\Psi(x)-\psi(x)\right]}{\Psi(x)^2}, \qquad \frac{\partial}{\partial x}\frac{\psi(x)}{\Psi(-x)} = \frac{\psi(x)\left[-x\Psi(-x)+\psi(x)\right]}{\Psi(-x)^2}. && (120)
\end{aligned}$$

(iii) Show that

$$\frac{\partial(\mathbf{Q}_1^T - \mathbf{Q}_2^T)}{\partial \mathbf{w}} = \begin{bmatrix} \uparrow & & \uparrow \\ G_1\boldsymbol{\phi}(\mathbf{x}_1) & \cdots & G_N\boldsymbol{\phi}(\mathbf{x}_N) \\ \downarrow & & \downarrow \end{bmatrix} = \boldsymbol{\Phi}\mathbf{D}, \tag{121}$$

where for $z_i := \boldsymbol{\phi}(\mathbf{x}_i)^T\mathbf{w}$, we denote

$$G_i := (1-y_i)\frac{\psi(z_i)\left[-z_i\Psi(-z_i)+\psi(z_i)\right]}{\Psi(-z_i)^2} + y_i\frac{\psi(z_i)\left[z_i\Psi(z_i)+\psi(z_i)\right]}{\Psi(z_i)^2}. \tag{122}$$

and $\mathbf{D}$ is the $N\times N$ diagonal matrix with diagonal entries $\mathbf{D}_{ii} = G_i$.

(iv) Show that

$$\mathbf{H} := \frac{\partial^2 \ell_{PR}(\mathbf{w})}{\partial\mathbf{w}\partial\mathbf{w}^T} = \nabla_{\mathbf{w}}(\mathbf{Q}_1^T - \mathbf{Q}_2^T)\boldsymbol{\Phi}^T = \boldsymbol{\Phi}\mathbf{D}\boldsymbol{\Phi}^T \in \mathbb{R}^{p\times p}. \tag{123}$$

(v) (Optional*) Argue that the Hessian $\mathbf{H}$ is positive semi-definite. Conclude that the probit regression loss ℓ_{PR} is convex. (Hint: $\mathbf{D}$ is a diagonal matrix with positive diagonal entries. This was clear for logistic regression. Here, one can show the funtions $-z\Psi(-z)+\psi(z)$ and $-z\Psi(z)+\psi(z)$ always assume positive values by some calculus argument. See [ZL12, Lem. 1])

6. Binary classification of MNIST by Probit Regression

We revisit the MNIST data with probit regression. In this subsection, we will apply probit regression to classify the MNIST images of chosen two digits. The first subplot in Figure 21 displays $\hat{\mathbf{w}}$ for digits '0' and '1', learned by the gradient descent algorithm in (115) with $\gamma = 1$. A similar discussion for LR we made below Figure 23 holds as well in the PR case.

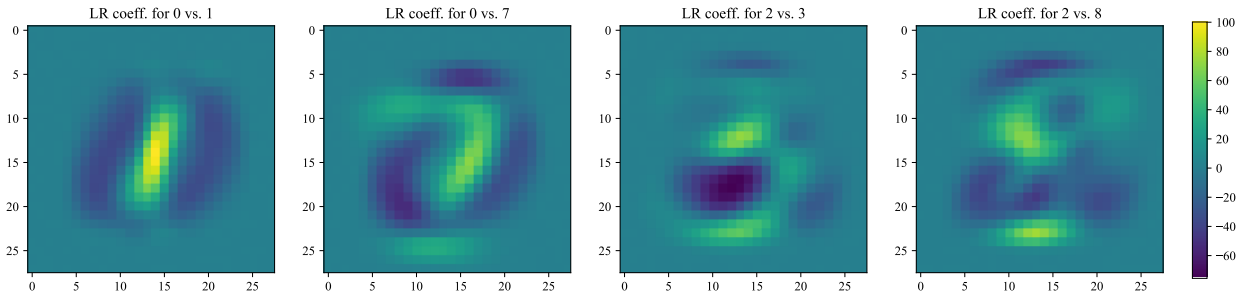


FIGURE 21. Examples of the fitted probit regression coefficients to the MNIST dataset with chosen two digits. Gradient descent algorithm is used for training.

Next, we discuss testing probit regression for binarty classification of MNIST dataset. Recall that probit regression is a probabilistic model, where one models the output as a Bernoulli RV Y with success probability $p = \Psi(\boldsymbol{\phi}(\mathbf{x})^T\mathbf{w})$ as in (108). The result of fitting PR to the training set is the optimal parameter $\hat{\mathbf{w}}$ defined in (111), so we can compute the predictive probability $p = p(\mathbf{y}|\boldsymbol{\phi}(\mathbf{x}))$ for the unknown output. For the current example, we use the following decision algorithm

$$\mathbf{x} \longmapsto \hat{\mathbf{y}} = \mathscr{A}(\mathbf{x}) = \begin{cases} 1 & \text{if } p = \Psi(\boldsymbol{\phi}(\mathbf{x})^T\hat{\mathbf{w}}) \geq 0.5 \\ 0 & \text{if } p = \Psi(\boldsymbol{\phi}(\mathbf{x})^T\hat{\mathbf{w}}) < 0.5. \end{cases} \tag{124}$$

Namely, we are predicting $\hat{\mathbf{y}}$ to be more likely class label predicted by the predicted probability, with the classification threshold being 0.5.

In Figure 16, we show experiments on teseting PR on the MNIST dataset similar to LR in Figure 16. As berfore, we see that as we increase the size of the train set, the optimal regression coefficients become more aware of the shapes of '7' as positive (yellow) and '4' as negative (blue), and the classification metrics improve in general.

Below, we investigate the gradient descent algorithm (115) in a bit more detail as we did for LR in Exercise 2.6.1. We also compare the performance of LR and PR on MNIST dataset with varying degree of noise in the data.

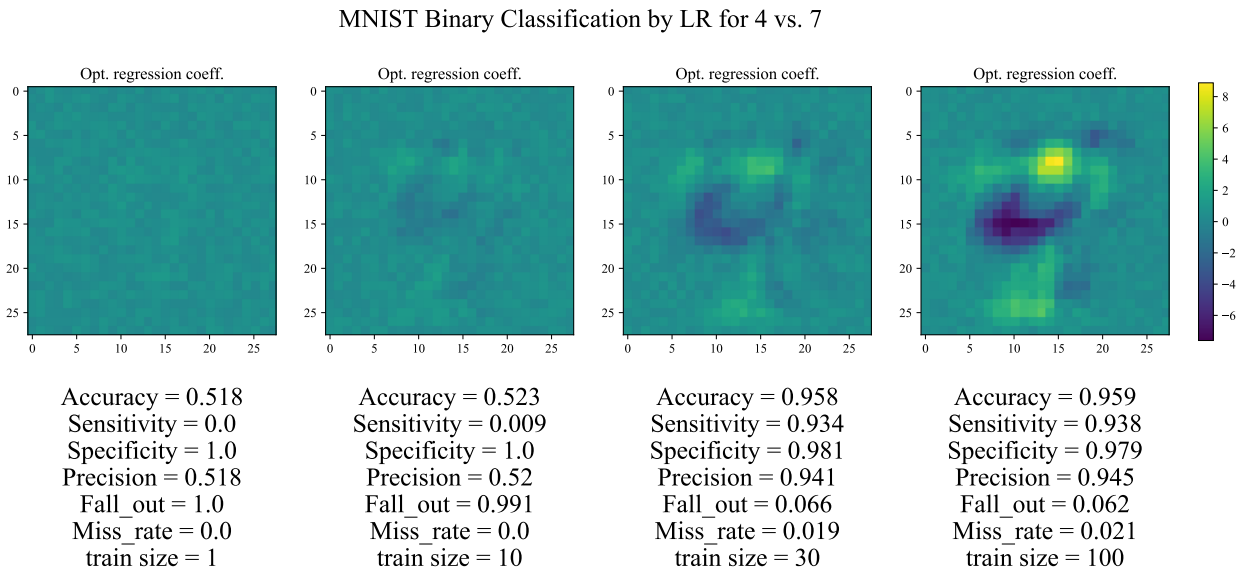


FIGURE 22. Examples of the training and testing probit regression coefficients to the MNIST dataset with chosen two digits. Gradient descent algorithm is used for training. Training datasets of size in $\{1, 10, 30, 100\}$ are subsampled and used for fitting the model. Various statistics for binary classification are also shown.

Exercise 2.6.1. The gradient descent (GD) algorithm for probit regression training in (82) is implemented as the python function 'fit_PR_GD' in the Jupyter notebook provided in the course repository.

```
def fit_PR_GD(Y, H, W0=None, sub_iter=100, stopping_diff=0.01):
        '''
        Convex optimization algorithm for Probit Regression using Gradient Descent
        Y = (n x 1), H = (p x n) (\Phi in lecture note), W = (p x 1)
        Logistic Regression: Y ~ Bernoulli(Q), Q = Probit(H.T @ W)
        '''
        if W0 is None:
            W0 = np.random.rand(H.shape[0],1) #If initial coefficients W0 is None, randomly initialize

        W1 = W0.copy()
        i = 0
        grad = np.ones(W0.shape)
        while (i < sub_iter) and (np.linalg.norm(grad) > stopping_diff):
            Q = norm.pdf(H.T @ W1) * ( (1-Y)/norm.cdf(-H.T @ W1) - Y/norm.cdf(H.T @ W1) )
            grad = H @ Q
            W1 = W1 - (np.log(i+1) / (((i + 1) ** (0.5)))) * grad
            i = i + 1
            # print('iter %i, grad_norm %f' %(i, np.linalg.norm(grad)))
        return W1
```

FIGURE 23. A python implementation of the gradient descent algorithm for logistic regression training (115).

(i) Modify the code (or make your own function) so that each gradient descent step (115), the current value of objective function ℓ_{PR} in (111) is printed. Test it on the MNIST example with digits $['0','1']$. Does the loss monotonically decrease?

(ii) The implemented version of GD for (115) uses the step size $\eta_t = \gamma \log(t+1)/\sqrt{(t+1)}$ with $\gamma = 1$. Test the same step sizes for $\gamma = 10$ and also $\eta_t = 1$ and $\eta_t = 1/(t+1)$. Test it on the MNIST example with digits $['0','1']$. Is there any difference in the way the loss function decrease?

(iii) If we use the step size of the form $\eta_t = \gamma t^{-\delta}$ for $0 \le \delta \le 1$, what would be your optimal choice of the hyperparameters γ and δ? (There is no answer and trying out a few choices and make your observation.)

Exercise 2.6.2. In this exercise, we investigate the 'robustness' of the logistic and the probit regression for binary classification.

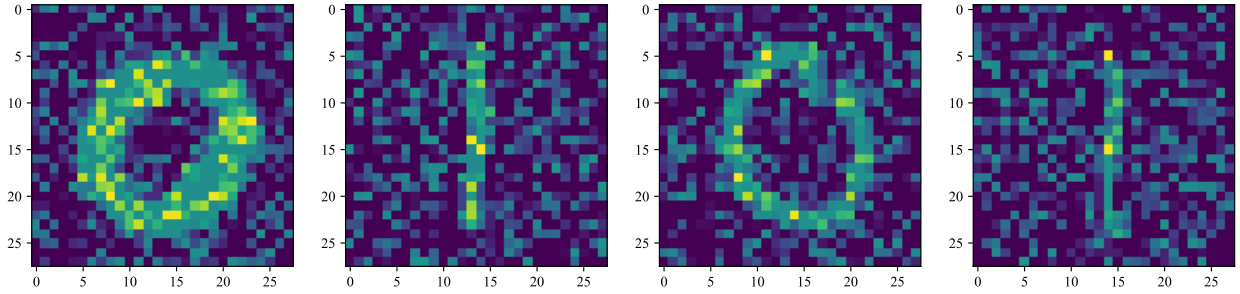


FIGURE 24. Corrupted MNIST images where Uniform([0,1]) is added to 50% of randomly chosen pixels.

(i) Modify the data preprocessing function "sample_binary_MNIST" so that it has the option to add noise with given ratio. Namely, if 'noise_ratio' is 0.5, then it will add independent Uniform([0,1]) variables to 50% of randomly chosen pixels. (See Figure 24)

(ii) For logistic regression, re-generate figures similar to Figure 16 with varying noise ratios in $\{0.1, 0.5, 0.9\}$.

(iii) For probit regression, re-generate figures similar to Figure 22 with varying noise ratios in $\{0.1, 0.5, 0.9\}$.

(iv) From the previous experiments, compare the robustness of LR and PR against random noise you gave. Can you conclude that one is significantly more robust than the other?

7. Naive Bayes Classifier

In this section, we discuss a class of classification algorithms called *Naive Bayes classifier*. Recall that in logistic (or probit) regression, we modeled the output class Y given the feature vector $\boldsymbol{\phi}(\mathbf{x})$ directly as a RV. In other words, we modeled the *predictive probability* $p(\text{class}\,|\,\text{features})$. However, Naive Bayes approach, we instead model the likelihood $p(\text{features}\,|\,\text{class})$, which is also called the *class-conditional probability* in the current situation. This change of perspective is possible by using Bayes' Theorem, and the 'Naive Bayes' assumption is used to simplify the computation of class-conditional probabilities.

7.1. The model formulation. We discuss the basic formulation of Naive Bayes Classifier in the setting of multiclass classification. We assume that there are κ class labels in $\{1,2,\dots,\kappa\}$. Say we have training examples $(\boldsymbol{\phi}(\mathbf{x}_1), y_1), \dots, (\boldsymbol{\phi}(\mathbf{x}_N), y_N)$, where

- $\mathbf{x}_1, \dots, \mathbf{x}_N$: Input data (e.g., collection of all medical records of each patient)
- $\boldsymbol{\phi}(\mathbf{x}_1), \dots, \boldsymbol{\phi}(\mathbf{x}_N) \in \mathbb{R}^p$: Features (e.g., some useful (derived) information for each patient)
- $y_1, \dots, y_n \in \{1,2,\dots,\kappa\}$: κ class labels (e.g., digits from 0 to 9).

As in the multiclass logistic regression, we model the output y as a discrete random variable Y with probability mass function (PMF) $[p_1, \dots, p_\kappa]$ that depends on the observed feature $\boldsymbol{\phi}(\mathbf{x})$:

$$\mathbb{P}\left(Y = i \,|\, \boldsymbol{\phi}(\mathbf{X})\right) = p_i \qquad \text{for } i = 1, \dots, \kappa. \tag{125}$$

Instead of modeling the predictive PMF p directly, we use Bayes' Theorem to rewrite the above as

$$\overbrace{\mathbb{P}\left(Y = i \,|\, \boldsymbol{\phi}(\mathbf{X}) = [\phi_1, \dots, \phi_m]\right)}^{\text{predictive prob.}} \propto \overbrace{\mathbb{P}\left(\boldsymbol{\phi}(\mathbf{X}) = [\phi_1, \dots, \phi_p] \,|\, Y = i\right)}^{\text{class-conditional prob.}} \overbrace{\mathbb{P}(Y = i)}^{\text{prior}}. \tag{126}$$

Now the main focus is to model the class-conditional probability. Namely, what is the probability that, given the class label Y is i, the observed feature vector $\phi(\mathbf{X}) = [\phi_1(\mathbf{X}), \dots, \phi_p(\mathbf{X})]$ equals $[\phi_1, \dots, \phi_p]$?

In order to properly model the p-dimensional random feature vector $\boldsymbol{\phi}(\mathbf{X}) \in \mathbb{R}^p$, we need to specify its p-dimensional class-conditional joint distribution. When the features $\phi_1, \dots, \phi_p$ assume discrete values,

then the joint distribution can be represented as a $p \times p$ contingency table, which has $(p-1)^2$ degrees of freedom to be specified. When p is large, such as quadratic dependence of model complexity is not desirable. In order to simplify the situation, we make the following 'naive Bayes assumption':

Naive Bayes Assumption: Given the class label $Y = i$, the features $\boldsymbol{\phi}(\mathbf{X}) = [\phi_1(\mathbf{X}), \dots, \phi_p(\mathbf{X})]$ are *independent*.

Clearly, such an assumption is not realistic and hence 'naive'. For instance, the events that a patient has weight > 75kg and age > 65 are not independent conditionally on having cancer. However, the naive Bayes assumption simplifies our model to linear complexity:

$$\mathbb{P}\left([\phi_1(\mathbf{X}), \dots, \phi_p(\mathbf{X})] = [\phi_1, \dots, \phi_p] \,\middle|\, Y = i\right) = \prod_{j=1}^{p} \mathbb{P}\left(\phi_j(\mathbf{X}) = \phi_j \mid Y = i\right). \tag{127}$$

Hence, we only need to model the marginal class-conditional probabilities $\mathbb{P}\left(\phi_j(\mathbf{X}) = \phi_j \mid Y = i\right)$ individually and then multiply all to get the joint class-conditional probability.

Suppose we use parameter $\mathbf{w}_{ij}$ for the marginal class-conditional probability $\mathbb{P}\left(\phi_j(\mathbf{X}) = \phi_j(\mathbf{x}_i) \mid Y = y_i\right)$ of jth feature given ith class. Then the joint likelihood of observing the class labels $y_1, \dots, y_N$ given the observed features $\boldsymbol{\phi}(\mathbf{x}_1), \dots, \boldsymbol{\phi}(\mathbf{x}_N)$ can be written as

$$L(y_1, \dots, y_N; \mathbf{W}) = \prod_{s=1}^{N} \overbrace{\mathbb{P}\left(Y_s = y_s \mid \boldsymbol{\phi}(\mathbf{X}) = \boldsymbol{\phi}(\mathbf{x}_s)\right)}^{\text{predictive prob.}} \tag{128}$$

$$\propto \prod_{s=1}^{N} \overbrace{\mathbb{P}\left(\boldsymbol{\phi}(\mathbf{X}) = \boldsymbol{\phi}(\mathbf{x}_s) \mid Y_s = y_s\right)}^{\text{class-conditional prob.}} \overbrace{\mathbb{P}(Y_s = y_s)}^{\text{prior}} \tag{129}$$

$$= \prod_{s=1}^{N} \prod_{i=1}^{\kappa} \left(\prod_{j=1}^{p} \mathbb{P}\left(\phi_j(\mathbf{X}) = \phi_j(\mathbf{x}_s) \,\middle|\, Y_s = i\right) \mathbb{P}(Y_s = i) \right)^{\mathbf{1}(y_s = i)}. \tag{130}$$

Hence the optimal paramter $\hat{\mathbf{W}} = (\hat{\mathbf{w}}_{ij})$ can be found by the standard maximum likelihood method:[6]

$$\hat{\mathbf{W}} = \underset{\mathbf{W}}{\arg\max} \left[\sum_{s=1}^{N} \sum_{i=1}^{\kappa} p\mathbf{1}(y_s = i) \log \mathbb{P}(Y_s = i) + \sum_{s=1}^{N} \sum_{j=1}^{p} \sum_{i=1}^{\kappa} \mathbf{1}(y_s = i) \log \underbrace{\mathbb{P}\left(\phi_j(\mathbf{X}) = \phi_j(\mathbf{x}_s) \,\middle|\, Y_s = i\right)}_{\text{parameterized by } \mathbf{w}_{i,j}} \right]. \tag{131}$$

In the following sections, we will solve the above optimization problem for some well-known cases of naive Bayes classifiers.

Lastly, given a new input $\mathbf{x}$ and the the optimal paramter $\hat{\mathbf{W}}$ for naive Bayes classifier found by (131) using the training set, we can make the prediction of the class label using the computed predictive probabilities. As before for the multiclass logistic regression (105), we can use the maximum a posteriori (MAP) decision rule:

$$\mathbf{x} \longmapsto \mathscr{A}(\mathbf{x}) = \underset{i \in \{1,2,\dots,\kappa\}}{\arg\max} \overbrace{\mathbb{P}\left(Y = i \mid \boldsymbol{\phi}(\mathbf{X}) = \boldsymbol{\phi}(\mathbf{x}), \hat{\mathbf{W}}\right)}^{\text{predictive prob.}} \tag{132}$$

$$= \underset{i \in \{1,2,\dots,\kappa\}}{\arg\max} \left[\sum_{i=1}^{N} \log \mathbb{P}(Y = i) + \sum_{i=1}^{N} \sum_{j=1}^{p} \log \underbrace{\mathbb{P}\left(\phi_j(\mathbf{X}) = \phi_j(\mathbf{x}_i) \mid Y = y_i\right)}_{\text{parameterized by } \hat{\mathbf{w}}_{y_i,j}} \right]. \tag{133}$$

[6] Here we treat prior as is given, but we also regard it as a parameter to be optimized.

7.2. Multinomial Naive Bayes. In this subsection, we discuss the case naive Bayes classifier with discrete features in the context of document classification. We will make the following version of naive Bayes assumption:

Naive Bayes Model for Document Classification:

- A set of 'vocabulary' $\mathcal{V} = \{a_1, \dots, a_M\}$ is given. (e.g., $\mathcal{V} = \{\text{Dear}, \text{Sir}, \text{Money}, \text{Friend}, \text{Thanks}\}$)
- Class lables $\mathscr{Y} = \{1, 2, \dots, \kappa\}$ (e.g., 0 ='normal', 1 ='spam')
- For each data point (document) $\mathbf{x}$, assume its feature $\boldsymbol{\phi}(\mathbf{x})$ is given by a frequency vector (histogram) $\boldsymbol{\phi}(\mathbf{x}) = [c_1, \dots, c_M]$, where c_i means the count of word a_i in $\mathbf{x}$.

$$\text{(e.g., } \mathbf{x} = \text{"Dear Sir, Money Money Money. Thanks."}, \boldsymbol{\phi}(\mathbf{x}) = [1,1,3,0,1]) \tag{134}$$

- Given the class label $Y = i$, each word in document $\mathbf{x}$ is sampled independently from the set $\mathcal{V}$ of vocabulary with probability q_{ij}.
- Use q_i for the prior probability of a random document being of class i.

$$\text{(e.g., } \quad \mathbb{P}\left(\boldsymbol{\phi}(\mathbf{X}) = [1,1,3,0,1] \,|\, Y = \text{spam}\right) \propto q_{11} \cdot q_{12} \cdot q_{13}^3 \cdot q_{14}^0 \cdot q_{15}). \tag{135}$$

- Let $\mathbf{W} = (q_i, q_{ij})$ denote the parameter vector.

Suppose we have training examples $(\boldsymbol{\phi}(\mathbf{x}_1), y_1), \dots, (\boldsymbol{\phi}(\mathbf{x}_N), y_N)$. For each $s \in \{1, \dots, N\}$ (data index), $i \in \{1, \dots, \kappa\}$ (class label), and $j \in \{1, \dots, M\}$ (word index), using the naive Bayes assumption above, we can write

$$\mathbb{P}\left(\phi_j(\mathbf{X}) = \phi_j(\mathbf{x}_s) \,\middle|\, Y = i\right) = \mathbb{P}(\text{seeing the } j\text{th word } a_j \in \mathcal{V} \ \phi(\mathbf{x}_s) \text{ times in } \mathbf{x}_s \,|\, Y = i) \tag{136}$$

$$\propto (q_{i,j})^{\phi_j(\mathbf{x}_s)}. \tag{137}$$

Hence we have

$$\hat{\mathbf{W}} = \underset{\mathbf{W}=(q_i, q_{ij})}{\operatorname{argmax}} \left[\sum_{s=1}^{N} \sum_{i=1}^{\kappa} M \mathbf{1}(y_s = i) \log q_i + \sum_{s=1}^{N} \sum_{j=1}^{M} \sum_{i=1}^{\kappa} \mathbf{1}(y_s = i) \phi_j(\mathbf{x}_s) \log(q_{i,j}) \right] \tag{138}$$

$$= \underset{\mathbf{W}=(q_i, q_{ij})}{\operatorname{argmax}} \left[M \sum_{i=1}^{\kappa} \underbrace{\left(\sum_{s=1}^{N} \mathbf{1}(y_s = i) \right)}_{c_i := \#\text{ of class } i \text{ examples}} \log q_i + \sum_{i=1}^{\kappa} \sum_{j=1}^{M} \underbrace{\left(\sum_{s=1}^{N} \mathbf{1}(y_s = i) \phi_j(\mathbf{x}_s) \right)}_{c_{i,j} := \#\text{ of } j\text{th word in class } i \text{ examples}} \log(q_{i,j}) \right], \tag{139}$$

where the parameters $\mathbf{W} = (q_i, q_{i,j})$ are subject to the constraint

$$0 \le q_1, \dots, q_\kappa \le 1, \qquad q_1 + \cdots + q_\kappa = 1 \tag{140}$$

$$0 \le q_{i,1}, \dots, q_{i,M} \le 1, \qquad q_{i,1} + \cdots + q_{i,M} = 1 \qquad \text{for } i = 0, 1, \dots, \kappa. \tag{141}$$

In the following proposition, we show that the optimal parameters that solve the above maximum likelihood formulation are given by simple formulas involding empirical frequencies of each class and of each words given each class.

Proposition 2.7.1. *The solution $\hat{\mathbf{W}} = (\hat{q}_i, \hat{q}_{i,j})$ to the optimization problem (138) is given by*

$$\hat{q}_i = \frac{1}{N} \sum_{s=1}^{N} \mathbf{1}(y_s = i), \qquad \hat{q}_{i,j} \propto \sum_{s=1}^{N} \mathbf{1}(y_s = i) \phi_j(\mathbf{x}_s) \quad \textit{for } j \in \{1, \dots, M\}. \tag{142}$$

PROOF. First, note that the second term in the objective function in (138) does not depend on the parameters $q_1, \dots, q_\kappa$. Hence $[\hat{q}_1, \dots, \hat{q}_\kappa]$ is a PMF on $\{1, \dots, \kappa\}$ that maximizes $\sum_{i=1}^{\kappa} c_i \log q_i$, where $c_i := \sum_{s=1}^{N} \mathbf{1}(y_s = i) \ge 0$. Hence the formula for $\hat{q}_i$'s follow from Exercise 2.7.2.

Next, fix a class label $i \in \{1, \dots, \kappa\}$. Observe that the only terms in the objective function in (138) that depends on $q_{i,1}, \dots, q_{i,M}$ is $\sum_{j=1}^{M} \left(\sum_{s=1}^{N} \mathbf{1}(y_s = i) \phi_j(\mathbf{x}_s) \right) \log(q_{i,j})$. Since $[q_{i,1}, \dots, q_{i,M}]$ is a PMF on $\{a_1, \dots, a_M\}$, again the formula for $\hat{q}_{i,j}$'s for $j = 1, \dots, M$ follow from Exercise 2.7.2. □

Exercise 2.7.2. In this exercise, we solve a simple optimization problem related to the multinomial naive Bayes classifier. Fix a finite set $\mathscr{X} = \{x_1, \ldots, x_M\}$ and real numbers $c_1, \cdots, c_M \geq 0$. Consider the following optimization problem

$$\hat{\mathbf{w}} = \underset{\mathbf{w}=[w_1,\ldots,w_M]^T \in \mathbb{R}^M}{\arg\max} \left(L(\mathbf{w}) := \sum_{i=1}^{M} c_i \log w_i \right) \quad \text{subject to: } \mathbf{w} \text{ is a PMF on } \mathscr{X}. \tag{143}$$

We will show that the solution is given by

$$\hat{\mathbf{w}} = \left[\frac{c_1}{\sum_{i=1}^M c_i}, \ldots, \frac{c_M}{\sum_{i=1}^M c_i} \right]^T. \tag{144}$$

(i)) The constraint set of (143) is the 'simplex' $\mathscr{C} = \{\mathbf{w} \in \mathbb{R}^M \,|\, 0 \leq w_1, \cdots, w_M \leq 1, \sum_{i=1}^M w_i = 1\}$. Denote the larger constraint set $\overline{\mathscr{C}} = \{\mathbf{w} \in \mathbb{R}^M \,|\, \sum_{i=1}^M w_i = 1\}$. Let $\lambda \in \mathbb{R}$ be a Lagrange multiplier for $\overline{\mathscr{C}}$. Then the Lagrangian is

$$g(\lambda, \mathbf{w}) = L(\mathbf{w}) - \lambda \left(\sum_{i=1}^{M} w_i - 1 \right). \tag{145}$$

Show that λ needs to satisfy

$$\frac{\partial g(\lambda, \mathbf{w})}{\partial \mathbf{w}} = \left[\frac{c_1}{w_1}, \ldots, \frac{c_M}{w_M} \right]^T - \lambda \, [1, \ldots, 1]^T = 0. \tag{146}$$

Conclude that (144) is the global maximum $L(\mathbf{w})$ over $\overline{\mathscr{C}}$.

(ii) From **(i)**, conclude that (144) is the global maximum $L(\mathbf{w})$ over $\mathscr{C}$.

Example 2.7.3 (Email Classification)**.** Suppose we have training examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{100}, y_{100})$ of normal and spam emails over the set of vocabulary $\mathscr{V} = \{\text{Dear}, \text{Sir}, \text{Money}, \text{Friend}, \text{Thanks}\}$. The class labels are from $\{0, 1\}$, where 0 is for normal and 1 is for spam emails. Let $c_{i,j}$ denote the total number of jth word in $\mathscr{V}$ that appear in all emails in the training set of class i. Suppose the training set gives the following statistics:

$$c_0 := \sum_{s=1}^{100} \mathbf{1}(y_s = \text{'0'}) = 70, \quad c_1 := \sum_{s=1}^{100} \mathbf{1}(y_s = \text{'1'}) = 30, \tag{147}$$

$$[c_{0,1}, c_{0,2}, c_{0,3}, c_{0,4}, c_{0,5}] = [70, 20, 15, 40, 90], \tag{148}$$

$$[c_{1,1}, c_{1,2}, c_{1,3}, c_{1,4}, c_{1,5}] = [73, 65, 98, 20, 120]. \tag{149}$$

Then according to Proposition 2.7.1, the optimal prior and class-conditional probabilities for naive Bayes classifier are given by

$$\text{prior PMF} = [\hat{q}_0, \hat{q}_1] = [0.7, 0.3], \tag{150}$$

$$\text{class-conditional PMF for normal emails} = [\hat{q}_{0,1}, \ldots, \hat{q}_{0,5}] = \frac{1}{235} [70, 20, 15, 40, 90], \tag{151}$$

$$\text{class-conditional PMF for spam emails} = [\hat{q}_{1,1}, \ldots, \hat{q}_{1,5}] = \frac{1}{376} [73, 65, 98, 20, 120]. \tag{152}$$

Now suppose we have a new email $\mathbf{x}$ = "Dear Sir, Money Money Money. Thanks.", so its bag-of-word feature vector is $\boldsymbol{\phi}(\mathbf{x}) = [1, 1, 3, 0, 1]$. In order to classify this email, we compute the predictive probabilities using the trained naive Bayes classifier as follow:

$$\mathbb{P}(Y = 0 \,|\, \boldsymbol{\phi}(\mathbf{X}) = [1, 1, 3, 0, 1]) \propto \frac{70 \cdot 20 \cdot 15^3 \cdot 40^0 \cdot 90}{235^6} \cdot (0.7)] = 1.7674 \times 10^{-6}, \tag{153}$$

$$\mathbb{P}(Y = 1 \,|\, \boldsymbol{\phi}(\mathbf{X}) = [1, 1, 3, 0, 1]) \propto \frac{73 \cdot 65 \cdot 98^3 \cdot 20^0 \cdot 120}{376^6} \cdot (0.3) = 5.6897 \times 10^{-5}. \tag{154}$$

It follows that

$$\mathbb{P}(Y=0\,|\,\boldsymbol{\phi}(\mathbf{X})=[1,1,3,0,1])=0.0301, \qquad \mathbb{P}(Y=1\,|\,\boldsymbol{\phi}(\mathbf{X})=[1,1,3,0,1])=0.9698. \tag{155}$$

According to the MAP decision rule in (132), we classify $\mathbf{x}$ as a spam email. ▲

Exercise 2.7.4. Consider the following training data: The goal is to predict whether new subjects with

age	income	student	credit_rating	buys_computer
≤ 30	high	no	fair	no
≤ 30	high	no	excellent	no
$[31,40]$	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	no	fair	yes
>40	low	yes	excellent	no
$[31,40]$	low	yes	excellent	yes
≤ 30	medium	yes	fair	yes
≤ 30	low	no	fair	yes
>40	medium	yes	fair	yes
≤ 30	medium	yes	excellent	yes
$[31,40]$	medium	yes	excellent	yes
$[31,40]$	high	no	fair	yes
>40	medium	no	excellent	no

given features (age, income, student, credit rating) would buy a computer, using naive Bayes classifier.

(i) Compute the maximum likelihood prior distribution on the label $\{0,1\}$, where $0=$'does not buy computer' and $1=$'buys computer'.

(ii) We will model the feature space as the following 10-dimensional product space of binary values $\{0,1\}$:[7]

$$\boldsymbol{\phi}(\mathbf{x})=\big[\mathbf{1}(\text{age}\leq 30),\,\mathbf{1}(\text{age}\in[31,40]),\,\mathbf{1}(\text{age}>40),\ldots,\mathbf{1}(\text{credit}=\text{ecellent})\big]\in\{0,1\}^{10}. \tag{156}$$

Compute the maximum likelihood class-conditional probabilities of each feature.

(iii) Suppose we have the following testing examples:

$$\mathbf{x}_1=\text{“ age}\leq 30\text{, medium income, student, fair credit rating”} \tag{157}$$

$$\mathbf{x}_2=\text{“ age}\in[31,40]\text{, low income, not student, fair credit rating”} \tag{158}$$

$$\mathbf{x}_3=\text{“ age}>40\text{, high income, not student, excellent credit rating”} \tag{159}$$

Compute the predictive probabilities (posterior distribution of class labels) of each testing examples. Make the prediction. For each example, can you tell which factor affected the most for the classification result?

(iv) Write a python script (in Jupyter notebook) that implements your computation automatically. Test your code on a similar but larger (≥ 100 training examples and ≥ 20 testing examples) dataset of your choice.

Exercise 2.7.5 (Multinomial Naive Bayes for nonnegative features)**.** In this exercise, we extend the multinomial naive Bayes classifier from discrete features to continuous nonnegative features.

1. Consider a κ-class classification problem for class labels $\mathcal{Y}=\{1,\ldots,\kappa\}$ and the feature vector $\boldsymbol{\phi}(\mathbf{x})\in\mathbb{R}^M$ of each data point $\mathbf{x}$ assume *nonnegative real* values in each coordinate.

[7] The resulting naive Bayes classifier is a special instance of the multinomial naive Bayes called *Bernoulli naive Bayes.*

2. We assume the following probabilitic model for (data, class) = $(\mathbf{X}, Y)$:

$$\text{(prior)} \quad \mathbb{P}(Y=i) =: q_i \qquad \text{for } i \in \{1,\dots,\kappa\} \tag{160}$$

$$\text{(class-conditional PMF)} \quad \mathbb{P}(\boldsymbol{\phi}(\mathbf{X}) = [c_1,\dots,c_M] \,|\, Y=i) \propto \prod_{j=1}^{M} (q_{ij})^{c_j} \qquad \text{for } i \in \{1,\dots,\kappa\} \tag{161}$$

$$\text{(constraints)} \quad \begin{cases} 0 \le q_1,\dots,q_\kappa \le 1, \qquad q_1 + \cdots + q_\kappa = 1 \\ 0 \le q_{i,1},\dots,q_{i,M} \le 1, \qquad q_{i,1} + \cdots + q_{i,M} = 1 \qquad \text{for } i = 0,1,\dots,\kappa. \end{cases} \tag{162}$$

We denote $\mathbf{W} = (q_i, q_{i,j})$ for the vector of parameters of our model above.

Now show the following:

(i) Suppose training examples are given: $(\boldsymbol{\phi}(\mathbf{x}_s), y_s)$, $s = 1,\dots,N$, and write $\boldsymbol{\phi}(\mathbf{x}_s) = [\phi_1(\mathbf{x}_s),\dots,\phi_N(\mathbf{x}_s)]$ for each s. Show that the maximum likelihood estimate of $\mathbf{W}$, $\hat{\mathbf{W}}$, is given by

$$\hat{\mathbf{W}} = \underset{\mathbf{W}=(q_i,q_{ij})}{\operatorname{argmax}} \left[M \sum_{i=1}^{\kappa} \left(\sum_{s=1}^{N} \mathbf{1}(y_s = i) \right) \log q_i + \sum_{i=1}^{\kappa} \sum_{j=1}^{M} \left(\sum_{s=1}^{N} \mathbf{1}(y_s = i)\phi_j(\mathbf{x}_s) \right) \log(q_{i,j}) \right]. \tag{163}$$

(ii) Show that the solution to (163) is given by

$$\hat{q}_i = \frac{1}{N} \sum_{s=1}^{N} \mathbf{1}(y_s = i), \qquad \hat{q}_{i,j} \propto \sum_{s=1}^{N} \mathbf{1}(y_s = i)\phi_j(\mathbf{x}_s) \quad \text{for } j \in \{1,\dots,M\}. \tag{164}$$

(Hint: Notice that the conclusion of Exercise 2.7.2 holds for nonnegative constants $c_1,\dots,c_N \ge 0$, not necessarily only for integers. Then argue similarly as in the proof of Proposition 2.7.1.)

(iii) Does the same conclusion hold if we allow negative feature values?

8. Classification with the 20Newsgroups dataset

All experiements in this section can be found in the Jupyter notebook Math156_20NewsGroups.ipynb.

8.1. Preliminaries. The 20 Newsgroups data set is a collection of approximately 18,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering[8]. We will use the convenient 20 Newgroups loader by sklearn for our experiments.

Of the 20 categories, we will use the following 10 categories:

$$\text{categories} = \begin{bmatrix} \text{'comp.graphics', 'comp.sys.mac.hardware', 'misc.forsale',} \\ \text{'rec.motorcycles', 'rec.sport.baseball', 'sci.med', 'sci.space',} \\ \text{'talk.politics.guns', 'talk.politics.mideast', 'talk.religion.misc'} \end{bmatrix} \tag{165}$$

Here is an example of raw text in the 'macs.hardware' category:

```
»» 4257 th doc:  Anyone know what would cause my IIcx to not turn on when I hit
the keyboard switch?  The one in the back of the machine doesn't work either...
The only way I can turn it on is to unplug the machine for a few minutes, then
plug it back in and hit the power switch in the back immediately...  Sometimes
this doesn't even work for a long time...
   I remember hearing about this problem a long time ago, and that a logic
board failure was mentioned as the source of the problem...is this true?
```

After we load the raw documents, we need to extract features from them and represent in vector form. There are two widely used vectorization method, the *bag-of-words* (BOW) and the *tf-idf* (term-frequency-inverse-document-frequency). We use a standard vocabulary $\mathcal{V}$ of 45534 words provided in sklearn. For a given document $\mathbf{x}$, its BOW and tf-idf feature vectors are defined by

$$\boldsymbol{\phi}_{\text{BOW}}(\mathbf{x}) = \left[\#\text{ of } j\text{th word in } \mathbf{x};\ 1 \le j \le |\mathcal{V}| \right], \tag{166}$$

[8] See http://qwone.com/~jason/20Newsgroups/

$$\boldsymbol{\phi}_{\text{tf-idf}}(\mathbf{x}) = \left[\frac{\#\text{ of } j\text{th word in } \mathbf{x}}{\#\text{ of documents that contains the } j\text{th word}} ; 1 \le j \le |\mathcal{V}| \right]. \tag{167}$$

Below are the BOW and tf-idf feature vectors of the sample document (4257th) above.

	Coordinate	Bag-of-words	tf-idf
ago	1286	1	0.115122
back	3503	3	0.288116
board	4850	1	0.138195
cause	6355	1	0.125130
either	12551	1	0.108417
failure	14215	1	0.170555
hearing	17842	1	0.170555
hit	18278	2	0.261832
iicx	19200	1	0.217706
immediately	19354	1	0.155037
keyboard	21813	1	0.160377
know	22116	1	0.075488
logic	23500	1	0.158818
long	23525	2	0.203652
machine	23918	2	0.266555
mentioned	25060	1	0.135408
minutes	25592	1	0.145699
plug	30501	1	0.171330
power	30919	1	0.111667
problem	31391	2	0.203382
remember	33429	1	0.119059
sometimes	37293	1	0.135854
source	37380	1	0.127372
switch	39267	2	0.321834
time	40488	2	0.164800
true	41292	1	0.114800
turn	41456	2	0.269503
unplug	42226	1	0.233554
way	43812	1	0.089654
work	44510	2	0.197874

FIGURE 25. Bag-of-words and tf-idf feature vectors of a sample document from 20Newsgroups.

One can think of the tf-idf representation as an inverse-document-frequency (idf) correction to the BOW representation. Namely, even though some words appear many times in a document (e.g., 'The'), if it appears in many other documents, we discount its frequency by the idf. On the other hand, if some words appear only a few times in a document (e.g., 'OECD'), if it does not appear in many other documents, we boost its frequency (relatively to the more widely used words).

8.2. Classification by multiclass Logistic Regression. In this subsection, we use the multiclass logistic regression (MLR) in Section 3.3 for the task of document classification for the 20Newsgroups dataset we discussed in Subsection 8.1. In Figure (26), we show the optimal MLR coefficients (for two classes) fitted to the 20Newsgroups dataset using the gradient descent algorithm. We show the coefficients for both choice of vectorizers, BOW and tf-idf. By comparing these plots, we see that the coefficients with BOW vectoerizer has larger variance than the one with tf-idf vectorizer (see the reported standard deviation in legends). This is because the inverse-document-frequency acts as a regularizer in feature extraction step and the resulting feature vector has smaller variance (check this numerically).

In Figure 29, we show the confusion matrix of testing the fitted MLR on the 20Newsgroups dataset. As before for the MNIST dataset (see Sec. 4), we use the MAP decision algorithm 105 in order to turn the computed predictive probabilities into final predicted class labels. Overall MLR perfoms well in the 10-class classification task regardless of the choice of the vectorizer, as the confusion matrices are close to being diagonal.

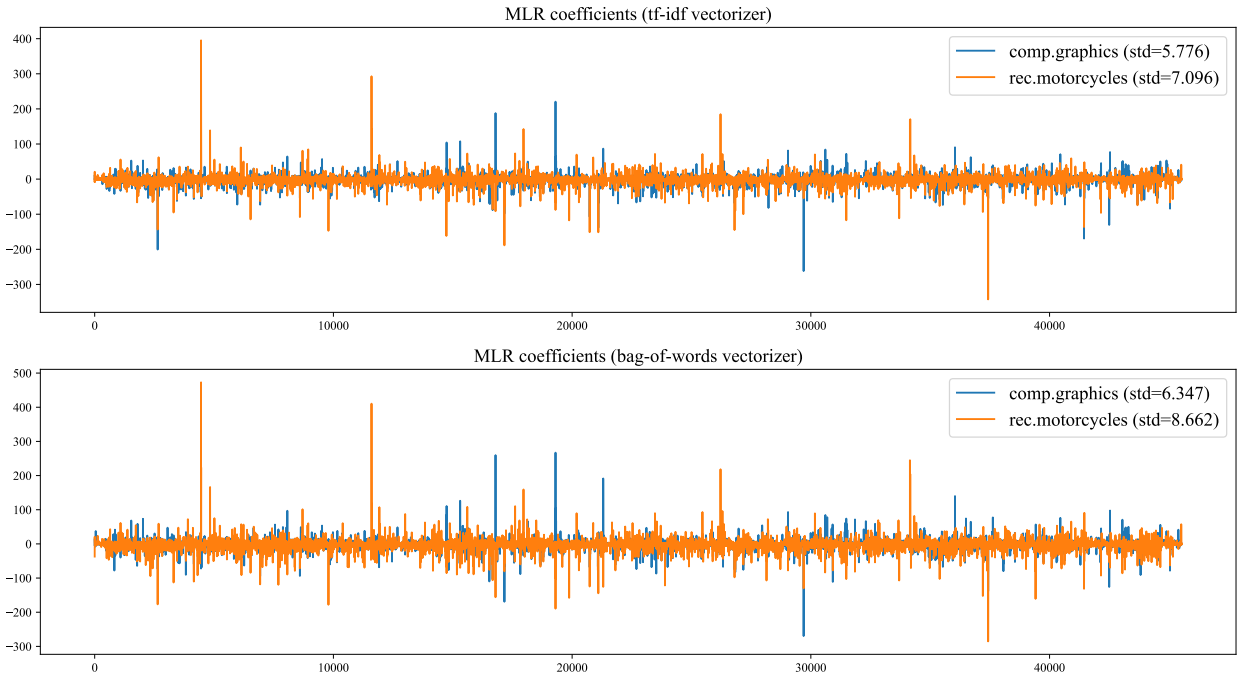


FIGURE 26. Examples of the fitted logistic regression coefficients to the 20Newsgroups dataset with digits. Gradient descent algorithm is used for training. Regression coefficients for the two classes 'comp.graphics' and 'rec.motorcycles' are shown.

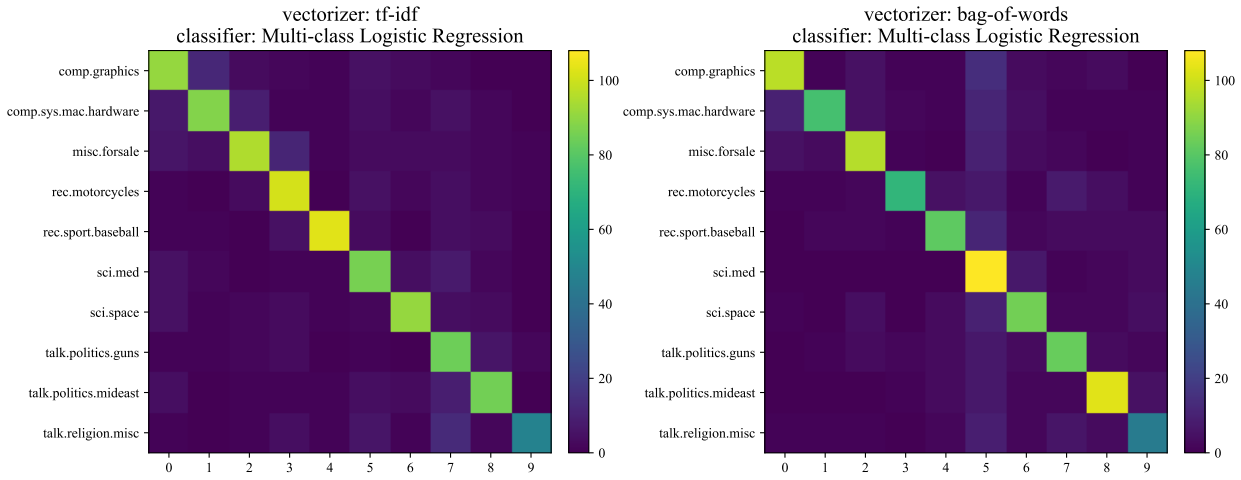


FIGURE 27. Confusion matrices for the 10-class document classficiation for the 20Newsgroups datatset with multiclass logistic regression. Vectorizers of bag-of-words and tf-idf are both used.

8.3. Classification by Multinomial Naive Bayes. In this subsection, we use the multinomial naive Bayesian classifier (MNB) in Section 7.2 for the task of document classification for the 20Newsgroups dataset in Subsection 8.1. In Figure (26), we show the optimal (maximum likelihood) class-conditional probabilities (see Prop. 2.7.1) for two classes. We plot the class-conditional PMFs on the vocabulary for both choice of vectorizers, BOW and tf-idf. Note that even though tf-idf feature vectors are not discrete, according to Exercise 2.7.5, we can still apply MNR to tf-idf feature vectors.

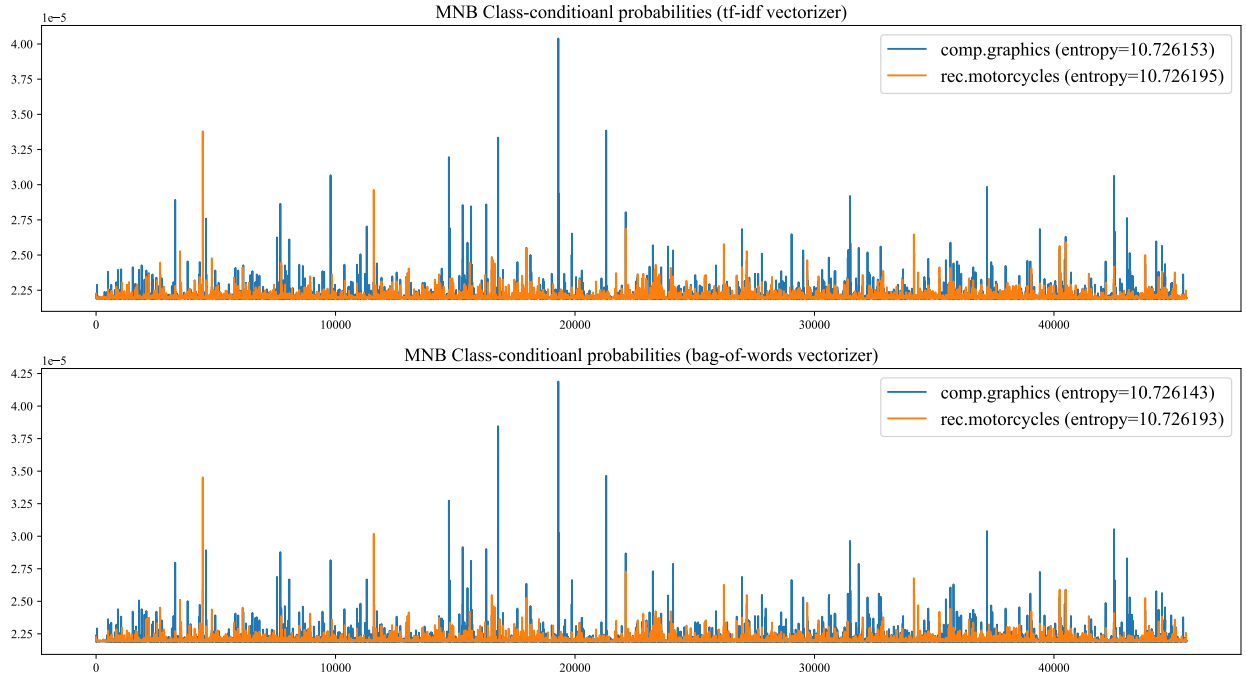


FIGURE 28. Examples of the fitted class-conditional probabilities of the 20Newsgroups dataset with digits. Two classes 'comp.graphics' and 'rec.motorcycles' are shown.

For each PMF, we also show its (Shannon) *entropy*, which is a measure of the 'information' or 'surprise' in a probability distribution. For a PMF $\mathbf{p} = [p_1, \ldots, p_n]$, its entropy $H(\mathbf{p})$ is defined as

$$H(\mathbf{p}) := -\sum_{i=1}^{n} p_i \log p_i \geq 0. \tag{168}$$

For any finite sample space, the PMF of maximum entropy on it is the uniform distribution, which should be the 'most disordered' among all possible PMFs, so in order to get to know it, we need large samples from it (hence more information content). On the other extreme, the PMFs supported on one single outcome have the lowest possible entropy of zero, and such PMFs contain 'no surprise'. In general, we regard PMFs with large entropy to be more disordered and hence contain more information. In Figure 28, we see that the class-conditional PMFs for the two classes have almost the same entropy regardless of the choice of the vectorizer. However, the ones with tf-idf vectorizer have slightly larger entropy for both classes. As before, this can be understood as the inverse-document-frequency being a regularizer in feature extraction, so that the feature values are more uniform on the coordinates. Since the (normalied) features have larger entropy, so are the fitted class-conditional probabilities.

In Figure 29, we show the confusion matrix of testing the fitted MNB on the 20Newsgroups dataset. As before for the MNIST dataset (see Sec. 4), we use the MAP decision algorithm 105 in order to turn the computed predictive probabilities into final predicted class labels. Overall MLR perfoms well in the 10-class classification task regardless of the choice of the vectorizer, as the confusion matrices are close to being diagonal. But less accuratly than multiclass logistic regression (see Figure 28). In particular, it seems that MNB suffers classifiying documents in the category 'sci.med'. (Why?)

Exercise 2.8.1. According to Exercise 2.7.5, we can apply multinomial naive Bayes for the image classification of MNIST dataset, since the features there (pixel values) are nonnegative.
(Ref: Math156_classification_MNIST.ipynb and Math156_20NewsGroups.ipynb)

(i) Reproduce Figure 16 for classifying digits '4' and '7' using multinomial naive Bayes. Compare the performance with logistic regression and probit regression.

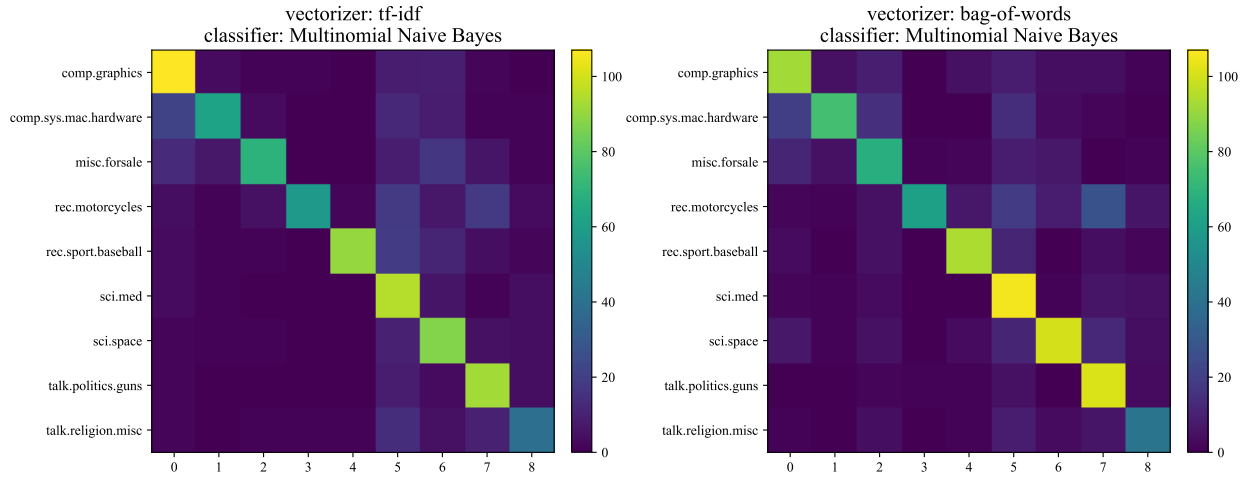


FIGURE 29. Confusion matrices for the 10-class document classficiation for the 20Newsgroups datatset with multinomial naive Bayes. Vectorizers of bag-of-words and tf-idf are both used.

(ii) Reproduce Figure 19 for classifying digites 0 through 4 using multinomial naive Bayes. Compare the performance with multiclass logistic regression.

Exercise 2.8.2. In this exercise, we investigate the 'robustness' of multinomial naive Bayes (MNB) classifier on the 20Newsgroups dataset. (Ref: Math156_20NewsGroups.ipynb)

(i) Modify the data preprocessing function "sample_multiclass_20NEWS" in the Jupyter notebook so that it has additional parameters 'noise_ratio' and 'noise_intensity': If 'noise_ratio' is $\varepsilon \in [0,1]$, and 'noise_intensity' is $\delta > 0$, then then it will add independent Uniform($[0,1]$) variables times δ to ε density of randomly chosen coordinates.

(ii) Choose two categories in the 20Newsgroups dataset and compare the performance of binary classification using MNB with varying noise parameters $(\varepsilon, \delta) \in \{0.1, 0.5, 0.9\} \times \{0.001, 0.01, 0.1\}$. (Use tf-idf vectorizer.) Do you think if MNB is robust against noise on the 20Newsgroups dataset? If so, is it more robust against noise ratio (ε) or noise intensity (δ)? Otherwise, can you say why?

(iii) Perform similar experiments as in **(ii)** for multiclass logistic regression.

9. Gaussian Naive Bayes

Consider the case of classification problem where the features are contiuous (e.g., height, weight, pixel values, tf-idf frequency) and may also assume negative values (e.g., bitcoin price change, EEG signal). In fact, in Exercise 2.7.5, we did discuss a naive Bayes model for continuous and nonnegative featrues, which was a straightforward generalization of the multinomial naive Bayes for nonnegative integer valued features. In this section, we introduce another widely used naive Bayes model called the *Gaussian naive Bayes* (GNB) model. The gist of it is the following: Assume each feature coordinate is a Gaussian RV with its own mean and variance, independently conditional on the class.

1. Consider a κ-class classification problem for class labels $\mathcal{Y} = \{1, \dots, \kappa\}$ and the feature vector $\boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^M$ of each data point $\mathbf{x}$ assume general *real* values in each coordinate (including negative values).

2. We assume the following probabilitic model for (data, class) = $(\mathbf{X}, Y)$:

$$\text{(prior)} \quad \mathbb{P}(Y = i) =: q_i \qquad \text{for } i \in \{1, \dots, \kappa\} \tag{169}$$

$$\text{(class-conditional PDF)} \quad p(\boldsymbol{\phi}(\mathbf{X}) = [c_1, \dots, c_M] \,|\, Y = i) \propto \prod_{j=1}^{M} f_{\mu_{ij}, \sigma_{ij}}(c_j) \qquad \text{for } i \in \{1, \dots, \kappa\}, \tag{170}$$

$$\text{(constraints)} \quad \begin{cases} 0 \le q_1, \dots, q_\kappa \le 1, & q_1 + \cdots + q_\kappa = 1 \\ \sigma_{i,j} \ge 0 & \text{for all } i \in \{1, \dots, \kappa\},\ j \in \{1, \dots, M\}, \end{cases} \tag{171}$$

where $f_{\mu,\sigma}(x)$ is the PDF of $N(\mu, \sigma)$:

$$f_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \tag{172}$$

We denote $\mathbf{W} = (q_i, \mu_{ij}, \sigma_{ij})$ for the vector of parameters of our model above.

Suppose training examples are given: $(\boldsymbol{\phi}(\mathbf{x}_s), y_s)$, $s = 1, \dots, N$, and write $\boldsymbol{\phi}(\mathbf{x}_s) = [\phi_1(\mathbf{x}_s), \dots, \phi_N(\mathbf{x}_s)]$ for each s. We appeal to the general maximum likelihood formulation of naive Bayes models in (131) and obtain the following:

$$\hat{\mathbf{W}} = \operatorname*{arg\,max}_{\mathbf{W}} \left[\sum_{s=1}^{N} \sum_{i=1}^{\kappa} p\mathbf{1}(y_s = i) \log q_i + \sum_{s=1}^{N} \sum_{j=1}^{p} \sum_{i=1}^{\kappa} \mathbf{1}(y_s = i) \log \frac{1}{\sqrt{2\pi\sigma_{ij}^2}} \exp\left(-\frac{(\phi_j(\mathbf{x}_s) - \mu_{ij})^2}{2\sigma_{ij}^2}\right) \right] \tag{173}$$

$$= \operatorname*{arg\,max}_{\mathbf{W}} \left[\ell_{GNB}(\mathbf{W}) := \begin{aligned} & p \sum_{i=1}^{\kappa} \left(\sum_{s=1}^{N} \mathbf{1}(y_s = i) \right) \log q_i - \frac{1}{2} \sum_{j=1}^{p} \sum_{i=1}^{\kappa} \left(\sum_{s=1}^{N} \mathbf{1}(y_s = i) \right) \log(2\pi\sigma_{ij}^2) \\ & - \sum_{s=1}^{N} \sum_{j=1}^{p} \sum_{i=1}^{\kappa} \mathbf{1}(y_s = i) \frac{(\phi_j(\mathbf{x}_s) - \mu_{ij})^2}{2\sigma_{ij}^2} \end{aligned} \right]. \tag{174}$$

The optimization problem (173) can be solved in closed form using standard arguments. The details are given in (2.9.1).

Exercise 2.9.1. Let ℓ_{GNB} denote the Gaussian naive Bayes loss function defined in (173). Denote the solution of (173) as $\hat{\mathbf{W}} = (\hat{\mathbf{q}}_i, \hat{\mu}_{ij}, \hat{\sigma}_{ij})$.

(i) Argue that the optimal prior PMF $\hat{\mathbf{q}} := [\hat{q}_1, \dots, \hat{q}_\kappa]$ solves the following optimization problem

$$\hat{\mathbf{q}} = \operatorname*{arg\,max}_{\substack{\mathbf{q}=[q_1,\dots,q_\kappa] \\ \text{PMF on } \{1,\dots,\kappa\}}} \sum_{i=1}^{\kappa} \left(\sum_{s=1}^{N} \mathbf{1}(y_s = i) \right) \log q_i. \tag{175}$$

Conclude that $\hat{q}_i = \frac{1}{N} \sum_{s=1}^{N} \mathbf{1}(y_s = i)$ (Hint: Use Exercise 2.7.2).

(ii) Show that

$$\frac{\partial \ell_{GNB}(\mathbf{W})}{\partial \mu_{ij}} = \sum_{s=1}^{N} \mathbf{1}(y_s = i) \frac{(\phi_j(\mathbf{x}_s) - \mu_{ij})}{\sigma_{ij}^2}. \tag{176}$$

Deduce that $\hat{\mu}_{ij}$ equals the following 'sample mean of the jth feature in class i':

$$\hat{\mu}_{ij} = \frac{\sum_{s=1}^{N} \mathbf{1}(y_s = i) \phi_j(\mathbf{x}_s)}{\sum_{s=1}^{N} \mathbf{1}(y_s = i)}. \tag{177}$$

(Compare this with $\hat{q}_{ij}$ in Prop. 2.7.1.)

(iii) Show that

$$\frac{\partial \ell_{GNB}(\mathbf{W})}{\partial \sigma_{ij}} = -\sigma_{ij}^{-1} \left(\sum_{s=1}^{N} \mathbf{1}(y_s = i) \right) + \sigma_{ij}^{-3} \sum_{s=1}^{N} \mathbf{1}(y_s = i)(\phi_j(\mathbf{x}_s) - \mu_{ij})^2. \tag{178}$$

Deduce that $\hat{\sigma}_{ij}$ equals the following 'variance of the class-i empirical distribution':

$$\hat{\sigma}_{ij}^2 = \frac{\sum_{s=1}^{N} \mathbf{1}(y_s = i)(\phi_j(\mathbf{x}_s) - \hat{\mu}_{ij})^2}{\sum_{s=1}^{N} \mathbf{1}(y_s = i)}, \tag{179}$$

where $\hat{\mu}_{ij}$ is given in **(ii)**.

CHAPTER 3

Neural Networks

In this chapter, we discuss a simple form of multilayer feedforward neural network. The model is simple enough to discuss without too much technical details but also is sophisticated enough to convey essential ideas in modern neural netowrk models.

1. Model formulation

Very simpliy put, one can think of a feedword neural network as a composition of multiple general linear models (e.g., logistic regression, probit regression). To motivate this construction, recall that in generalized linear models, we model the predictive probabilities as

$$\hat{\mathbf{y}} := \underbrace{\begin{bmatrix} p(1 \mid \boldsymbol{\phi}(\mathbf{x})) \\ \vdots \\ p(\kappa \mid \boldsymbol{\phi}(\mathbf{x})) \end{bmatrix}}_{\text{predictive prob.}} = \sigma(\underbrace{\boldsymbol{\phi}(\mathbf{x})}_{???}{}^T \mathbf{W}), \tag{180}$$

where σ is an activation function (e.g., signomid, logit, ReLu, tanh. See Figure 31.), $\boldsymbol{\phi} \in \mathbb{R}^M$ is a M-dimensional feature vector of data $\mathbf{x}$, $\mathbf{W} \in \mathbb{R}^{M \times \kappa}$ is a parameter matrix to be learned. So far, we have chosen a particular feature vector $\boldsymbol{\phi}(\mathbf{x})$ before we fit the model to the training data (e.g., pixel values for MNIST images, bag-of-words or tf-idf vectors for 20Newgroups texts). But how do we know what feature should we choose? Could we somehow also *learn* what feature to use during the training process?[1]

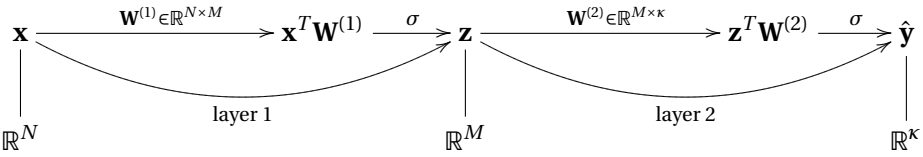
A simple idea is to *recycle the same generalized linear model in order to model the feature vector*. Namely, we consider

$$\mathbf{z} := \boldsymbol{\phi}(\mathbf{x}) = \sigma(\mathbf{x}^T \mathbf{W}'), \tag{181}$$

where we view $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{W}' \in \mathbb{R}^{N \times M}$ is another parameter matrix. Rewriting the parameter vectors as $\mathbf{W}^{(1)} = \mathbf{W}' \in \mathbb{R}^N$ and $\mathbf{W}^{(2)} = \mathbf{W} \in \mathbb{R}^M$, the whole model reads as

$$\hat{\mathbf{y}} = \sigma\left(\sigma(\mathbf{x}^T \mathbf{W}^{(1)})^T \mathbf{W}^{(2)}\right). \tag{182}$$

As a diagram, it can also be described as (see also Figure 30)



(183)

This model is called a *two-layer neural network* (a.k.a. *multilayer perceptron*). Since the information only flows across layers in one direction, it is also called a *feedforward* neural network[2]. It is also called

[1] In the language of linear algebra, the feature vectors are coordinates on a chosen basis. We have worked with a fixed basis, but here we are attempting to also learn the basis from the training data. We will revisit this point later in dimensionality reduction (e.g., principal component analysis, nonnegative matrix factorization)

[2] Recurrent Neural Networks, which is a current state-of-the-art model for natural languate processing, use feedback loop.

a *fully connected* neural network, since each coordinate in a given layer can get information from all coordinates in the previous layer[3].

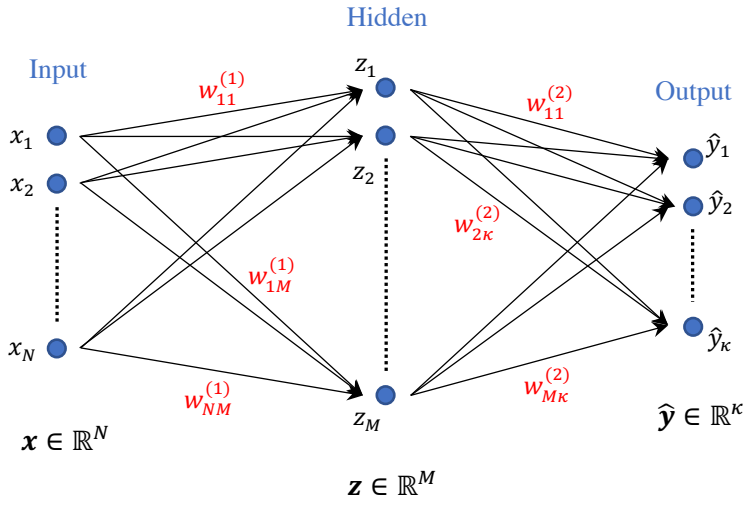


FIGURE 30. Diagrammatic representation of a 2-layer neural network.

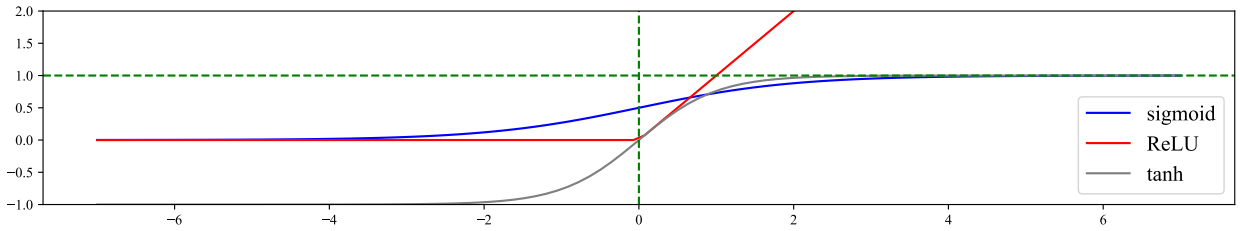


FIGURE 31. Common activation functions used in neural networks. Sigmoid: $x \mapsto 1/(1+e^{-x})$, ReLU: $x \mapsto \max(0,x)$, and tanh: $x \mapsto (e^{2x}-1)/(e^{2x}+1)$.

2. Training Feedforward Neural Networks

In this section, we discuss how we train feedforward neural networks.

2.1. Optimization framework. As always, the key step is to formulate a suitable optimization problem for training the model. Suppose we are given training examples $(\mathbf{x}_1,\mathbf{y}_1),\dots,(\mathbf{x}_n,\mathbf{y}_n)$ in $\mathbb{R}^n \times \mathbb{R}^\kappa$. Denote the predicted output for each input $\mathbf{x}_s$ using parameter $\mathbf{w}$ by $\hat{\mathbf{y}}(\mathbf{x}_s;\mathbf{w}) = [\hat{y}_1(\mathbf{x}_s;\mathbf{w}),\dots,\hat{y}_\kappa(\mathbf{x}_s;\mathbf{w})]$. We think of them as the output computed by the two-layer neural network (183), where $\mathbf{w}$ denotes the collection of all paramter matrices $[\mathbf{W}^{(1)},\mathbf{W}^{(2)}]$[4]. We seek to find an optimal parameter $\hat{\mathbf{w}}$ that minimizes the overall trainign loss:

$$\hat{\mathbf{w}} := \arg\min_{\mathbf{w}} \left[\ell_{NN}(\mathbf{w}) := \sum_{s=1}^{n} \ell(\mathbf{y}_s, \hat{\mathbf{y}}(\mathbf{x}_s;\mathbf{w})) \right], \tag{184}$$

where ℓ is a loss function that measures the discrepancy between the true output $\mathbf{y}_i$ and the predicted output $\hat{\mathbf{y}}(\mathbf{x}_i;\mathbf{w})$, which may depend on whether we are solving a regression or a classification problem[5].

[3]Convolutional Neural Network, wich is a current state-of-the-art for image classification, uses locally connected convolutional layers, mimicing the human visual cortex.

[4]However, our discussion in this section is general and not necessarily limited to the 2-layer neural networks.

[5]For classification problems, we may represent each class i by the indicator vector (one-hot encoding) $[0,\dots,0,1,0,\dots,0] \in \mathbb{R}^\kappa$, where the 1 is at the ith coordinate.

The standard choices are

$$\texttt{Regression} \qquad \text{(square loss):} \quad \ell(\mathbf{y}_s, \hat{\mathbf{y}}(\mathbf{x}_s;\mathbf{w})) = \frac{1}{2}\|\mathbf{y}_s - \hat{\mathbf{y}}(\mathbf{x}_s;\mathbf{w})\|_F^2, \tag{185}$$

$$\texttt{Classification} \quad \text{(cross-entropy loss):} \quad \ell(\mathbf{y}_s, \hat{\mathbf{y}}(\mathbf{x}_s;\mathbf{w})) = -\sum_{i=1}^{\kappa} \mathbf{1}(\mathbf{y}_s = \text{class } i) \log \hat{y}_i(\mathbf{x}_s;\mathbf{w}), \tag{186}$$

$$\text{(softmax-cross-entropy loss):} \quad \ell(\mathbf{y}_s, \hat{\mathbf{y}}(\mathbf{x}_s;\mathbf{w})) = -\sum_{i=1}^{\kappa} \mathbf{1}(\mathbf{y}_s = \text{class } i) \log \operatorname{softmax}_i\left(\hat{\mathbf{y}}(\mathbf{x}_s;\mathbf{w})\right). \tag{187}$$

The square loss for the regression case is natural and we have seen it for polynomial regression models in Section 2. On the other hand, for classification problems, recall that we usually model the predictive probabilities given features so that $\hat{\mathbf{y}}(\mathbf{x}_s;\mathbf{w}) \in \mathbb{R}^\kappa$ can be viewed as a predictive PMF of the class of $\mathbf{x}_s$. One can think of the 'cross-entropy' as a similarity meature between two probability distributions. Namely, if we have two PMFs $\mathbf{p} = [p_1, \ldots, p_\kappa]$ and $\mathbf{q} = [q_1, \ldots, q_\kappa]$, then the *cross-entropy* of $\mathbf{q}$ relative to $\mathbf{p}$ is defined by

$$H(\mathbf{p}, \mathbf{q}) = -\sum_{i=1}^{\kappa} p_i \log q_i. \tag{188}$$

Note that the following class-indicator vector

$$\left[\mathbf{1}(\mathbf{y}_s = \text{class } 1), \ldots, \mathbf{1}(\mathbf{y}_s = \text{class } \kappa)\right] \tag{189}$$

is a PMF on the κ classes. Hence the cross-entropy loss in (186) is in fact the cross-entropy between the class-indicator vector above with the predictive PMF $\hat{\mathbf{y}}(\mathbf{x}_s;\mathbf{w}) \in \mathbb{R}^\kappa$. Lastly, the softmax-cross-entropy loss in (187) is frequently used in classification problems where the target output $\mathbf{y}_s$ is a PMF but the predicted output $\hat{y}(\mathbf{x}_s;\mathbf{w})$ is not and is yet to be converted into a PMF. In this case, one can use the following softmax transform

$$\mathbf{x} = [x_1, \ldots, x_\kappa]^T \mapsto \operatorname{softmax}(\mathbf{x}) = \left[\frac{\exp(x_1)}{\sum_{i=1}^{\kappa} \exp(x_i)}, \ldots, \frac{\exp(x_n)}{\sum_{i=1}^{\kappa} \exp(x_i)}\right]^T \in \mathbb{R}^\kappa \tag{190}$$

first and then compute the cross-entropy loss, which results in the combined loss of the form in (187).

One of the recurreing theme in all of regression and classification models we studied so far is to take the *loss function as the negative log likelihood*. Namely,

$$\ell_{\text{total}}(\mathbf{w}) := -\log L(y_1, \ldots, y_n; \mathbf{w}). \tag{191}$$

In fact, the square loss and cross-entropy loss above can be derived from this maximum likelihood formulation. For instance, consider the case of κ-class classification. Recall that $\hat{\mathbf{y}}(\mathbf{x}_s;\mathbf{w})$ is the predictive PMF so that its ith coordinate, $\hat{\mathbf{y}}_i(\mathbf{x}_s;\mathbf{w})$, is the probability that the class of $\mathbf{x}_s$ is i according to our model. Hence the joint likelihood is

$$L(y_1, \ldots, y_N; \mathbf{w}) = \mathbb{P}(Y_1 = y_1, \ldots, Y_N = y_N; \mathbf{w}) = \prod_{s=1}^{n} \prod_{i=1}^{\kappa} (\hat{y}_i(\mathbf{x}_s;\mathbf{w}))^{\mathbf{1}(\mathbf{y}_s = \text{class } i)} \tag{192}$$

Hence we have

$$-\log L(y_1, \ldots, y_N; \mathbf{w}) = -\sum_{s=1}^{n} \sum_{i=1}^{\kappa} \mathbf{1}(\mathbf{y}_s = \text{class } i) \log(\hat{y}_i(\mathbf{x}_s;\mathbf{w})), \tag{193}$$

which is exactly the total loss function in (184) using the cross-entropy loss in (186)[6].

Exercise 3.2.1. Derive the total loss function in (184) using the square loss in (185) by using the maximum likelihood framework in (191). You may assume that the true output $\mathbf{y} \in \mathbb{R}^\kappa$ is generated by a k-diemnsional multivariate Normal distribution $N(\hat{\mathbf{y}}(\mathbf{x};\mathbf{w}), \sigma^2 I)$, where I is the $\kappa \times \kappa$ identity matrix. (See Example 3.2.2.)

[6]Compare this computation with the one for the multiclass logistic regression in Section 3.3.

Example 3.2.2 (Multivariate Gaussian distribution)**.** The a p-dimensional *multivariate Gaussian distribution* is a function $\mathbb{R}^p \to \mathbb{R}$ given by

$$p(\mathbf{x};\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right], \tag{194}$$

where $\boldsymbol{\mu} \in \mathbb{R}^p$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{p\times p}$ is a symmetric positive semi-definite matrix[7] and $|\boldsymbol{\Sigma}|$ denotes the determinant of $\boldsymbol{\Sigma}$. If a random vector $\mathbf{X} \in \mathbb{R}^p$ has distribution given by (194), then we denote $\mathbf{X} \sim N(\boldsymbol{\mu},\boldsymbol{\Sigma})$. Then it has the following properties

$$\boldsymbol{\mu} = \mathbb{E}[X], \qquad \boldsymbol{\Sigma} = \mathrm{Cov}(X) = \mathbb{E}\left[(\mathbf{X}-\boldsymbol{\mu})(\mathbf{X}-\boldsymbol{\mu})^T\right]. \tag{195}$$

For these reasons, we call $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ the *mean* and the *covariance matrix* of $\mathbf{X}$, respectively. ▲

2.2. Complexity of the optimization problem. The first thing to note for solving the optimization problem (184) for the two-layer neural networks is that it is a *non-convex* optimization problem. Namely, in both cases of square and cross-entropy loss, the total loss $\ell_{\text{total}}(\mathbf{w})$ depends on the model output $\hat{\mathbf{y}}(\mathbf{x}_s;\mathbf{w})$ in a convex manner but the model output depends on the weight matrices $\mathbf{w} = [\mathbf{W}^{(1)},\mathbf{W}^{(2)}]$ in a non-convex manner. On the contrary, recall that the cross-entropy loss for logistic regression, or a single-layer neural network, is a convex function on the parameters (see Exercise 2.3.2). So, unfortunately, it is going to be very difficult (practically impossible) to find the global minimum of the total loss for training a 2-layer neural network.

Another nice way to see the complexity of the current optimization problem is using model symmetry. Observe that the model output of a 2-layer neural network (see Figure 30) is invariant under permuting the order of hidden units along with the corresponding in- and out-edge weights. For instance, if we swap z_1 and z_N, then the weights $w_{i1}^{(1)}$ from input x_i to z_1 should be renamed to $w_{iN}^{(1)}$ and also the weights $w_{N1}^{(1)}$ from x_i to z_N should be renamed to $w_{i1}^{(1)}$, and so on; also the out-weights $w_{1j}^{(2)}$ and $w_{Nj}^{(2)}$ should be swapped. The model output $\mathbf{y}$ should be unaware of this permutation since the values of the two linear transforms in the network is unchanged. Note that there are $M!$ ways to permute the hidden units and their neural weights in this way. A consequence of this symmetry observation is that there are at least $M!$ 'equivalent local minima' of the total loss function (184)[8]. For instance, if we use $M = 10$ hidden units, then there are at least $10! = 3,628,800$ global minima to look for.

Since the training of the 2-layer neural network has a complex 'optimization landscape', we change our goal from the ambitious one of finding a global minimum to the more modest one of finding a local minimum. It is a consistent experience in the literature that a local minimum found by some standard non-convex optimization algorithm (e.g., stochastic gradient descent, block coordinate descent) will perform as well as any other local minima, and this phenomenon belongs to an active area of machine learning research.

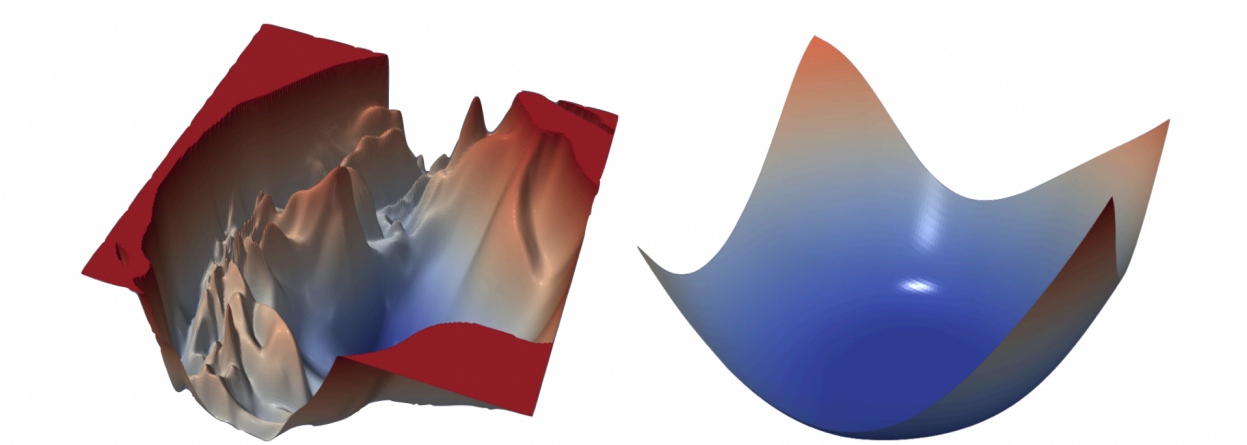
2.3. Stochastic Gradient Descent and Backpropagation. In order to find a local minimum of of (183), consider the familar optimizatoin algorithm of gradient descent, which we introduced in discussing training of logistic regression in (82):

$$\mathbf{w}_{t+1} \longleftarrow \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} \ell_{NN}(\mathbf{w}_t) \tag{196}$$

One of the main bottleneck in the development of neural network models in machine learning literature was the huge computational cost in computing the full gradient $\ell_{NN}(\mathbf{w}_t)$ for using the above gradient descent algorithm. We will discuss two ways to overcome this issue: 1) Stochastic Gradient Descent (SGD) and 2) Backpropagation.

[7] A matrix $\mathbf{A} \in \mathbb{R}^{p\times p}$ is *symmetric* if $\mathbf{A} = \mathbf{A}^T$ and *positive semi-definite* if $\mathbf{x}^T\mathbf{A}\mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{R}^p$. Moreover, $\mathbf{A}$ is *positive definite* if it is positive semi-definite and furthermore, $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$ if $\mathbf{x} \neq \mathbf{0}$.

[8] For the hyperbolic tangent activation function $\sigma(x) = \tanh(x) = (e^{2x}-1)/(e^{2x}+1)$, the model output is also invariant under sign flips of the neural weights, so there are at least $M!2^M$ equivalent local minimia.



(a) ResNet-110, no skip connections (b) DenseNet, 121 layers

FIGURE 32. Optimization landscape for ResNet-110-noshort (left, nonconvex) and DenseNet for CIFAR-10 (right, convex). Figure excerpted from [LXT+17].

2.3.1. *Stochastic Gradient Descent.* The idea of SGD is not limited only to the context of neural networks, but is applicable for general nonconvex optimization problems. Simply speaking it is to use only a 'partial gradient' in each iteration of the gradient descent, which typically requires much less computational cost. In our case, note that we can decompose the gradient $\nabla \ell_{NN}(\mathbf{w}_t)$ as the sum of gradients per sample:

$$\nabla \ell_{NN}(\mathbf{w}_t) = \sum_{s=1}^{n} \nabla_{\mathbf{w}} \ell(\mathbf{y}_s, \hat{\mathbf{y}}(\mathbf{x}_s; \mathbf{w}_t)), \tag{197}$$

by using the definition of total loss function ℓ_{NN} and the linearity of differentiation. Hence, instead of computing the full gradient $\nabla \ell_{\mathbf{w}} \ell(\mathbf{w}_t)$, why don't we choose a small set $I \subseteq \{1, \ldots, n\}$ of training data samples and compute the corresponding partial gradient

$$\ell_{NN;I} \ell(\mathbf{w}_t) := \sum_{i \in I} \nabla_{\mathbf{w}} \ell(\mathbf{y}_s, \hat{\mathbf{y}}(\mathbf{x}_s; \mathbf{w}_t)). \tag{198}$$

The stochastic component of SGD lies in choosing the index set I randomly. Hence the corresponding SGD algorithm takes the following form:

Algorithm 1 Stochastic Gradient Descent for Neural Networks

1: **Input:** $\mathbf{w}_0$ (initial parameter); T (number of iterations); $(\eta_t)_{t \geq 1}$, (non-increasing weights in $(0, 1]$); $n_0 \in \{1, \ldots, n\}$ (size of subsample)
2: **for** $t = 1, \ldots, T$ **do**:
3: Sample $I \subseteq \{1, \cdots, n\}$ with $|I| = n_0$ over all subsets of size n_0;
4: $\mathbf{w}_t \longleftarrow \mathbf{w}_{t-1} - \eta_t \nabla_{\mathbf{w}} \ell_{NN;I}(\mathbf{w}_{t-1})$
5: **end for**
6: **output:** $\mathbf{w}_T$

An extreme case of the SGD algorithm in Algorithm 1 is when the size n_0 of subsample is 1. In that case, in each iteration, we only use a single randomly chosen training example to update the current parameter $\mathbf{w}_t$. In general, the chosen set of training examples $\{\mathbf{x}_i ; i \in I\}$ is called a *minibatch*, and SGD amounts to process only a randomly selected minibatch in each iteration of gradient descent, instead of the full *batch* $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. Processing minibatches instead of the full batch is what reduces the computational cost of SGD. Moreover, the randomized choice of minibatches is what helps the SGD not to overfit to some particular training examples and to escape from undesirable local minima.

It is a classical result in the theory of nonconvex optimization that the parameters $(\mathbf{w}_t)_{t\geq 0}$ produced by SGD converges to a stationary point of the loss function (183) if a 'diminishing step size' is used: $\sum_{t=1}^{\infty}\eta_t=\infty$ and $\sum_{t=1}^{\infty}\eta_t^2<\infty$ when the loss function is nice enough (see, e.g., [SSY18]). As in the case of logistic regression, we will use the schedule $\eta_t=\gamma(\log t)/\sqrt{t}$, where $\gamma>0$ is a constant.

2.3.2. *Backpropagation. Error backpropagation* is an effective method to *evaluate* the gradient of a feedforward neural network at the current parameters, first developed by Rumelhart, Hinton, and Williams [RHW86].

In order to simplify the notation, we only consider the loss at sth training sample $(\mathbf{x}_s,\mathbf{y}_s)$:

$$\ell_s(\mathbf{w}):=\ell(\mathbf{y}_s,\hat{\mathbf{y}}(\mathbf{x}_s;\mathbf{w})). \tag{199}$$

In order to execute the SGD algorithm for training neural networks, we need to compute the following derivatives: For each $s\in\{1,\dots,n\}$,

$$\frac{\partial\ell_s}{\partial w_{ij}^{(1)}},\qquad \frac{\partial\ell_s}{\partial w_{jq}^{(2)}}. \tag{200}$$

Namely, we want to differentiate the loss function with respect to the edge weights. However, it would be extremely helpful for us to compute the derivative of the loss function with respect to *node weights* z_j and $\hat{y}_q$. Recall the following relations:

$$z_j=\sigma\left(\mathbf{x}^T\mathbf{W}^{(1)}[:,j]\right)=\sigma\left(\sum_{i=1}^{N}x_i w_{ij}^{(1)}\right),\qquad \hat{y}_q=\sigma\left(\mathbf{z}^T\mathbf{W}^{(2)}[:,q]\right)=\sigma\left(\sum_{j=1}^{M}z_j w_{jq}^{(2)}\right). \tag{201}$$

Denoting $\mathbf{y}_s=[(\mathbf{y}_s)_1,\dots,(\mathbf{y}_s)_\kappa]$, we first differentiate the loss function with respect to $\hat{y}_q$:

$$\text{(square loss):}\qquad \frac{\partial\ell_s}{\partial\hat{y}_q}=\frac{1}{2}\frac{\partial}{\partial\hat{y}_q}\|\mathbf{y}_s-\hat{\mathbf{y}}(\mathbf{x}_s;\mathbf{w})\|_F^2=\left(\hat{y}_q-(\mathbf{y}_s)_q\right), \tag{202}$$

$$\text{(cross-entropy loss):}\qquad \frac{\partial\ell_s}{\partial\hat{y}_q}=-\frac{\partial}{\partial\hat{y}_q}\sum_{i=1}^{\kappa}\mathbf{1}(\mathbf{y}_s=\text{class } i)\log\hat{y}_i(\mathbf{x}_s;\mathbf{w})=-\frac{\mathbf{1}(\mathbf{y}_s=\text{class } q)}{\hat{y}_q}. \tag{203}$$

$$\text{(softmax-cross-entropy loss):}\qquad \frac{\partial\ell_s}{\partial\hat{y}_q}=-\frac{\partial}{\partial\hat{y}_q}\sum_{i=1}^{\kappa}\mathbf{1}(\mathbf{y}_s=\text{class } i)\log\operatorname{softmax}_i\left(\hat{\mathbf{y}}(\mathbf{x}_s;\mathbf{w})\right) \tag{204}$$

$$=\operatorname{softmax}_q\left(\hat{\mathbf{y}}(\mathbf{x}_s;\mathbf{w})\right)-(\mathbf{y}_s)_q. \tag{205}$$

For (205), see Exercise 3.2.3.

Next, we compute the derivatives of the loss function with respect to the hidden state z_j. By using chain rule:

$$\underbrace{\frac{\partial\ell_s}{\partial z_j}}_{\text{'error' at node } z_j}=\sum_{q=1}^{\kappa}\frac{\partial\ell_s}{\partial\hat{y}_q}\frac{\partial\hat{y}_q}{\partial z_j}=\sum_{q=1}^{\kappa}\underbrace{\frac{\partial\ell_s}{\partial\hat{y}_q}}_{\text{'error' at node } \hat{y}_q}\sigma'\left(\underbrace{\mathbf{z}^T\mathbf{W}^{(2)}[:,q]}_{\text{activation for } \hat{y}_q}\right)w_{jq}^{(2)}, \tag{206}$$

where σ' denotes the derivative of the activation function σ. The equation in (206) is an instance of error backpropagation (See Figure 35).

In order to explain the mechanics of error backpropagation, we introduce the notion of 'errors at nodes' of a neural network to be the loss function with respect to the state of the node[9]. For instance, $\frac{\partial\ell_s}{\partial\hat{y}_q}$ is the error at node $\hat{y}_q$ and $\frac{\partial\ell_s}{\partial z_j}$ is the error at node z_j, and so on[10]. With this notion at hand, in (202) and (203), we computed the errors at the output nodes $\hat{y}_j$ directly. Also, we can view (206) as a *recursive relation* that computes the error at node z_j in terms of the errors at nodes $\hat{y}_q$ in the output layer. Below is a verbal description of the error backpropagation for 2-layer neural networks:

[9]We are making a slight abuse of notation of identiying a node in a neural network with its state.

[10]Why such a derivative is called an 'error'? One can make sense of it at least for the square loss at the output layer. See (202).

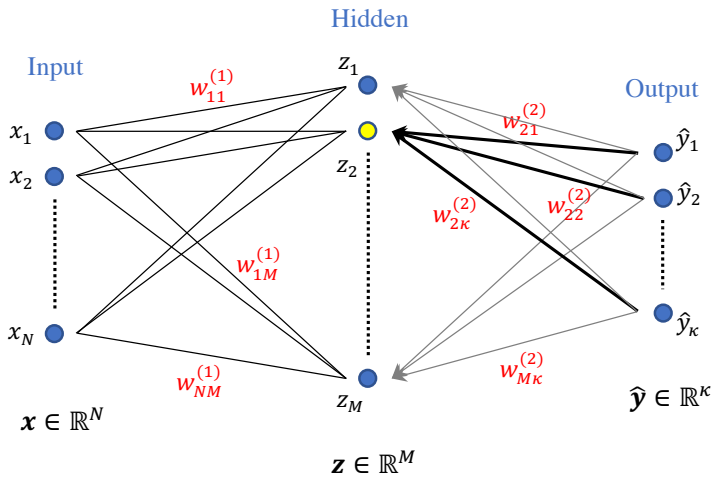


FIGURE 33. Error backpropatation in a 2-layer neural network. The 'errors' computed at the output layer are backpropagated to the second hidden node through the edges, where the corresponding edge weight is multiplied. The total sum of such weighted backpropagated errors at node z_2 is then the 'error' at node z_2.

Computation of gradients for training a 2-layer FFNN

1. Forward propagate the input all the way to the output layer and compute the activation and state of every node.
2. Compute the errors at the output nodes using (202)-(205).
3. Backprogagate the errors in the output layer to the hidden layer and compute the errors at hidden nodes, using (206).

In order to see that such recursion can be used to compute the errors in deeper layers, it is instrumental to compute the errors in the input layer, even though we do not use them for training the 2-layer neural network. One can easily verify

$$\underbrace{\frac{\partial \ell_s}{\partial x_i}}_{\text{'error at node } x_i\text{'}} = \sum_{j=1}^{M} \frac{\partial \ell_s}{\partial z_j}\frac{\partial z_j}{\partial x_i} = \sum_{j=1}^{M} \underbrace{\frac{\partial \ell_s}{\partial z_j}}_{\text{'error' at } z_j} \sigma'\left(\underbrace{\mathbf{x}^T\mathbf{W}^{(1)}[:,j]}_{\text{activation for } z_j}\right) w_{ij}^{(1)}. \tag{207}$$

Hence, *if we had a deep neural network with 2 or more layers, then we could use the same backpropagation recursion layer by layer to compute the errors at each node in all layers.*

Now that we can compute the errors at each node, it remains to see how we use them to compute the desired derivatives of the loss function with respect to the neural weights in (200). This can be easily done by the relation (201) and chain rule:

$$\frac{\partial \ell_s}{\partial w_{jq}^{(2)}} = \frac{\partial \ell_s}{\partial \hat{y}_q}\frac{\partial \hat{y}_q}{\partial w_{jq}^{(2)}} = \frac{\partial \ell_s}{\partial \hat{y}_q}\sigma'(\mathbf{z}^T\mathbf{W}^{(2)}[:,q])z_j, \tag{208}$$

$$\frac{\partial \ell_s}{\partial w_{ij}^{(1)}} = \frac{\partial \ell_s}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}^{(1)}} = \frac{\partial \ell_s}{\partial z_j}\sigma'(\mathbf{x}^T\mathbf{W}^{(1)}[:,j])x_i. \tag{209}$$

Lastly, with all the derivatives as above, we can now execute the SGD algorithm (Algorithm 1) by aggregating the above per-sample derivatives to get the minibatch gradient $\nabla_{\mathbf{w}}\ell_{NN;I}(\mathbf{w}_{t-1})$ by using (198).

Exercise 3.2.3 (Derivative of softmax-cross-entropy)**.** The softmax function $\sigma : \mathbb{R}^n \to \mathbb{R}^n$, is defined by

$$\mathbf{x} = [x_1, \ldots, x_n]^T \mapsto \sigma(\mathbf{x}) = [\sigma_1(\mathbf{x}), \ldots, \sigma(\mathbf{x}_n)] = \left[\frac{\exp(x_1)}{\sum_{i=1}^n \exp(x_i)}, \ldots, \frac{\exp(x_n)}{\sum_{i=1}^n \exp(x_i)} \right]^T. \tag{210}$$

Define the following softmax-cross-entropy loss

$$\ell(\mathbf{y}, \hat{\mathbf{y}}) := -\sum_{i=1}^{\kappa} \mathbf{1}(\mathbf{y} = \text{class } i) \log \sigma_i(\hat{\mathbf{y}}) \tag{211}$$

for each $\mathbf{y}, \hat{\mathbf{y}} \in \mathbb{R}^n$.

(i) Show that

$$\frac{\partial}{\partial x_j} \sigma_i(\mathbf{x}) = \sigma_i(\mathbf{x}) \left(\mathbf{1}(i = j) - \sigma_j(\mathbf{x}) \right). \tag{212}$$

(ii) Show that

$$\frac{\partial}{\partial \hat{\mathbf{y}}} \ell(\mathbf{y}, \hat{\mathbf{y}}) = \sigma(\hat{\mathbf{y}}) - \mathbf{y}. \tag{213}$$

Exercise 3.2.4. Formulate an algorithm that computes the gradient of a general L-layer neural network. You may need to extend the error backpropagation algorithm for 2-layer NN we discussed above to the general L-layer case and show its correctness.

3. Examples of 2-layer FFNN

Example 3.3.1. Consider a two-layer neural network with the following choices:

Loss function: $\ell_s(\mathbf{y}_s, \hat{\mathbf{y}}(\mathbf{x}_s; \mathbf{w})) = (1/2)\|\mathbf{y}_s - \hat{\mathbf{y}}(\mathbf{x}_s; \mathbf{w})\|_F^2$ (Square loss)

Activation functions: $\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$ for the hidden layer and sigmoid $\sigma(x)$ for the output layer.

Then the total loss function is given by

$$\nabla \ell_{NN}(\mathbf{w}) = \frac{1}{2} \sum_{s=1}^{n} \|\mathbf{y}_s - \hat{\mathbf{y}}(\mathbf{x}_s; \mathbf{w})\|_F^2. \tag{214}$$

After forward-propagating an input $\mathbf{x}_s \in \mathbb{R}^N$, the hidden states $\mathbf{z}_s \in \mathbb{R}^M$ and the output $\hat{\mathbf{y}}_s = \hat{\mathbf{y}}(\mathbf{x}_s; \mathbf{w}) \in \mathbb{R}^\kappa$ are given by

$$\mathbf{z}_s = \tanh(\mathbf{x}_s^T \mathbf{W}^{(1)}), \qquad \hat{\mathbf{y}}_s = \sigma(\mathbf{z}_s^T \mathbf{W}^{(2)}). \tag{215}$$

For backpropagation, the errors at the output nodes are given by

$$\frac{\partial \ell_s}{\partial (\hat{\mathbf{y}}_s)_q} = \left((\hat{\mathbf{y}}_s)_q - (\mathbf{y}_s)_q \right), \tag{216}$$

Hence, from (206), the errors at the hidden nodes are given by

$$\frac{\partial \ell_s}{\partial (\mathbf{z}_s)_j} = \sum_{q=1}^{\kappa} \frac{\partial \ell_s}{\partial (\hat{\mathbf{y}}_s)_q} \left((\hat{\mathbf{y}}_s)_q (1 - (\hat{\mathbf{y}}_s)_q) \right) w_{jq}^{(2)}, \tag{217}$$

where we have used that $\sigma'(x) = \sigma(x)(1 - \sigma(x))$.

Now from (208), we get

$$\frac{\partial \ell_s}{\partial w_{jq}^{(2)}} = \frac{\partial \ell_s}{\partial (\hat{\mathbf{y}}_s)_q} \left((\hat{\mathbf{y}}_s)_q (1 - (\hat{\mathbf{y}}_s)_q) \right) (\mathbf{z}_s)_j. \tag{218}$$

On the other hand, note that $\tanh'(x) = 1 - \tanh^2(x)$ so

$$\tanh'(\mathbf{x}_s^T \mathbf{W}^{(1)}[:, j]) = 1 - [(\mathbf{z}_s)_j]^2. \tag{219}$$

Then (209) gives

$$\frac{\partial \ell_s}{\partial w_{ij}^{(1)}} = \frac{\partial \ell_s}{\partial (\mathbf{z}_s)_j} \left(1 - [(\mathbf{z}_s)_j]^2\right) (\mathbf{x}_s)_i. \tag{220}$$

Lastly, the derivatives of the total loss ℓ_{NN} with repsect to the neural weights are given by summing up the per-sample derivatives above over s in the selected minibatch $I \subseteq \{1, \dots, n\}$. ▲

Exercise 3.3.2 (2-layer FFNN on MNIST)**.** Below we show the train and test accuracy of classifying MNIST dataset (digits 0 through 4) using the 2-layer FFNN discussed in Example 3.3.1. Here the accuracy for multiclass classification is computed by the total number of correctly classified examples divided by the total number of examples. (Ref: Jupyter Notebook)

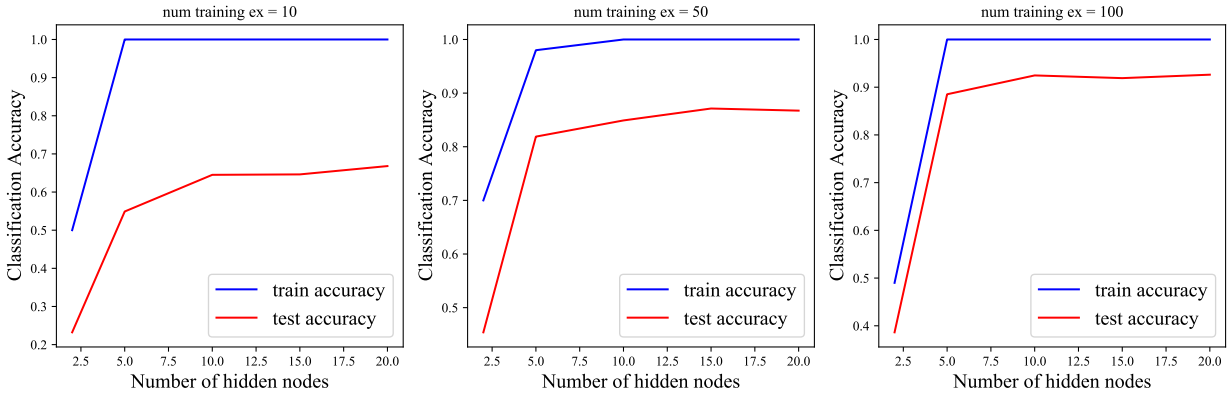


FIGURE 34. Classification accuracy of 2-layer FFNN for MNIST data with digits from 0 up to 4. The number of hidden nodes are varied within $\{2, 5, 10, 15, 20\}$ and the number of training examples are varied within $\{10, 50, 100\}$.

(i) What can you conclude from Figure 34? Optimal number of hidden nodes for each size of training set? Overfitting in terms of the number of hidden nodes and the size of training set?

(ii) Make similar plots for (1) digits 5 through 9 and (2) all digits while keeping other parameters the same. (You may increase the number of iterations for training for classifing all digits.) Do your observations in (ii) still hold in these experiments?

Exercise 3.3.3. Consider the following L_2-regularized 2-layer neural network training problem:

$$\hat{\mathbf{w}} := \underset{\mathbf{w}=[\mathbf{W}^{(1)}, \mathbf{W}^{(2)}]}{\operatorname{argmin}} \left[\ell_{NN}(\mathbf{w}; \lambda) := \left(\sum_{s=1}^{n} \ell(\mathbf{y}_s, \hat{\mathbf{y}}(\mathbf{x}_s; \mathbf{w})) \right) + \lambda \left(\|\mathbf{W}^{(1)}\|_F^2 + \|\mathbf{W}^{(2)}\|_F^2 \right) \right], \tag{221}$$

where $\lambda \geq 0$ is a L_2-regularization constant. This will penalize the weight matrices $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}$ to have large L_2-norm.

(i) How does this change the training algorithm? Derive the gradient of the above loss function and formulate the modified training algorithm.

(ii) Modify the Multilayer Perceptron implementation `MLP` in Jupyter Notebook so that it has the option to use L_2-regularization. Reproduce Figure 34 with varing levels of regularization. Report a reaonable value of the regularization constant for the experiment in Figure 34.

Exercise 3.3.4. Implement the two-layer neural network discussed in Example 3.3.1 on your own, where the activation function in the output layer should allow the identity function in order for applying it on regression problems. Using your code, reproduce a figure similar to Figure 35 below. For each $n = 6, 10$ data points in the sinusoidal data set (see (15)), what do you think is the best value of M?

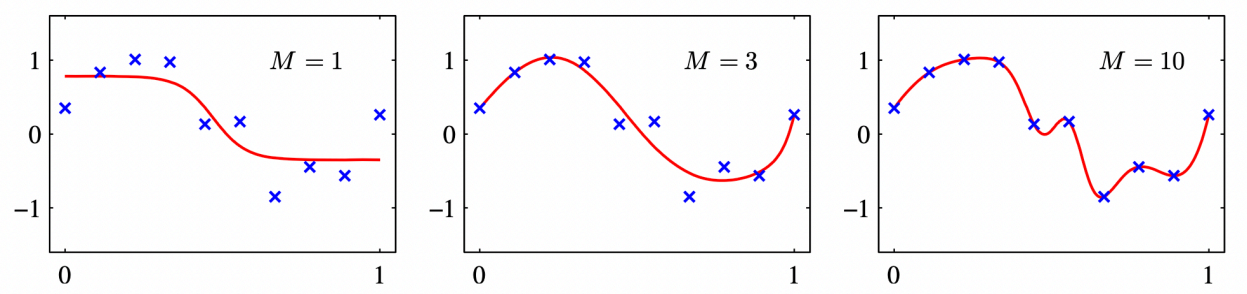


FIGURE 35. Examples of two-layer networks trained on 10 data points drawn from the sinusoidal data set (see Sec. 2.2). The graphs show the result of fitting networks having $M = 1$, 3 and 10 hidden units, respectively, by minimizing a sum-of-squares error function using a SGD algorithm. Figure excerpted from [Bis06].

4. Convolutional Neural Networks

4.1. Introduction. In this section, we discuss one of the most successful and architecture of feedforward deep neural networks for image classification, namely, the *convolutional neural networks* (CNN, or ConvNet). A few highlighing features of CNN are as follow:

1. CNN is a 'localized' version of the fully connected deep feedforward neural network (inspired by how animal visual cortex works).
2. CNN learns patterns in images that are 'shift-invariant', meaning that the location of a certain geometric pattern in the image matters less[11].
3. CNN not only to recognize patterns from images, but also to abstract such patterns. (e.g., first layer learns edges or colors, next layer learns larger curves and shapes, and so on.)

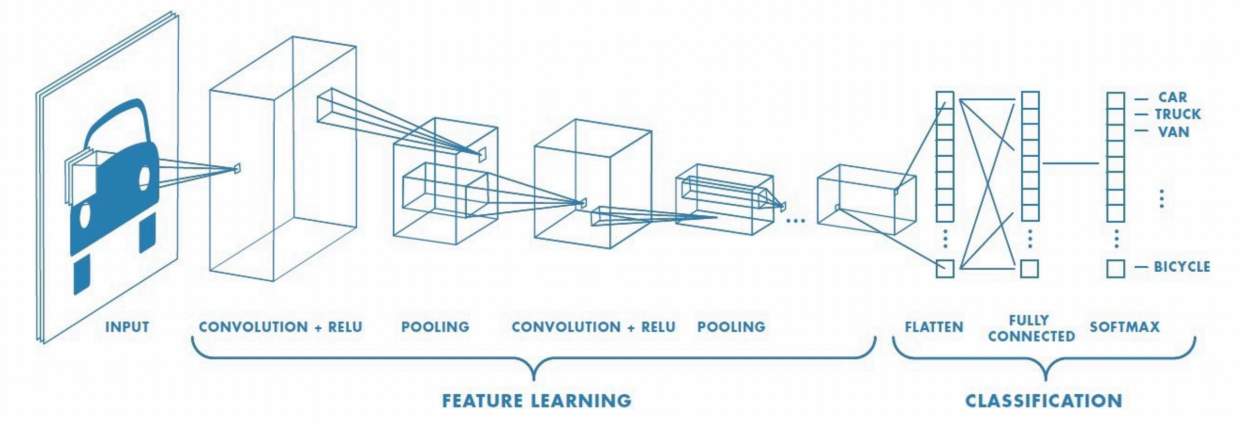


FIGURE 36. Architecture of a Convolutional Neural Network for image classification. Figure by S. Saha link

We first discuss a motivating example. Consider the now-familar MNIST handwritten digits image dataset. We have seen that regardless of what algorithm we use for classifying the digits, we usually get very high accuracy (e.g., see Figure 16, 17, 22 for binary classification with LR and PR, Figure 19 for multiclass classification with MLR). Since the images in the MNIST dataset are so well alinged within 28×28 frame, it only means that we can classify the images to the correct classes against variation within each class under such a good alignment. So what if the images are not perfectly alinged within the same frame, as in Figure 37? There, we 'padded' additional zeros aound the orignal image to purturb the

[11] Think about whether the usual FFNN or multiclass logistic regression have such abilities

alignment. In other words, we embedded each 28×28 image inside 48×48 zero matrices at uniformly randomly chosen locations. Do you think the classification algorithms we have developed so far would still work well after this perturbation?[12]

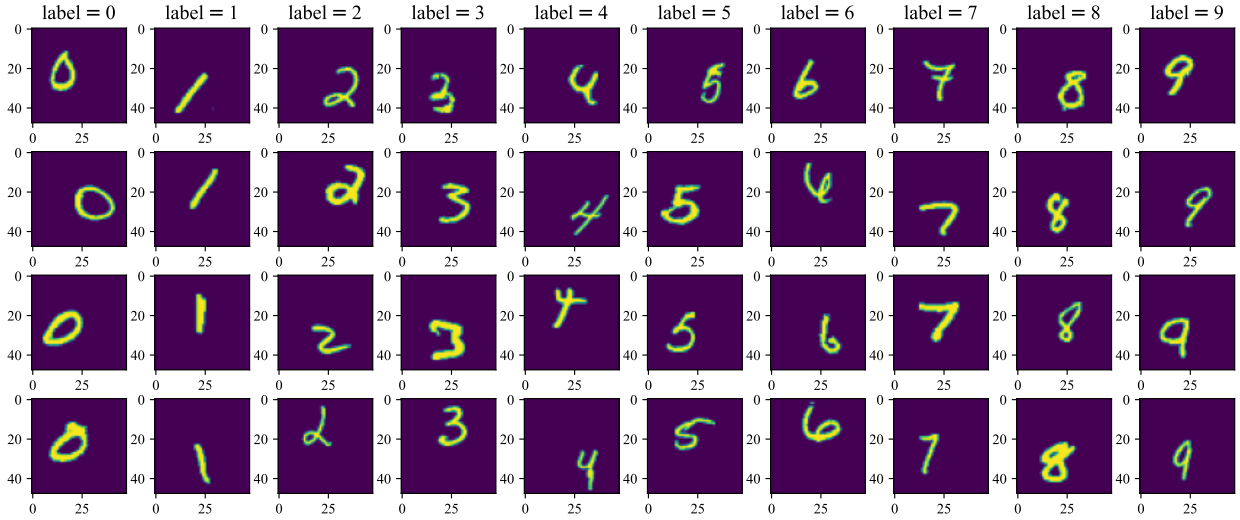


FIGURE 37. Examples of MNIST dataset of handwritten digits by Yann LeCun, Corinna Cortes, and Christopher J.C. Burges with random padding. Each 28×28 image is superimposed within a 48×48 zero matrix at uniformly chosen random location.

In Figure 38, we show the training and testing accuracies of using 2-layer neural network with 10 hidden nodes for classifing MNIST (five digits from 0 through 4) with random padding. We vary the 'thickness' of the padding, namely, when the thickness is r, it means that each 28×28 image is randomly embedded inside a zero matrx of shape $(28+r) \times (28+r)$. As we can see in Figure 38, the training accuracies are close to 1 but the testing accurcies drop quickly as we increase the padding thickness. This means our model overfits to some non-shift-invariant features. Can we overcome this shortcoming by increasing the number of layers in our model? Not quite, as we see in Figure 39 for 3-layer FFNN.

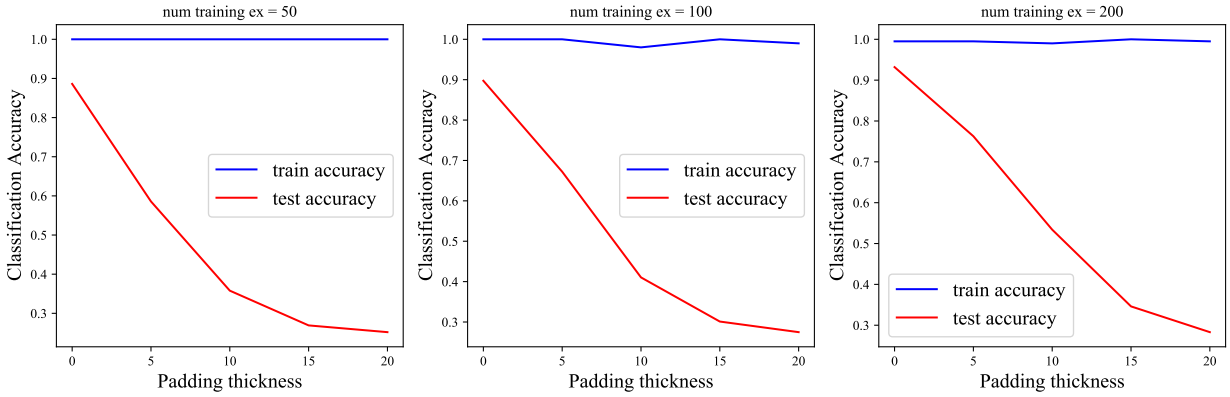


FIGURE 38. Classification accuracy of 2-layer FFNN for MNIST data with digits from 0 up to 4 with random padding with varying thickness. The number of hidden nodes is 10. When the thickness is r, it means that each 28×28 image is randomly embedded inside a zero matrx of shape $(28+r) \times (28+r)$.

[12] Certainly our brain doesn't care the location of the digits within a larger image.

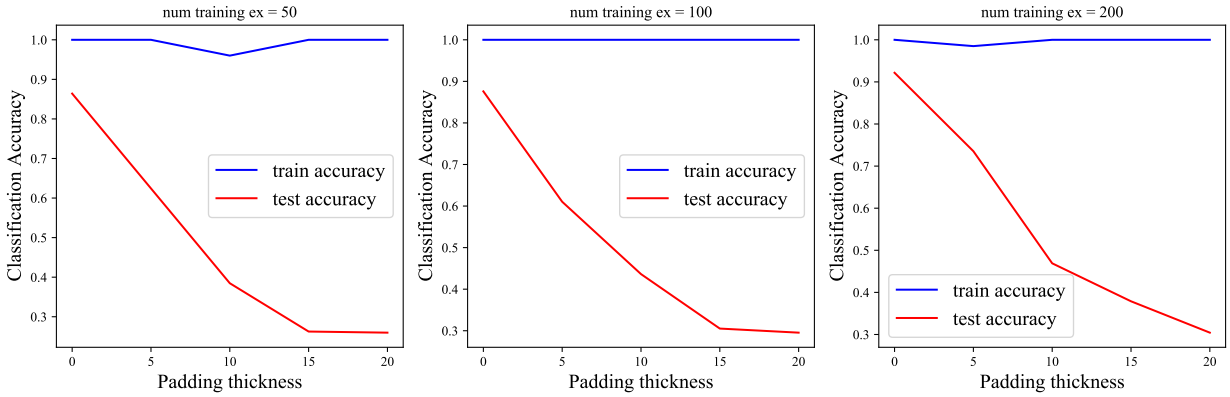


FIGURE 39. Classification accuracy of 3-layer FFNN for MNIST data with digits from 0 up to 4 with random padding with varying thickness. The number of hidden nodes are 10 for each hidden layer. When the thickness is r, it means that each 28×28 image is randomly embedded inside a zero matrx of shape $(28+r) \times (28+r)$.

Exercise 3.4.1. Reproduce figures similar to Figure 38 using (1) multiclass logistic regression; and (2) naive Bayes classifier. Conclude that whether these algorithms are able to learn shift-invariant features. (Ref: Jupyter Notebook – 'Classifying non-aligned MNIST images')

4.2. Model formulation. In order to formulate CNN, we need to introduce (1) convolutional layer and (2) pooling layer and how they work. Each 'layer' in a CNN is the following composition: convolutional with set of filters + ReLU + pool. Every time we complete such procedure, we learn more abstract and shift-invariant features from the previous layer. (See Figure 40 for an illustration[13])

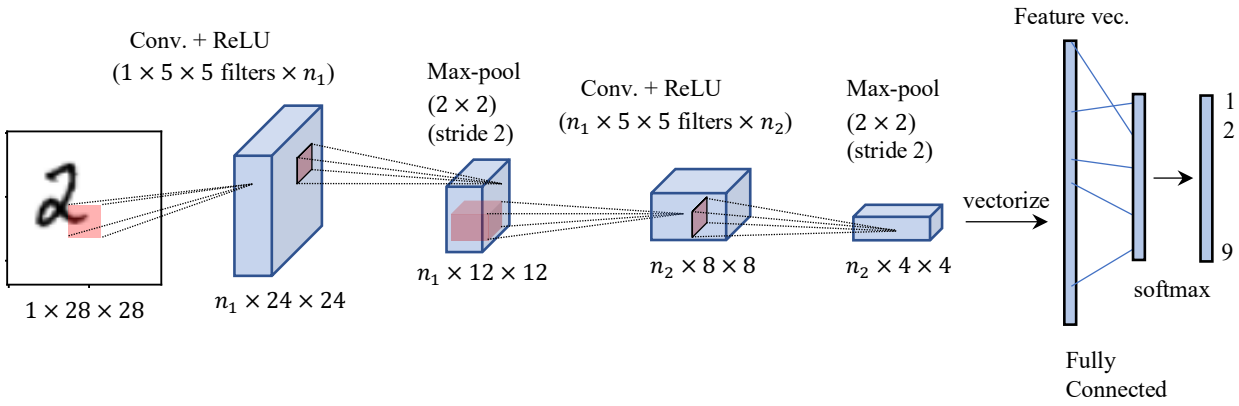


FIGURE 40. An example of CNN architecture for MNIST image classification. Convolution + ReLU activation followed by a max-pull layers are applied twice with to extract shift-invariant feature vector. Pass this into a multiclass logistic regression (fully connected layer + softmax) for classification.

4.2.1. *Convolutional layer.* First we explain how a convolutional layer works in CNN. Recall that in a fully connected neural network, the previous layer's activation (node state), say $\mathbf{z}$, is dot producted with each colums of the following weight matrix to produce the activation of each nodes in the following layer. In CNN, it is useful to note that

$$\text{Weight matrix in a CNN layer} = \text{Set of tensors (filters/kernels) } F_1, \dots, F_m \in \mathbb{R}^{c \times r \times r}, \tag{222}$$

[13]See also 'A Comprehensive Guide to Convolutional Neural Networks' by S. Saha

where the 'tensor depth' c should match the depth of the previous 'image'. For example, color images have shape $3 \times x \times y$ and monochrome images have shape $1 \times x \times y$. Each filter is supposed to capture a particular pattern *everywhere* in the image. We explain this in steps as follow:

1. Say we have a *filter* F of shape $1 \times 3 \times 3$. For an input X of shape $1 \times 3 \times 3$, the convolution $X * F$ of X with filter F is the sum of entrywise product of X with F.

$$\underbrace{\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}}_{X} * \underbrace{\begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}}_{F} = \operatorname{sum}\left(\begin{bmatrix} x_{11}f_{11} & x_{12}f_{12} & x_{13}f_{13} \\ x_{21}f_{21} & x_{22}f_{22} & x_{23}f_{23} \\ x_{31}f_{31} & x_{32}f_{32} & x_{33}f_{33} \end{bmatrix} \right) = \sum_{i,j} x_{ij} f_{ij}. \tag{223}$$

For general depth c, the convolution of a filter F of shape $c \times r \times r$ on an input of shape $1 \times r \times r$ is a scalar given similarly as follow:

$$X * F = \sum_{i,j,k} X[i,j,k]\, F[i,j,k], \tag{224}$$

where the sum is over all entries (i,j,k).

2. If the input X is $1 \times 4 \times 4$, then we can choose four $1 \times 3 \times 3$ *patches*[14] inside X by *sliding*[15] a window (frame) of $1 \times 3 \times 3$. We can convolve each patch with F to get a scalar. So output of $X * F$ is a $1 \times 2 \times 2$ tensor.

$$\underbrace{\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix}}_{X} * \underbrace{\begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}}_{F} = \underbrace{\begin{bmatrix} X[0:3,0:3] * F & X[0:3,1:] * F \\ X[1:,0:3] * F & X[1:,1:] * F \end{bmatrix}}_{\text{In python slicing}}. \tag{225}$$

3. If we have a $c \times r \times r$ filter F and $c \times a \times b$ input X, then the result of convolution $X * F$ with *stride* 1 is a $1 \times (a-r+1) \times (b-r+1)$ tensor:

$$(X * F)[0,j,k] = X[:, i:i+r, j:j+r] * F. \tag{226}$$

Namely, the (i,j) entry of $X * F$ is convolution of the (i,j) patch $X[:, i:i+r, j:j+r] \in \mathbb{R}^{c \times r \times r}$ with F.

4. If we have a $c \times r \times r$ filter F and $c \times a \times b$ input X, then the result of convolution $X * F$ with *stride* s is a $1 \times (\frac{a-r}{s}+1) \times (\frac{b-r}{s}+1)$ tensor:

$$(X * F)[0,j,k] = X[:, is:is+r, js:js+r] * F. \tag{227}$$

Namely, the (i,j) entry of $X * F$ is convolution of the (i,j) patch $X[:, is:is+r, js:js+r] \in \mathbb{R}^{c \times r \times r}$ with F.

4. The result of the convolutional layer with input $X \in \mathbb{R}^{c \times a \times b}$ and filters $F_1, \ldots, F_m \in \mathbb{R}^{c \times r \times r}$ with stride s is

$$X * (\text{filters } F_1, \ldots, F_m) = [X * F_1, \ldots, X * F_m] \in \mathbb{R}^{m \times (\frac{a-r}{s}+1) \times (\frac{b-r}{s}+1)}, \tag{228}$$

which we think of as a 'depth' m image of size $(\frac{a-r}{s}+1) \times (\frac{b-r}{s}+1)$.

See Figure 40 for an illustration.

[14] Square sub-matrices of consecutive rows and colums

[15] We may want to skip every some rows and columns in extracting patches from X to avoid too much overlap between patches and/or reduce the output of convolution more rapidly. The number of rows/columns we slide over is called *stride*. For stride= 1, we sample the maximum number of patches.

4.2.2. *Pooling layer.* In CNN, a *pooling* is used to reduce the size of the previous layer's output by either keeping the most dominant cell value in each patch (called 'max-pool') or averaging every patch into a single scalar (called 'avg-pool'). We introduce these concepts with the following examples.

1. 2×2 pooling on $1\times 2\times 2$ input X:

$$\text{pool}\left(X=\begin{bmatrix}x_{11} & x_{12}\\ x_{21} & x_{22}\end{bmatrix}\right)=\begin{cases}\max(X)=\max\{x_{ij}\,|\,1\le i,j\le 2\} & \text{for max-pool}\\ \text{avg}(X)=(1/4)\sum_{1\le i,j\le 2}x_{ij} & \text{for avg-pool}\end{cases}. \tag{229}$$

2. 2×2 pooling on $1\times 4\times 4$ input X with stride 2:

$$\text{pool}\left(X=\begin{bmatrix}x_{11} & x_{12} & x_{13} & x_{14}\\ x_{21} & x_{22} & x_{23} & x_{24}\\ x_{31} & x_{32} & x_{33} & x_{34}\\ x_{41} & x_{42} & x_{43} & x_{44}\end{bmatrix}\right)=\begin{bmatrix}\text{pool}\begin{pmatrix}x_{11} & x_{12}\\ x_{21} & x_{22}\end{pmatrix} & \text{pool}\begin{pmatrix}x_{13} & x_{14}\\ x_{23} & x_{24}\end{pmatrix}\\ \text{pool}\begin{pmatrix}x_{31} & x_{32}\\ x_{41} & x_{42}\end{pmatrix} & \text{pool}\begin{pmatrix}x_{33} & x_{34}\\ x_{43} & x_{44}\end{pmatrix}\end{bmatrix}\in\mathbb{R}^{1\times 2\times 2}. \tag{230}$$

3. 2×2 pooling on $1\times 4\times 4$ input X with stride 1:

$$\text{pool}\left(X=\begin{bmatrix}x_{11} & x_{12} & x_{13} & x_{14}\\ x_{21} & x_{22} & x_{23} & x_{24}\\ x_{31} & x_{32} & x_{33} & x_{34}\\ x_{41} & x_{42} & x_{43} & x_{44}\end{bmatrix}\right)=\left[\text{pool}\left(X[i:i+2,j:j+2];0\le i,j\le 2\right)\right]\in\mathbb{R}^{1\times 3\times 3}. \tag{231}$$

4. 2×2 pooling on $c\times a\times b$ input X with stride 1:

$$\text{pool}(X)=\left[\text{pool}\left(X[d,i:i+2,j:j+2];0\le d<c,0\le i\le a-2,0\le j\le b-2\right)\right] \tag{232}$$

$$\in\mathbb{R}^{c\times(a-1)\times(b-1)}. \tag{233}$$

5. $r\times r$ pooling on $c\times a\times b$ input X with stride s:

$$\text{pool}(X)=\left[\text{pool}\left(X[d,is:is+r,j:j+r];0\le d<c,0\le i\le\tfrac{a-r}{s},0\le j\le\tfrac{b-r}{s}\right)\right] \tag{234}$$

$$\in\mathbb{R}^{c\times(\frac{a-r}{s}+1)\times(\frac{b-r}{s}+1)}. \tag{235}$$

Exercise 3.4.2. Consider the CNN architecture shown in Figure 40. Suppose we use the following general choices of convolutional and pooling layers:

Conv1: n_1 filters of shape $1\times r\times r$ with stride s_f
Conv2: n_2 filters of shape $n_1\times r\times r$ with stride s_f
maxpool (both): $d\times d$ maxpool with stride s_p

Also assume that input image has dimension $1\times a\times b$. Then compute the dimension of the feature vector that goes into the first dense layer as a function of n_1,n_2,r,d,s_f,s_p,a,b.

4.3. Training. Training a CNN is similar to that for the feedforward neural networks. Namely, we use stochastic gradient descent in order to find the optimal parameters (that is, all filters, weight matrices, and biases). In order to compute the gradients of the parameters, we first backpropagate a chosen training image, compute the error at the output layer, and then backpropagate it all the way down to the input layer. This will give the gradients of the loss with respect to node states, which we can use to compute the gradients of filters and weight matrices by chain rule.

In order to execute backpropagation for CNN, we need to perform the followng four kinds of backpropagations:

1. Backprop through a dense layer — Same as in FFNN
2. Backprop through a maxpool layer — Assign errors to the 'winning pixels'
3. Backprop through a ReLU layer — Set the errors zero at coordinates where the input activation is negative
4. Backprop through a convolutional layer –Details omited. (Ref: Backprop. in CNN)

Computation of gradients for training a CNN

1. Forward propagate the input all the way to the output layer and compute the activation and state of every node.
2. Compute the errors at the output nodes using (202)-(205).
3. Backprogagate the errors in the output layer to the hidden layer and compute the errors at hidden nodes, using (206).

4.4. Example: MNIST + padding. Next, we give some examples of using CNN for MNIST image classification. We are going to use MNIST with random padding as shown in Figure 37. The architecture of CNN we use here is as follow:

$$\text{Input} \to \underbrace{\text{Conv1}}_{\text{filter1}\in\mathbb{R}^{n_1\times 1\times 5\times 5}} + \text{ReLU} + \text{maxpool} \to \underbrace{\text{Conv2}}_{\text{filter1}\in\mathbb{R}^{n_2\times n_1\times 5\times 5}} + \text{ReLU} + \text{maxpool} \tag{236}$$

$$\to \underbrace{\text{Dense1}}_{W1} + \text{ReLU} \to \underbrace{\text{Dense2}}_{W2} + \text{Softmax} \tag{237}$$

$$\to \text{Output} \tag{238}$$

The paramters we need to optimize are n_1 filters of shape 1×5 in the first convolutional layer, n_2 filters of shape $n_1\times 5\times 5$ in the second convolutional layer, and the two weight matrices for the dense layers. For all pooling layer, we use 2×2 maxpool with stride 2. All convolutions use stride 1.

In Figure 41, we show the training and testing accuracies of using the above CNN in (238) for classifing MNIST (five digits from 0 through 4) with random padding. Again we vary the thickness of the padding, namely, when the thickness is r, it means that each 28×28 image is randomly embedded inside a zero matrx of shape $(28+r)\times(28+r)$. The overall trend is similar as for FFNN in Figure 38: The training accuracies are close to 1 but the testing accurcies drop quickly as we increase the padding thickness. However, the test accuracy is about 10% higher with 100 training examples and 20% higher with 200 training examples than those for FFNN. This means that the above CNN is able to learn shift-invariant features to some extent, at least much better than FFNN.

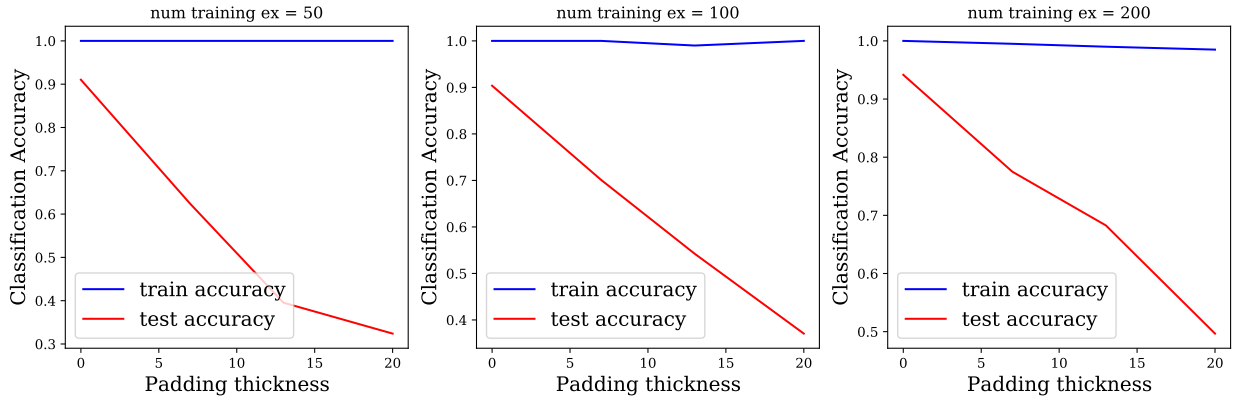


FIGURE 41. Classification accuracy of 3-layer FFNN for MNIST data with digits from 0 up to 4 with random padding with varying thickness. The number of hidden nodes are 10 for each hidden layer. When the thickness is r, it means that each 28×28 image is randomly embedded inside a zero matrx of shape $(28+r)\times(28+r)$.

Exercise 3.4.3. Reproduce Figures 39 and 41 with larger number of training examples: 300, 500, 1000. Compare the performance of FFNN and CNN using your plots. (Ref: Jupyter notebook on FFNN and on CNN)

Exercise 3.4.4. Produce a plot similar to Figure 34 for the CNN model in (238), where the horizontal axis correspond to the shared number $n_1 = n_2 \in \{2,5,10,20\}$ of filters in each convolutional layer. Use

padding thickness 10 and keep all the other settings the same as in Figure 41. What do you think the optimal number of filters to use is? (Ref: Jupyter notebook on CNN)

In Figure 42, we show 8 filters of shape $1 \times 5 \times 5$ in the first convolutional layer that are learned for a binary MNIST classification (digits 0 and 1). Can you imagine that kind of pattern that each filter is suppose to pick up from the image?

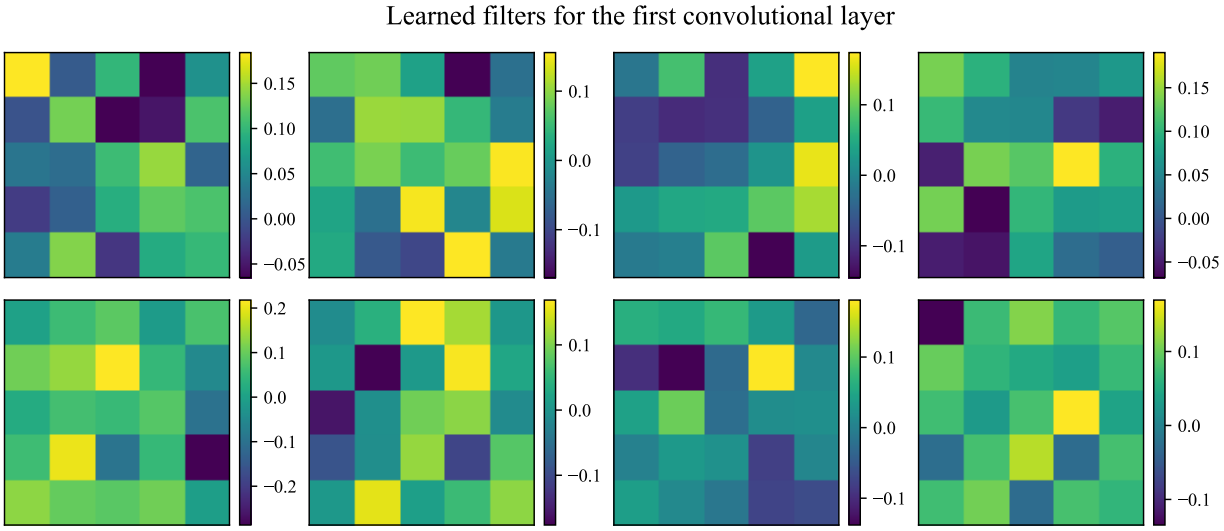


FIGURE 42. 8 filters learned for a binary MNIST classification (digits 0 and 1) using the CNN model in (238).

Exercise 3.4.5. Train the CNN in (238) for MNIST classification with digits 0 and 1. Pick two images of each digit. Using the trained model, plot the image after the first and the second convolutional layers for each of the four chosen images. (Ref: Jupyter notebook on CNN) (Hint: Modify the 'predict' function in src/CNN.py so that it also returns outputs of each convolutional layer.)

CHAPTER 4

Matrix Factorization and Dictionary Learning

In this chaper, we introduce the problem of matrix factorization, which is a powerful unifying framework for a number of unsupervised machine learning problems such as dimension reduction or dictionary learning. Two main techniques we will be learning are *nonnegative matrix factorization* (NMF) and *principal component analysis* (PCA).

1. Introduction

1.1. Dictionary Learning. *Dictionary-learning* algorithms are machine-learning techniques that are able to learn interpretable latent structures of complex data sets and are applied regularly in the data analysis of text and images. Such algorithms usually consist of two steps. First, one samples a large number of structured subsets of a data set (e.g., square patches of an image or collections of a few sentences of a text); Second, one finds a set of basis elements such that taking a *nonnegative* linear combination of them can successfully approximate each of the sampled mesoscale patches. Such a set of basis elements is called a *dictionary*, and one can interpret each basis element as a latent structure of the data set. As an example, consider the image of the artwork CYCLE by M. C. Escher in Figure 43**a**. We first sample 10,000 square patches of 21×21 pixels, and we then use a nonnegative matrix factorization (NMF) [LS99] algorithm to find a dictionary with $r = 25$ square patches (see Figure 43**a**). Each element of the learned dictionary describes a latent shape in the image.

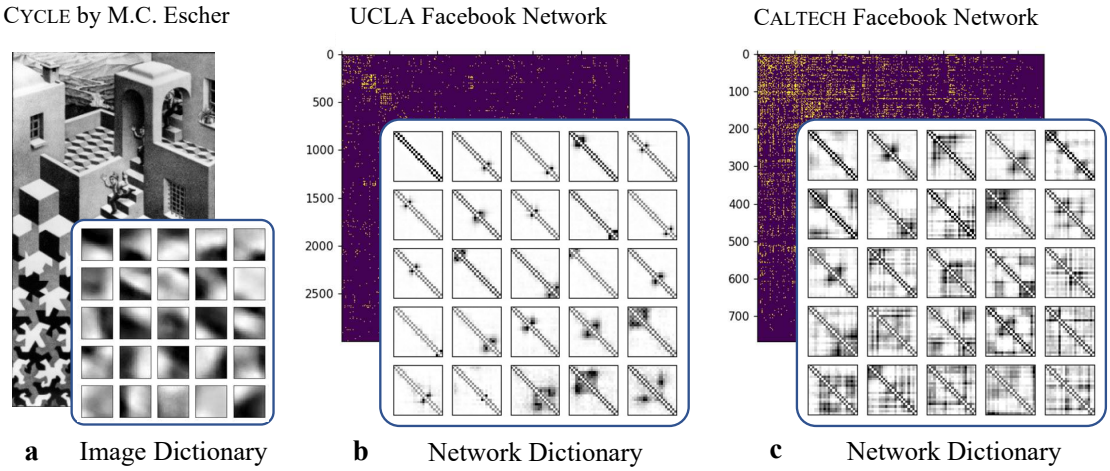


FIGURE 43. Illustration of matrix factorization. Each column of the data matrix is approximated by the linear combination of the columns of the dictionary matrix with coefficients given by the corresponding column of the code matrix. Figure excerpted from [LKVP21].

1.2. Matrix Factorization. *Matrix factorization* is one of the fundamental tools in dictionary learning problems. Given a large data matrix $\mathbf{X}$, can we find some small number of "dictionary vectors" so that we can represent each column of the data matrix as a linear combination of dictionary vectors? More precisely, given a data matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$ and a rank parameter $r \in \mathbb{N}$, we wish to factorize X into the

product of $\mathbf{W} \in \mathbb{R}^{d\times r}$ and $\mathbf{H} \in \mathbb{R}^{r\times n}$ by solving the following optimization problem

$$\text{(Matrix Factorization)} \qquad \inf_{\mathbf{W}\in\mathbb{R}^{d\times r}, \mathbf{H}\in\mathbb{R}^{r\times n}} \|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2. \tag{239}$$

Here $\mathbf{W}$ is called the *dictionary* and $\mathbf{H}$ is the *code* of data $\mathbf{X}$ using dictionary $\mathbf{W}$. A solution of such matrix factorization problem is illustrated in Figure 44.

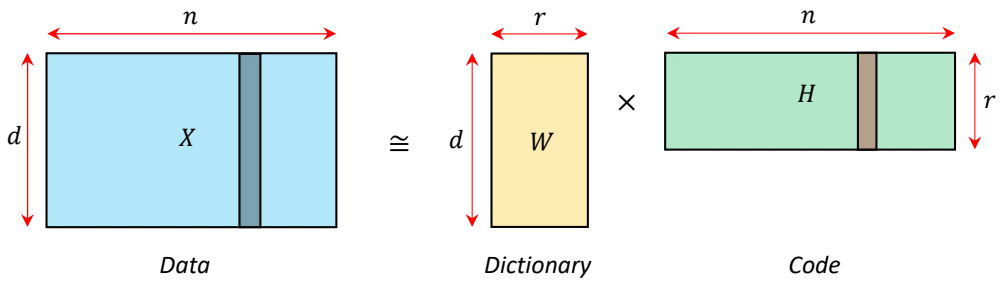


FIGURE 44. Illustration of matrix factorization. Each column of the data matrix is approximated by the linear combination of the columns of the dictionary matrix with coefficients given by the corresponding column of the code matrix. Figure excerpted from [LNB20].

Recall the method of least squared we studied for polynomial regression: (See also Exercise 1.2.1).

$$\hat{\mathbf{H}} = \underset{\mathbf{H}\in\mathbb{R}^{r\times n}}{\arg\min} \|\underbrace{\mathbf{X}}_{\text{given}} - \underbrace{\mathbf{W}\,\mathbf{H}}_{\text{given TBD}}\|_F^2 + \lambda\|\underbrace{\mathbf{H}}_{\text{TBD}}\|_F^2. \tag{240}$$

One can think of (240) as a matrix factorization problem where the first factor (dictionary) is known and we are only looking for the code $\mathbf{H}$. Becase of this reason, we also call (240) as a *coding* problem, which is also called a *sparse coding* problem if the regularizer λ is positive[1].

While ithe optimization problem 239 is an *unconstrained* problem in the sense that we are free to choose any dictionary $\mathbf{W} \in \mathbb{R}^{d\times r}$ and code $\mathbf{H} \in \mathbb{R}^{r\times n}$, We will be more interested in its *constrained* versions, in particular using nonnegativity constraints. Consider the following modifications:

$$\text{(Dictionary Learning)} \qquad \inf_{\mathbf{W}\in\mathbb{R}^{d\times r}, \mathbf{H}\in\mathbb{R}^{r\times n}_{\geq 0}} \|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2, \tag{241}$$

$$\text{(NMF)} \qquad \inf_{\mathbf{W}\in\mathbb{R}^{d\times r}_{\geq 0}, \mathbf{H}\in\mathbb{R}^{r\times n}_{\geq 0}} \|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2. \tag{242}$$

In (241), we imposed an additional constraint that the code $\mathbf{H} \in \mathbb{R}^{r\times n}$ should only take nonngative entries. This allows us to interpret the columns of dictionary $\mathbf{W}$ as 'features' that directly contribute to composing the columns of $\mathbf{X}$ (samples)[2]. We can go even further and also impose the same nonnegativity constraint on the dictioanry matrix $\mathbf{W} \in \mathbb{R}^{r\times n}$ as in (242). This problem is called *nonnegative matrix factorization* (NMF), which is an extremely popular technique in modern machine learning for its high interpretibility and mathematical simplicity.

1.3. Applications in Image processing. Before we discuss how to solve matrix factorization problems, we give some examples of matrix factorization in the context of image processing through image dictionary learning.

Suppose we are given an image, and we are intrested in the following *unsupervised problem* of learning 'essential features in all of its $k \times k$ patches'. In other words, we would like to know what important features are there that if we cut out a large number of $k \times k$ square patches from the original image.

[1] It is more effective to use an L_1-regularizer $\lambda\|\mathbf{H}\|_1$ for the purpose of sparse coding. This is also closely related to (Lagrangian version of) the popular algorithm of LASSO in statistics.

[2] A general formulation of dictionary learning may also impose a convex constraint $\mathscr{C} \subseteq \mathbb{R}^{d\times r}$ for dictionary $\mathbf{W}$ along with the nonnegativity constraint on the code $\mathbf{H}$.

Of course, still this problem is not precisely defined. Even though we can state a precise optimization problem for it, let's instead describe the algorithm first and then see what exactly we are solving.

As we have briefly mentioned in Section (1.1), for dictionary learning we first do patch sampling and then feature extraction. For patch sampling, we get a large number N (e.g., N =10K) of $k \times k$ (e.g., 21×21) square patches from the original image. We will vectorize each such patch as a k^2 dimensional column vector and form a data matrix $\mathbf{X}$ of shape $k^2 \times N$. We then use an algorithm[3] for NMF to get a dictionary matrix $\mathbf{W} \in \mathbb{R}^{k^2 \times r}$ and code matrix $\mathbf{H} \in \mathbb{R}^{r \times N}$ with $r = 25$. Each of the 25 columns of $\mathbf{W}$ are vectorized versions of the extracted features or 'basis shapes'. Reshaping these k^2 dimensional column vectors into $k \times k$ square form, we obtain the image dictionary of size 25, as shown in Figure 45.

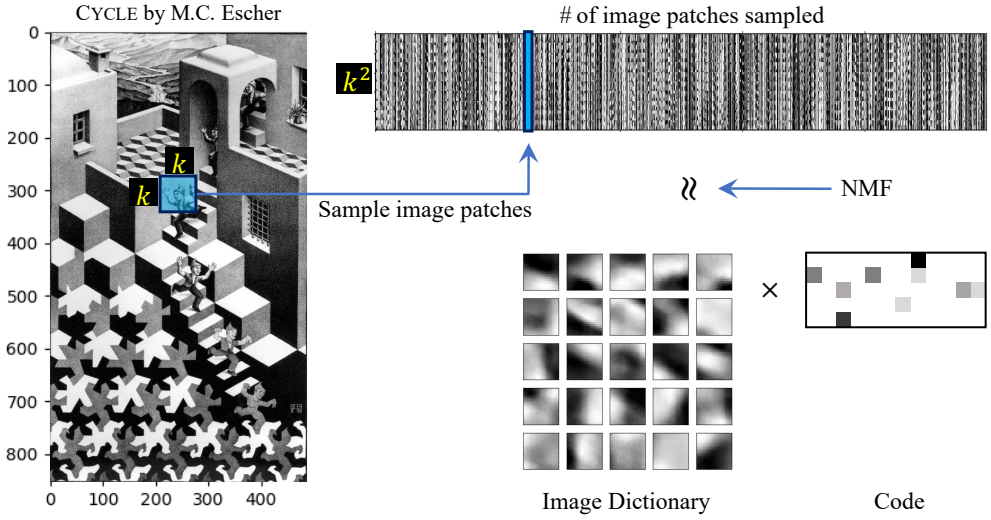


FIGURE 45. Examples of image dictionary learning by patch sampling and NMF.

After we learn a set of dictionary images from a given image, we can use them to reconstruct the original image. Moreover, if the given image is corrupted with some noise, then this reconstruction using learned image dictionary can be used for image denoising. Figure 46 gives an example of image dictionary learning and image denoising with noise separation. We omit details here.

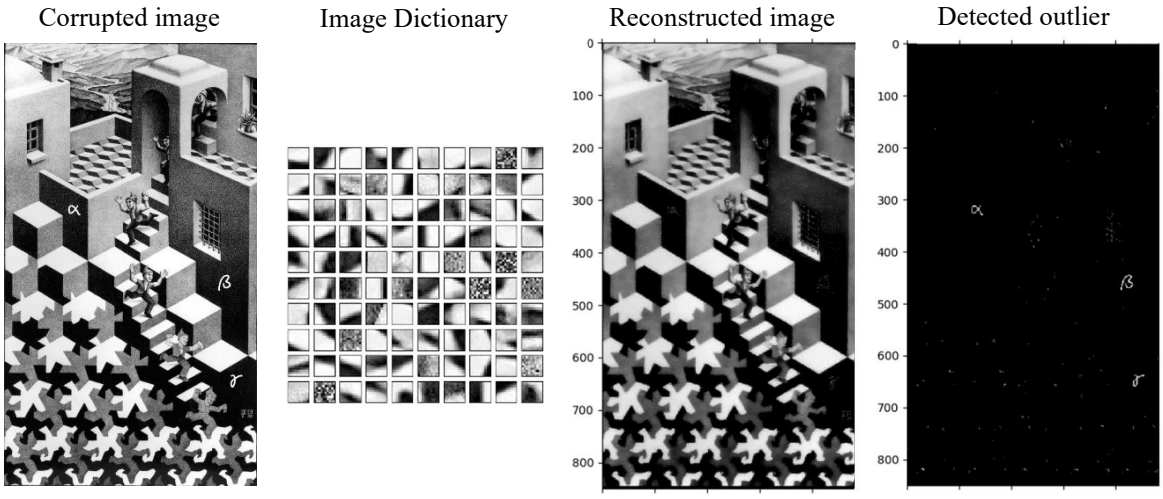


FIGURE 46. Examples of image dictionary learning by patch sampling and NMF. The learned image dictionary is used for denoising the original image corrupted with noises 'α', 'β', and 'γ'.

[3] We will discuss algorithms for solving matrix factorization problems later

2. Algorithms for Matrix Factorization

In this section, we discuss algorithms for solving matrix factorization problems. First, fix a data matrix $\mathbf{X} \in \mathbb{R}^{d\times n}$ and integer $r \geq 1$. Define the following loss function

$$\ell(\mathbf{W},\mathbf{H}) := \|\mathbf{X}-\mathbf{W}\mathbf{H}\|_F^2. \tag{243}$$

We would like to minimize the above loss function jointly in $\mathbf{W} \in \mathbb{R}^{d\times r}$ and $\mathbf{H} \in \mathbb{R}^{r\times n}$, possibly with convex constraints (e.g., nonnegativitity) depending on the situation.

Exercise 4.2.1. Show the following.

(i) Show that the function $\mathbf{H} \mapsto \|\mathbf{X}-\mathbf{W}\mathbf{H}\|_F^2$ is convex.

(ii) Show that the function $\mathbf{W} \mapsto \|\mathbf{X}-\mathbf{W}\mathbf{H}\|_F^2 = \|\mathbf{X}^T-\mathbf{H}^T\mathbf{W}^T\|_F^2$ is convex.

(iii) Show that the function $(\mathbf{W},\mathbf{H}) \mapsto \|\mathbf{X}-\mathbf{W}\mathbf{H}\|_F^2$ is non-convex.

According to Exercise 4.2.1, this is a non-convex optimization problem so it is difficult to find its global minima. However, if we fix one of the factors, then ℓ is convex with respect to the other factor (see Exercise 4.2.1), which is in fact a least squares problem (see Exercise 1.2.1). Hence one can iteratively fix one factor and optimize the other, each time solving a least squares problem. This is called *Alternating Least Squares* (ALS) algorithm for matrix factorizaiton, which we state in Algorithm 2 for NMF.

Algorithm 2 Alternating Least Squares for Matrix Factorization

Input: $\mathbf{X} \in \mathbb{R}^{d\times n}_{\geq 0}$ (data matrix); $r \geq 1$ (rank parameter); $[\mathbf{W}_0,\mathbf{H}_0] \in \mathbb{R}^{d\times r}_{\geq 0} \times \mathbb{R}^{r\times n}_{\geq 0}$ (initial estimate); N (number of iterations)
  for $n = 1,\dots,N$ **do**:
    $\mathbf{H}_n \leftarrow \arg\min_{\mathbf{H}\in\mathbb{R}^{r\times n}_{\geq 0}} \|\mathbf{X}-\mathbf{W}_{t-1}\mathbf{H}\|_F^2$
    $\mathbf{W}_n \leftarrow \arg\min_{\mathbf{W}\in\mathbb{R}^{d\times r}_{\geq 0}} \|\mathbf{X}-\mathbf{W}\mathbf{H}_n\|_F^2$
  end for
output: $[\mathbf{W}_N,\mathbf{H}_N]$.

The ALS algorithm in Algorithm 2 is a special case of *block coordinate descent* (BCD) algorithms, which is to iteratively optimize the objective function with respect to a block coordinate (group of individual coordinates). The general algorithm is stated in Algorithm 3. For more backgrounds and its convergence properties, see [Lyu20].

Algorithm 3 Block Coordinate Descent

Requirement: $\boldsymbol{\theta} = [\theta_1,\dots,\theta_m] \mapsto f(\theta_1,\dots,\theta_m)$ is convex in each block-coordinate θ_i, $i = 1,\dots,m$.
Input: $\boldsymbol{\theta}_0 = (\theta_0^{(1)},\cdots,\theta_0^{(m)}) \in \Theta^{(1)}\times\cdots\times\Theta^{(m)}$ (initial estimate); N (number of iterations)
  for $n = 1,\dots,N$ **do**:
    Update estimate $\boldsymbol{\theta}_n = [\theta_n^{(1)},\cdots,\theta_n^{(m)}]$ by
      For $i = 1,\cdots,m$ **do**:

$$\theta_n^{(i)} \in \arg\min_{\theta\in\Theta^{(i)}} f\left(\theta_n^{(1)},\cdots,\theta_n^{(i-1)},\theta,\theta_{n-1}^{(i+1)},\cdots,\theta_{n-1}^{(m)}\right); \tag{244}$$

      end for
  end for
output: $\boldsymbol{\theta}_N$

Exercise 4.2.2 (Nonnegative Coding with L_1 regularizer)**.** Suppose we have matrices $\mathbf{X} \in \mathbb{R}^{d\times n}$ and $\mathbf{W} \in \mathbb{R}^{d\times r}$. We seek to find a matrix $\hat{\mathbf{H}} \in \mathbb{R}^{r\times n}_{\geq 0}$ where

$$\hat{\mathbf{H}} = \arg\min_{\mathbf{H}\in\mathbb{R}^{r\times n}_{\geq 0}} \left(\ell(\mathbf{H}) := \|\mathbf{X}-\mathbf{W}\mathbf{H}\|_F^2 + \lambda_1\|\mathbf{H}\|_1 + \lambda_2\|\mathbf{H}\|_F^2\right). \tag{245}$$

Here $\lambda \geq 0$ is called the L_1-regularization parameter. (This is an instance of *constrained* quadratic optimization problem.) (c.f. The use of L_1 regularizer instead of L_2 as in Exercise 1.2.1 is more effective to increase the 'sparsity' of the solution, meaning that it will have less nonzero components.)

(i) Show that (use Exercise 1.2.2), denoting $\operatorname{sgn}(\mathbf{H}) = \mathbf{1}(\mathbf{H} \geq 0) - \mathbf{1}(\mathbf{H} < 0)$,

$$\frac{\partial}{\partial \mathbf{H}} \ell = 2(\mathbf{W}^T\mathbf{W}\mathbf{H} - \mathbf{W}^T\mathbf{X}) + \lambda_1 \operatorname{sgn}(\mathbf{H}) + 2\lambda_2 \mathbf{H}, \qquad \frac{\partial^2}{\partial \mathbf{H}^2} \ell = 2(\mathbf{W}^T\mathbf{W} + \lambda_2 \mathbf{I}). \tag{246}$$

(ii) From **(i)**, conclude that the quadratic function $\mathbf{H} \mapsto \ell(\mathbf{H})$ is convex, and and hence its every critical point is a local minimum. It follows that its every critical over the convex constraint set $\mathbb{R}^{r\times n}_{\geq 0}$ is a global minimum (but not necessarily unique). (See Ref)

Exercise 4.2.3. Give a psuedocode of a Block Coordinate Descent algorithm for the following matrix factorization problem

$$\inf_{\mathbf{W}\in\mathbb{R}^{d\times r}_{\geq 0},\, \mathbf{H}\in\mathbb{R}^{r\times n}_{\geq 0}} \|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2 + a_0\|\mathbf{H}\|_1 + a_1\|\mathbf{W}\|_1 + a_2\|\mathbf{W}\|_F \tag{247}$$

The algorithm should be similar to the Alternating Least Squares (ALS) in Algorithm 2, but the subproblems of finding $\mathbf{H}_n$ and $\mathbf{W}_n$ should be more detailed by using gradient descent algorithms. (Remark: You may use the gradient given in Exercise 4.2.2. (Hint: The full algorithm is implemented in Jupyter Notebook))

3. Applications of Matrix Factorization

In this section, we apply matrix factorization (MF) to various image and text datasets for the purpose of dictionary learning. See Jupyter Notebook for the experiments given in this section.

3.1. MNIST images. We first apply MF to MNIST dataset to learn 25 dictionary images. There are total four ways to put nonnegativity constriants to $\mathbf{W}$ and $\mathbf{H}$. In the unconstrained case where $\mathbf{W}$ and $\mathbf{H}$ have no constraints, the basis images (see Figure 47 left) themselves can take negative values and they are supposed to span the original MNIST images by using any linear combination. Next, if we give nonnegativity constraint on the code $\mathbf{H}$, then the basis images (see Figure 47 middle) are still allowed to take negative pixel values but they can only additively combined to span original images. In this case, pixel values in dictionary images can still cancel each other when spanning the original image.

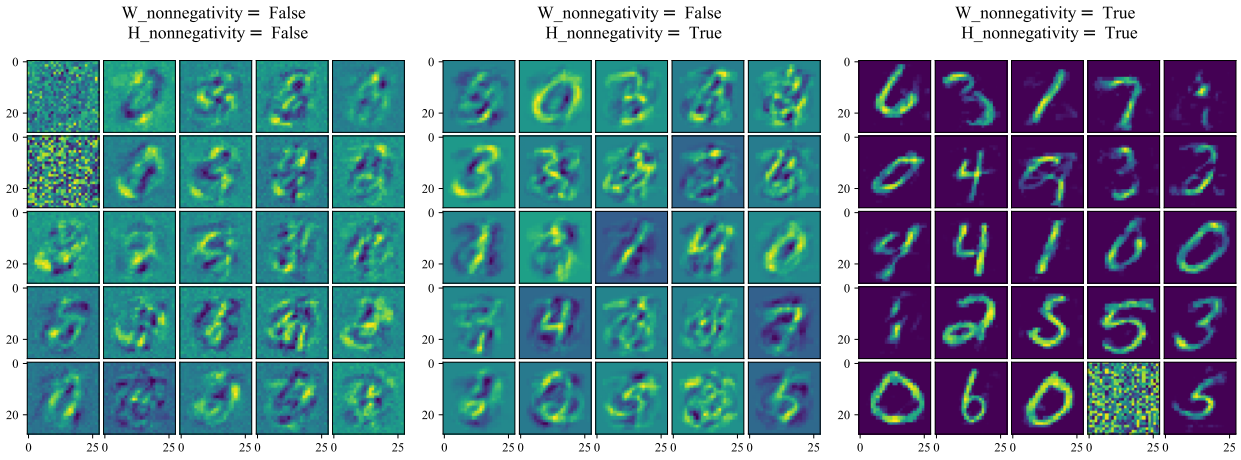


FIGURE 47. Examples of 25 dictionary images learned from MNIST dataset by Matrix Factorization with various nonnegativity constraints.

Lastly, if we constrain both $\mathbf{H}$ and $\mathbf{W}$ to be nonnegative, then there is absolutely no cancelation in any linear approximation of the original images by the basis images. Hence the basis images in this case

(see Figure 47) can only have positive pixel values that really contribute to approximating the original images. In other words, the features we see in the dictionary images are true features of the original images. This is the interpretibility of NMF that made it so popular in the machine learning and signal processing community [LS99].

In Figure 48, we show similar experiments on MNIST with random padding of thickness 20. We see similar patterns in dictionary images depending on the constraints as in Figure 47.

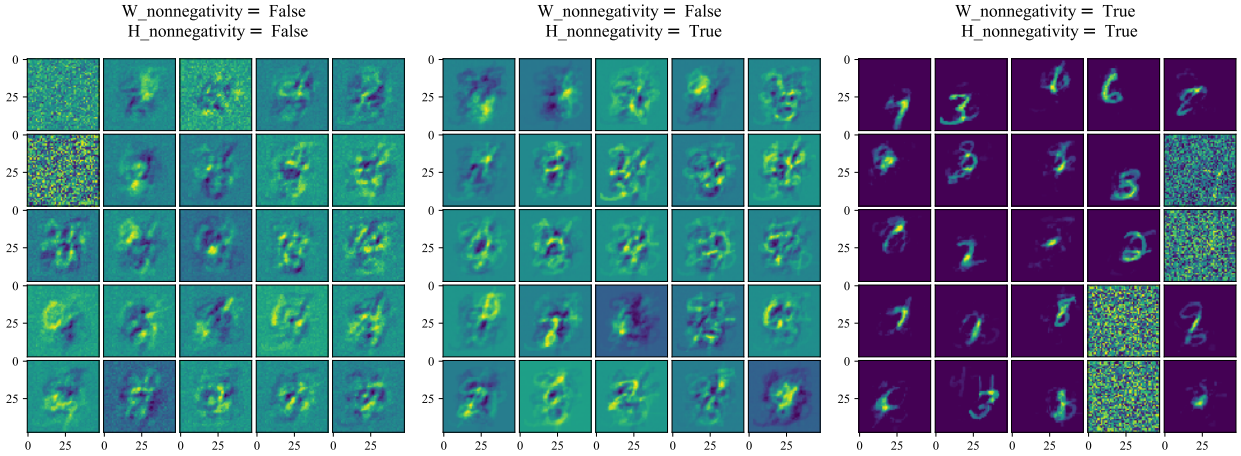


FIGURE 48. Examples of 25 dictionary images learned from MNIST dataset by Matrix Factorization with various nonnegativity constraints. Original 28×28 images are padded with zeros randomly so that they are 48×48 images.

3.2. Olivetti face images. The Olivetti face dataset contains a set of face images taken between April 1992 and April 1994 at AT&T Laboratories Cambridge. There are ten different images of each of 40 distinct subjects. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open / closed eyes, smiling / not smiling) and facial details (glasses / no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement) [SH94]. See Figure 49 for some sample images from this dataset.

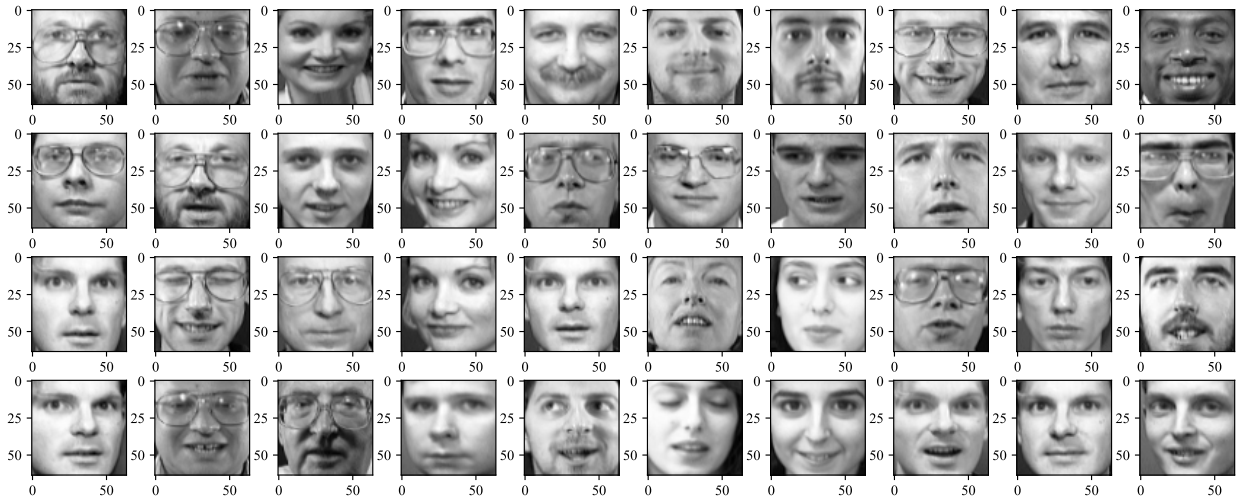


FIGURE 49. Sample images from the Olivetti face dataset

We perform a similar dictionary learning experiment for the Olivetti face dataset in Figure 50. We see similar patterns in dictionary images depending on the constraints as in Figure 47. Note that the "basis faces" learned by NMF (Figure 50 right) are not only the most interpretible, but also have the feature of *localization*. This ability of being able to learn localized features was first reported in [LS99] in 1999 by Lee and Seung. Later in 2004, Hoyer [Hoy04] further invesitgation on promoting such localized dictionary by using a particular sparseness regularizer.

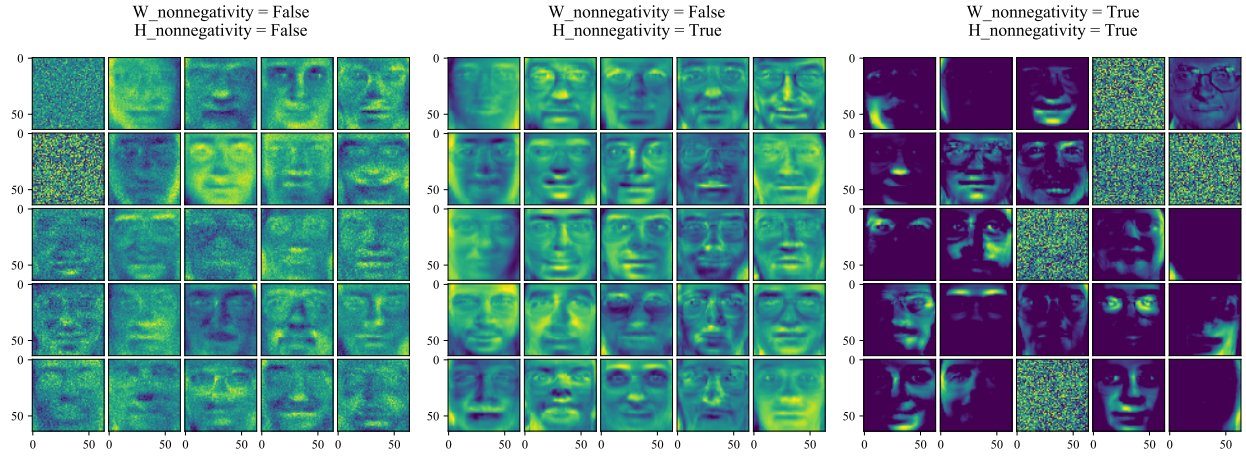


FIGURE 50. Examples of 25 dictionary images learned from the Olivetti face dataset by Matrix Factorization with various nonnegativity constraints.

Below in Figure 51, we discuss the effect of using regularizers on the dictionary matrix $\mathbf{W}$ in NMF. We use basic $L_1 + L_2$ regularizer for $\mathbf{W}$, which means that we are solving the following optimization problem in (247). The ALS algorithm for solving (247) is similar to Algorithm 2. In fact, the update for $\mathbf{H}$ is the same, and for updating $\mathbf{W}$, we need to solve the following coding problem:

$$\mathbf{W}_n \leftarrow \underset{\mathbf{W}\in\mathbb{R}^{d\times r}_{\geq 0}}{\arg\min} \|\mathbf{X} - \mathbf{W}\mathbf{H}_n\|_F^2 + a_1\|\mathbf{W}\|_1 + a_2\|\mathbf{W}\|_F^2. \tag{248}$$

Note that (248) is still a (constrained) convex optimization problem that can be solved iteratively by, e.g., gradient descent[4].

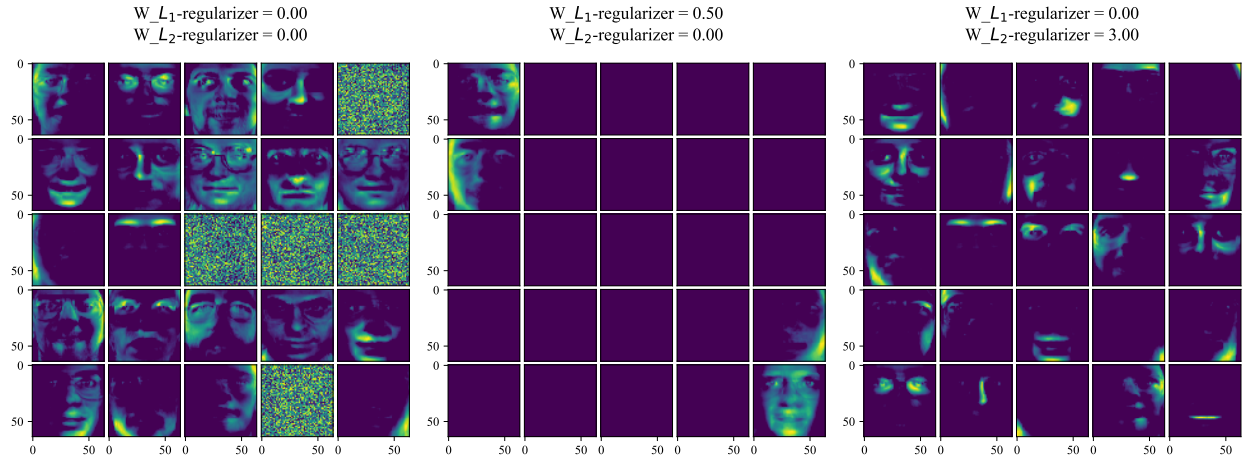


FIGURE 51. Examples of 25 dictionary images learned from the Olivetti face dataset by Matrix Factorization with various regularizers for the dictionary matrix.

[4]It is often the case that one does not have to spend too much iterations to solve sub-problems. In our implementation of ALS for MF in Jupyter Notebook, we used only up to 5 gradient descent itertaions for the sub-problems of coding $\mathbf{H}_n$ and $\mathbf{W}_n$.

In Figure 51, observe that using L_1-regularizer on $\mathbf{W}$ (see Figure 51 middle) makes most of its colums (i.e., vectorizations of basis images) to be zero, and its nonzero columns are not necessarily localized. On the other hand, using L_2-regularizer as in (Figure 51 right), we see the columns of $\mathbf{W}$ are nonzero but they are all localized. One often prefers to learn nonzero localized dictionary elements so that we get more simple and interpretible basis features; On the other hand, one often prefers to have the code matrix $\mathbf{H}$ to have a few nonzero coordinates so that we only need to use a few dictionary elements to approximate original vectors (e.g., for image reconstruction and denoising). The experiment in Figure 51 suggests that for this, we may use L_1 regularizer for the code $\mathbf{H}$ and L_2-regularizer for the dictionary $\mathbf{W}$.

Exercise 4.3.1. Give an explanation on the effect of L_1- and L_2-regularization on the dictionary matrix $\mathbf{W}$ as we see in Figure 51. Drawing contour plots of L_1- and L_2-norm of 2-dimensional vectors would be helpful. (Ref. [Bis06, Sec. 3.1.4]).

Exercise 4.3.2. Use the code in Jupyter Notebook that generates Figures 50 and 51 to determine the *largest* $r \in \mathbb{N}$ *such that every dictionary image properly converges to some non-random image*, where the nonnegativity constraints and regularizers are chosen as in the six cases in Figures 50 and 51 (estimate such r in each of the six cases.)

3.3. Topic modeling for 20Newsgroups. Next, we apply NMF to the 20Newsgroups dataset (see Section 8.1). Recall that each article is represented as a feature vector of word frequency using a fixed vocabulary (sklearn's standard vocab of size 45534) either by using bag-of-words or tf-idf vectorizer. We use 4483 articles from 10 categories (see (165)) combined. The input data matrix $\mathbf{X}$ has size 45534×4483 and we use NMF to factorize this as $\mathbf{WH}$, where $\mathbf{W} \in \mathbb{R}^{45534\times 10}_{\geq 0}$ and $\mathbf{H} \in \mathbb{R}^{10\times 45534}_{\geq 0}$.

In case of factorizing data matrix of vectorized images, the columns of $\mathbf{W}$ were dictionary images that form a basis for the images in the input. For factorizing vectorized texts here, the columns of $\mathbf{W}$ still form a basis for the texts in the input, but they can also be thought of as *topics* in the entire text courpus. Namely, they give groups of words that are used to (approximately) compose each article nonnegatively.

FIGURE 52. 10 topics learned from the 20Newsgroups data by NMF shown as wordclouds. Top 7 words in each topic are shown.

In Figure 52, we show ten topics learnd by NMF as wordclouds. We can indeed see that in each topic, words that belong to the same category appear together (e.g., 'jesus', 'bible', 'god' in topic 6 and 'os', 'pc', 'bit', 'mac' in topic 7). It is important to not, however, that we never used label information and these topics are learned in a purely unsupervised manner. NMF is one of the most widely used technique in topic modeling for text data[5].

[5]Other techniques include Latent Dirichlet Analysis and Nonnegative CP Tensor Decomposition.

We remark that using nonnegativity constraint both on the dictionary **W** and the code **H** are crucial. First, the columns of **W** needs to be vectors of nonnegative entries in order to be interpreted as word frequency vectors. Also, we need to prevent cancelation in the factorization by also making **H** nonnegative so that the words in each topic do share similar context. Indeed, Figure 53 shows the topics learned by a matrix factorization with nonnegative dictionary but unconstrained code. We can see that the words in each topic does not always share the same context when **H** is unconstrained.

FIGURE 53. 10 topics learned from the 20Newsgroups data by a matrix factorization with nonnegative dictionary and unconstrained code.Top 7 words in each topic are shown in wordclouds.

Exercise 4.3.3. The topics learned by NMF in Figures 52 and 53 uses tf-idf vectorizer of the documents. Use bag-of-words vectorizer instead and reproduce these figures. Can you see any difference? Use the code in Use the code in Jupyter Notebook that generates these plots.

4. Principal Component Analysis

In this section, we discuss another matrix-factorization-based classical dimension reduction technique called the 'principal component analysis'. Roughly speaking, the idea is to project n observed d-dimensional data vectors onto a lower r-dimensional space in a 'nice way'. We will discuss two equivalent such formulations of 'variance maximization' and 'error minimization'.

4.1. Maximum variance formulation of PCA. An important concept in the development of PCA is projection of a vector onto a linear subspace generated by a set of vectors. The details are provided in the following exercise.

Exercise 4.4.1 (Projection)**.** Let $\mathbf{x} \in \mathbb{R}^d$ and $C \subseteq \mathbb{R}^d$. The projection $\mathrm{Proj}_C(\mathbf{x})$ of $\mathbf{x}$ onto C is defined by

$$\mathrm{Proj}_C(\mathbf{x}) := \operatorname*{arg\,min}_{\mathbf{u} \in C \subseteq \mathbb{R}^d} \|\mathbf{x}-\mathbf{u}\|_F^2. \tag{249}$$

Namely, $\mathrm{Proj}_C(\mathbf{x})$ is the point in C that is closest to $\mathbf{x}$. It is defined as a constrianed quadratic minimization as above.

(i) Write $\|\mathbf{x}-\mathbf{u}\|_F^2 = \mathrm{tr}((\mathbf{x}-\mathbf{u})^T(\mathbf{x}-\mathbf{u})) = \mathrm{tr}(\mathbf{u}^T\mathbf{u}) - 2\mathrm{tr}(\mathbf{x}^T\mathbf{u}) + \mathrm{tr}(\mathbf{x}^T\mathbf{x})$. Show that

$$\frac{\partial}{\partial \mathbf{u}} \|\mathbf{x}-\mathbf{u}\|_F^2 = 2(\mathbf{u}-\mathbf{x}). \tag{250}$$

(ii) Assume C is the one-dimensional span $\langle \mathbf{v} \rangle := \{a\mathbf{v} \mid a \in \mathbb{R}\}$. Then

$$\text{Proj}_{\mathbf{v}}(\mathbf{x}) := \text{Proj}_{\langle \mathbf{v} \rangle}(\mathbf{x}) = \arg\min_{a \in \mathbb{R}} \|\mathbf{x} - a\mathbf{v}\|_F^2. \tag{251}$$

Use **(i)** and Lagrange multiplier or solve the above unconstrained optimization directly and show

$$\text{Proj}_{\mathbf{v}}(\mathbf{x}) = (\mathbf{x}^T\mathbf{v})\mathbf{v}. \tag{252}$$

(iii) Let $\mathbf{W} = [\mathbf{u}_1, \ldots, \mathbf{u}_r] \in \mathbb{R}^{d \times r}$ be such that its columns are orthonormal, i.e., $\mathbf{W}^T\mathbf{W} = \mathbf{I} \in \mathbb{R}^{r \times r}$. Show that

$$\text{Proj}_{\langle \mathbf{W} \rangle}(\mathbf{x}) := \text{Proj}_{\langle \mathbf{u}_1, \ldots, \mathbf{u}_r \rangle}(\mathbf{x}) = \sum_{i=1}^{r} (\mathbf{u}_i^T\mathbf{x})\mathbf{u}_i = \mathbf{W}(\mathbf{W}^T\mathbf{x}). \tag{253}$$

Next, we introduce the following notions:

Mean and Variance: Fix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{R}^{d \times n}$. We define its *mean* and *variance* as

$$\overline{\mathbf{X}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i \in \mathbb{R}^d, \qquad \text{Var}(\mathbf{X}) = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}_i - \overline{\mathbf{X}}\|_F^2 \in \mathbb{R}. \tag{254}$$

Coriance: The *covariance matrix* $\mathbf{S} = \mathbf{S}(\mathbf{X}) \in \mathbb{R}^{d \times d}$ of $\mathbf{X} \in \mathbb{R}^{d \times n}$ is defined by

$$\mathbf{S} = \frac{1}{n}\sum_{i=1}^{n} (\mathbf{x}_i - \overline{\mathbf{X}})(\mathbf{x}_i - \overline{\mathbf{X}})^T. \tag{255}$$

Projection: Fix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{R}^{d \times n}$ and a dictionary matrix $\mathbf{W} \in \mathbb{R}^{d \times r}$ with orthonormal columns (i.e., $\mathbf{W}^T\mathbf{W} = \mathbf{I} \in \mathbb{R}^{r \times r}$). Define the *projection of* $\mathbf{X}$ *onto the subspace spanned by* $\mathbf{W}$ as

$$\text{Proj}_{\langle \mathbf{W} \rangle}(\mathbf{X}) := \left[\mathbf{W}\mathbf{W}^T\mathbf{x}_1, \ldots, \mathbf{W}\mathbf{W}^T\mathbf{x}_n\right] = \mathbf{W}(\mathbf{W}^T\mathbf{X}) \in \mathbb{R}^{d \times n}. \tag{256}$$

According to Exercise 4.4.1, the columns of $\text{Proj}_{\langle \mathbf{W} \rangle}(\mathbf{X})$ agree with the projection of the columns of $\mathbf{X}$ onto $\langle \mathbf{W} \rangle$.[6]

Projection inherently loses information by collapsing two distinct data points closer to each other. Hence if we were to find the best r-dimensional projection of the data matrix $\mathbf{X}$ that preserves as much information as possible, we should keep the variance of the n vectors after projection as large as possible. Moreover, looking for r-dimensional basis to project onto, we will require the basis vecrors to have unit norm. Hence we would like to seek a matrix $\mathbf{W} \in \mathbb{R}^{d \times r}$ of r basis vectors of unit norm such that the variance after projection is maximized:

(**PCA; maximum variance**)
$$\hat{\mathbf{W}} = \arg\max_{\substack{\mathbf{W} \in \mathbb{R}^{d \times r} \\ \text{rank}(\mathbf{W}) = r}} \text{Var}(\text{Proj}_{\langle \mathbf{W} \rangle}(\mathbf{X})), \tag{257}$$

where $\text{rank}(\mathbf{W})$ denotes the dimension of the linear subspace $\langle \mathbf{W} \rangle$ generated by the columns of $\mathbf{W}$[7].

In the following proposition, we show that finding the top r eigenvectors of the covariance matrix $\mathbf{S}$ solves the PCA problem in (257)[8]. These vectors are also called the *principal components* of $\mathbf{X}$, and the associated eivengalues are called *singular values*.

Proposition 4.4.2. *Let* $\mathbf{X} \in \mathbb{R}^{d \times n}$ *and* $\mathbf{S} = \mathbf{S}(\mathbf{X}) \in \mathbb{R}^{d \times d}$ *denote its covariance matrix. Fix* $r \in \{1, \ldots, n\}$. *Then a solution to the maximum variance PCA problem in* (262) *is given by* $\mathbf{W} \in \mathbb{R}^{d \times r}$ *where*

$$\textit{The columns of } \mathbf{W} \textit{ are the unit eigenvectors of } \mathbf{S} \textit{ with top } r \textit{ largest eigenvalues.} \tag{258}$$

Furthermore, the maximum variance $\text{Var}(\mathbf{W}^T\mathbf{X})$ *after projection acheived equals the sum of the top* r *largest eigenvalues of* $\mathbf{S}$.

[6] Note that $\mathbf{W}^T\mathbf{X} \in \mathbb{R}^{r \times d}$ gives the r-dimensional compressed version of $\mathbf{X}$, whose columns give coordinates of the columns of $\mathbf{X}$ after projection w.r.t. the orthonormal basis of the columns of $\mathbf{W}$.

[7] Note that $\text{rank}(\mathbf{W}) = \dim(\langle \mathbf{W} \rangle)$. So the columns of $\mathbf{W}$ are linearly independent iff $\text{rank}(\mathbf{W}) = r$

[8] But there could be other equivalent solutions.

PROOF. First, write $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_r]$ and we will choose an orthonormal basis $\mathbf{u}_1, \dots, \mathbf{u}_r$ for the linear subspace $\langle \mathbf{w}_1, \dots, \mathbf{w}_r \rangle$ spanned by the columns of $\mathbf{W}$ (e.g., by Gram-Schmidt). Then since $\langle \mathbf{w}_1, \dots, \mathbf{w}_r \rangle = \langle \mathbf{u}_1, \dots, \mathbf{u}_r \rangle$, we have

$$\mathrm{Proj}_{\langle \mathbf{W} \rangle}(\mathbf{X}) = \mathrm{Proj}_{\langle \mathbf{u}_1, \dots, \mathbf{u}_r \rangle}(\mathbf{X}). \tag{259}$$

Hence without loss of generality, we may assume that the columns of $\mathbf{W}$ are orthonormal. i.e., $\mathbf{W}^T\mathbf{W} = \mathbf{I} \in \mathbb{R}^{r \times r}$, in (257).

Since the columns of $\mathbf{W}$ are orthonormal, note that

$$\mathrm{Var}(\mathrm{Proj}_{\langle \mathbf{W} \rangle}(\mathbf{X})) = \mathrm{Var}(\mathbf{W}\mathbf{W}^T\mathbf{X}) = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{W}\mathbf{W}^T(\mathbf{x}_i - \overline{\mathbf{X}})\|_F^2 = \mathrm{tr}\left(\mathbf{W}\mathbf{W}^T\mathbf{S}\mathbf{W}\mathbf{W}^T\right), \tag{260}$$

where $\mathbf{S} = \mathbf{S}(\mathbf{X}) \in \mathbb{R}^{d \times d}$ denotes the covariance matrix of $\mathbf{X}$ defined at (255). By the cyclic property of trace (see Exercise 4.4.3) and since $\mathbf{W}^T\mathbf{W} = \mathbf{I}$,

$$\mathrm{tr}\left(\mathbf{W}\mathbf{W}^T\mathbf{S}\mathbf{W}\mathbf{W}^T\right) = \mathrm{tr}\left(\mathbf{W}^T\mathbf{S}\mathbf{W}(\mathbf{W}^T\mathbf{W})\right) = \mathrm{tr}\left(\mathbf{W}^T\mathbf{S}\mathbf{W}\right). \tag{261}$$

Thus (257) is equivalent to

$$\hat{\mathbf{W}} = \underset{\substack{\mathbf{W} \in \mathbb{R}^{d \times r} \\ \mathbf{W}^T\mathbf{W} = \mathbf{I}}}{\arg\max}\ \mathrm{tr}\left(\mathbf{W}^T\mathbf{S}\mathbf{W}\right). \tag{262}$$

We claim the following: [9]

If $\mathbf{W}$ *is a stationary point of* (262) *then we have an orthornrmal* $\mathbf{U} \in \mathbb{R}^{r \times r}$ *s.t.*

(1) $\mathbf{U}^T\mathbf{W} \in \mathbb{R}^{d \times r}$ *is orthonormal; and*

(2) $\mathrm{tr}(\mathbf{W}^T\mathbf{S}\mathbf{W}) = \mathrm{tr}(\mathbf{U}\mathbf{W}^T\mathbf{S}\mathbf{W}\mathbf{U}^T)$*; and*

(3) $\mathbf{S}(\mathbf{W}\mathbf{U}^T) = (\mathbf{W}\mathbf{U}^T)\mathbf{D}$ *where* $\mathbf{D} \in \mathbb{R}^{r \times r}$ *is diagonal with* r *eigenvalues of* $\mathbf{S}$ *for its diagonal entries.*

In particular, this claim implies that if $\hat{\mathbf{W}} \in \mathbb{R}^{d \times r}$ is a solution to (262), then we can find an 'equivalent maximizer' $\mathbf{U}^T\hat{\mathbf{W}}$ (by (1) and (2)) with the additional property that the columns of $\mathbf{U}^T\hat{\mathbf{W}}$ are eigenvectors of $\mathbf{S}$ (by (3)). Notice that, by writing $\mathbf{D} = \mathrm{diag}(\lambda_1, \dots, \lambda_r)$ where λ_i's are eigenvalues of $\mathbf{S}$,

$$\begin{aligned} \mathrm{tr}\left((\mathbf{W}\mathbf{U}^T)^T\mathbf{S}(\mathbf{W}\mathbf{U}^T)\right) &= \mathrm{tr}\left((\mathbf{W}\mathbf{U}^T)^T(\mathbf{W}\mathbf{U}^T)\mathbf{D}\right) && (263)\\ &= \mathrm{tr}\left(\mathbf{U}\mathbf{W}^T\mathbf{W}\mathbf{U}^T\mathbf{D}\right) && (264)\\ &= \mathrm{tr}(\mathbf{D}) = \lambda_1 + \cdots + \lambda_r. && (265) \end{aligned}$$

Therefore, if we choose the columns of $\mathbf{U}^T\mathbf{W}$ to be an orthonormal set of r eigenvectors of $\mathbf{S}$ with top r eigenvalues of $\mathbf{S}$, then $\mathbf{U}^T\mathbf{W}$ has to be a solution to (262), hence proving the assertion.

It remains to show the claim above. Two essential ingrediemtns are multivariave Lagrange multipliers and spectral theorem for real symmetric matrices.

In order to find all stationary points of (262), we formulate its Lagrangian as

$$L(\mathbf{W}; \boldsymbol{\lambda}) := \mathrm{tr}\left(\mathbf{W}^T\mathbf{S}\mathbf{W}\right) + \sum_{j=1}^{r} \lambda_{jj}(1 - \mathrm{tr}(\mathbf{W}[:,j]^T\mathbf{W}[:,j])) + \sum_{\substack{1 \le i,j \le r \\ i \ne j}} \lambda_{ij}(0 - \mathrm{tr}(\mathbf{W}[:,i]^T\mathbf{W}[:,j])), \tag{266}$$

where $\boldsymbol{\lambda} = (\lambda_{ij}) \in \mathbb{R}^{r \times r}$ is a dual variable[10]. By the method of multivariate Lagrange multiplier, a matrix $\mathbf{W} \in \mathbb{R}^{d \times r}$ is a stationary point of (262) *only if* there exists $\boldsymbol{\lambda} \in \mathbb{R}^{r \times r}$ such that it is a stationary point of $L(\cdot; \boldsymbol{\lambda})$. So, differentiating the objective function above by the jth column $\mathbf{W}[:,j]$ of $\mathbf{W}$,

$$\begin{aligned} \frac{\partial}{\partial \mathbf{W}[:,j]} &= \mathbf{S}\mathbf{W}[:,j] - \sum_{1 \le i \le r} \lambda_{ij}\mathbf{W}[:,i] && (267)\\ &= \mathbf{S}\mathbf{W}[:,j] - \mathbf{W}[:,i]\boldsymbol{\lambda}[:,j]. && (268) \end{aligned}$$

[9] Simply speaking, it says that there exists a global maximizer $\hat{\mathbf{W}}_*$ of (262) such that its columns are eigenvectors of $\mathbf{S}$.

[10] Just think of it as a multi-dimensional Lagrange multiplier

Hence, in order for $\mathbf{W} \in \mathbb{R}^{d\times r}$ to be a stationary point of (262), there should exist a dual variable $\boldsymbol{\lambda} \in \mathbb{R}^{r\times r}$ such that

$$\mathbf{S}\mathbf{W} = \mathbf{W}\boldsymbol{\lambda}. \tag{269}$$

Using $\mathbf{W}^T\mathbf{W} = \mathbf{I} \in \mathbb{R}^{r\times r}$, this implies

$$\mathbf{W}^T\mathbf{S}\mathbf{W} = \boldsymbol{\lambda}. \tag{270}$$

Next, notice that $\mathbf{W}^T\mathbf{S}\mathbf{W} = \boldsymbol{\lambda}$ is a real symmetric matrix. So by a spectral theorem (see Exericse 4.4.4), there exists an orthonormal $\mathbf{U} \in \mathbb{R}^{r\times r}$[11] such that $\mathbf{W}^T\mathbf{S}\mathbf{W} = \mathbf{U}^T\mathbf{D}\mathbf{U}$, where $\mathbf{D}$ is a diagonal matrix whose diagonal entries are the eigenvalues of $\mathbf{W}^T\mathbf{S}\mathbf{W} \in \mathbb{R}^{r\times r}$. This shows

$$\mathbf{S}(\mathbf{W}\mathbf{U}^T) = (\mathbf{W}\mathbf{U}^T)\mathbf{D}, \quad \text{(claim (2))}. \tag{271}$$

Also, it is easy to see that $\mathbf{U}^T\mathbf{W} \in \mathbb{R}^{d\times r}$ is orthonormal, since $(\mathbf{U}^T\mathbf{W})^T(\mathbf{U}^T\mathbf{W}) = \mathbf{W}^T\mathbf{U}\mathbf{U}^T\mathbf{W} = \mathbf{W}^T\mathbf{W} = \mathbf{I} \in \mathbb{R}^{r\times r}$ (claim (1)). Lastly, observe that using $\mathbf{U}^T\mathbf{U} = \mathbf{I}$,

$$\operatorname{tr}(\mathbf{W}^T\mathbf{S}\mathbf{W}) = \operatorname{tr}(\mathbf{W}^T\mathbf{S}\mathbf{W}\mathbf{U}^T\mathbf{U}) = \operatorname{tr}(\mathbf{U}\mathbf{W}^T\mathbf{S}\mathbf{W}\mathbf{U}^T) \qquad \text{(claim (3))}. \tag{272}$$

This shows the claim, thereby finishing the proof of the assertion. □

Exercise 4.4.3 (Cyclic property of trace)**.** Show that $\operatorname{tr}(\mathbf{ABC}) = \operatorname{tr}(\mathbf{BCA})$.

The r principal components of $\mathbf{X}$ as stated in Proposition 4.4.2 are always orthogonal. This is implied by the spectral theorem for real symmetric matrices, see Exercise 4.4.4.

Exercise 4.4.4 (Spectral Theorem for Real Symmetric Matrices)**.** Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be a real symmetric matrix, i.e., $\mathbf{A}^T = \mathbf{A}$. Then the spectral theorem for real symmetric matrices states that the following hold:

1. $\mathbf{A}$ has n real eigenvalues (counting multiplicities);
2. There exists an orthonormal matrix $\mathbf{U} \in \mathbb{R}^{n\times n}$, $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and a diagonal matrix $\mathbf{D} \in \mathbb{R}^{n\times n}$ such that

$$\mathbf{A} = \mathbf{U}^T\mathbf{D}\mathbf{U}; \tag{273}$$

3. In **(ii)**, the orthonomal matrix $\mathbf{U}$ consists of eigenvectors of $\mathbf{A}$ and the diagonal entries of $\mathbf{D}$ are the eigenvalues of $\mathbf{A}$;

In this exercise, we will prove this result by an induction on n.

(i) Show that all of the eigenvalues of $\mathbf{A}$ are real.
(ii) Let $\mathbf{v}_1$ be an eigenvector of $\mathbf{A}$ and let λ_1 be its associated eigenvalue. By **(i)**, λ_1 is real. Extend $\mathbf{v}_1$ to an orthonormal basis $\mathscr{B} := \{\mathbf{v}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n\}$ for $\mathbb{R}^n$ and let $\mathbf{Q} := [\mathbf{v}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n] \in \mathbb{R}^{n\times n}$. Show that

$$\mathbf{Q}^T\mathbf{A}\mathbf{Q} = \left[\begin{array}{c|ccc} \lambda_1 & 0 & \ldots & 0 \\ \hline 0 & & & \\ \vdots & & \mathbf{A}' & \\ 0 & & & \end{array}\right], \tag{274}$$

where $\mathbf{A}' \in \mathbb{R}^{(n-1)\times(n-1)}$ is a real symmetric matrix.
(iii) Suppose there exists an orthonormal matrix $\mathbf{V} \in \mathbb{R}^{(n-1)\times(n-1)}$ and a diagonal matrix $\mathbf{D}' \in \mathbb{R}^{(n-1)\times(n-1)}$ such that $\mathbf{A}' = \mathbf{V}^T\mathbf{D}'\mathbf{V}$. Define

$$\overline{\mathbf{V}} = \left[\begin{array}{c|ccc} 1 & 0 & \ldots & 0 \\ \hline 0 & & & \\ \vdots & & \mathbf{V} & \\ 0 & & & \end{array}\right] \in \mathbb{R}^{n\times n}. \tag{275}$$

[11] $\mathbf{U}^T\mathbf{U} = \mathbf{U}\mathbf{U}^T = \mathbf{I}$ since $\mathbf{U}$ is a square matrix.

Then show that

$$\mathbf{A} = \mathbf{Q}\overline{\mathbf{V}}^T \left[\begin{array}{c|ccc} \lambda_1 & 0 & \dots & 0 \\ \hline 0 & & & \\ \vdots & & \mathbf{D}' & \\ 0 & & & \end{array}\right] \overline{\mathbf{V}}\mathbf{Q}^T. \tag{276}$$

Also show that $\mathbf{U} := \overline{\mathbf{V}}\mathbf{Q}^T \in \mathbb{R}^{n\times n}$ is orthonomal.

(iv) By using **(i)**-**(ii)**, prove that if $\mathbf{A} \in \mathbb{R}^{n\times n}$ is symmetric, show that there are orthogonal and diagonal matrices $\mathbf{U}, \mathbf{D} \in \mathbb{R}^{n\times n}$ such that $\mathbf{A} = \mathbf{U}^T\mathbf{D}\mathbf{U}$ and $\mathbf{D}$ consists of the eigenvalues of $\mathbf{A}$ as its diagonal entries. Also deduce that the columns of $\mathbf{U}$ are the eigenvectors of $\mathbf{A}$.

4.2. Minimum reconstruction error formulation of PCA. In this subsection, we give an alternative formulation of PCA by minimizing reconstruction error after projecting onto an r-dimensional orthonormal basis.

As before, let $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n] \in \mathbb{R}^{d\times n}$ be a data matrix. We will center it by subtracting its mean $\overline{\mathbf{X}}$ so that $\overline{\mathbf{X}} = \mathbf{0}$ and $\mathbf{S} = n^{-1}\mathbf{X}\mathbf{X}^T$. Let $\mathbf{W} \in \mathbb{R}^{d\times r}$ be such that its columns are orthonormal, i.e., $\mathbf{W}^T\mathbf{W} = \mathbf{I} \in \mathbb{R}^{r\times r}$. The 'reconstruction error' of approximating $\mathbf{x}_i$ by the projection $\mathrm{Proj}_{\langle \mathbf{W}\rangle}(\mathbf{x}_i)$ is $\|\mathbf{x}_i - \mathrm{Proj}_{\langle \mathbf{W}\rangle}(\mathbf{x}_i)\|_F^2$. In the minimum reconstruction error formulation of PCA, we would like to choose $\mathbf{W}$ such that the total reconstruction error of all columns of $\mathbf{X}$ is minimized:

(**PCA; min. recons. error**)
$$\hat{\mathbf{W}} = \operatorname*{arg\,min}_{\substack{\mathbf{W}\in\mathbb{R}^{d\times r} \\ \mathrm{rank}(\mathbf{W})=r}} \frac{1}{n}\left\|\mathbf{X} - \mathrm{Proj}_{\langle \mathbf{W}\rangle}(\mathbf{X})\right\|_F^2. \tag{277}$$

In the following proposition, we show the above alternative formulation of PCA is equivalent to the maximum variance one in (257).

Proposition 4.4.5. *The minimum reconstruction error formulation of PCA* (277) *is equivalent to the maximum variance formulation of PCA* (257).

PROOF. As in the proof of Proposition 4.4.2, we may assume that $\mathbf{W} \in \mathbb{R}^{d\times r}$ is orthonormal, i.e., $\mathbf{W}^T\mathbf{W} = \mathbf{I} \in \mathbb{R}^{r\times r}$. We claim that for orthonormal $\mathbf{W}$,

$$\frac{1}{n}\left\|\mathbf{X} - \mathrm{Proj}_{\langle \mathbf{W}\rangle}(\mathbf{X})\right\|_F^2 = \mathrm{Var}(\mathbf{X}) - \mathrm{tr}\left(\mathbf{W}^T\mathbf{S}\mathbf{W}\right), \tag{278}$$

where $\mathbf{S} \in \mathbb{R}^{d\times d}$ denotes the covariance matrix of $\mathbf{X} \in \mathbb{R}^{d\times n}$. Then the assertion follows from (278) since then (277) is equivalent to

$$\operatorname*{arg\,min}_{\substack{\mathbf{W}\in\mathbb{R}^{d\times r} \\ \mathbf{W}^T\mathbf{W}=\mathbf{I}}} \left[\mathrm{Var}(\mathbf{X}) - \mathrm{tr}\left(\mathbf{W}^T\mathbf{S}\mathbf{W}\right)\right] = \operatorname*{arg\,max}_{\substack{\mathbf{W}\in\mathbb{R}^{d\times r} \\ \mathbf{W}^T\mathbf{W}=\mathbf{I}}} \mathrm{tr}\left(\mathbf{W}^T\mathbf{S}\mathbf{W}\right), \tag{279}$$

and the last is an equivalent formulation of the maximum variance PCA (257) as we saw in (262).

It remains to show the identity (278) for orthornomal $\mathbf{W} \in \mathbb{R}^{d\times r}$. To this end, we extend $\mathbf{W}$ to obtain a complete orthonormal basis (ONB) for $\mathbb{R}^d$. Namely, we can choose $\mathbf{W}^\perp \in \mathbb{R}^{d\times(d-r)}$ be such that the columns in their horizontal concatenation $\overline{\mathbf{W}} := [\mathbf{W}, \mathbf{W}^\perp] \in \mathbb{R}^{d\times d}$ form an orthonormal basis (ONB) for $\mathbb{R}^d$ (i.e., $\overline{\mathbf{W}}^T\overline{\mathbf{W}} = \mathbf{I} \in \mathbb{R}^{d\times r}$.) Fix a column $\mathbf{x}_i$ of $\mathbf{X}$. Note that since the columns of $\overline{\mathbf{W}}$ span the full space $\mathbb{R}^d$, we have

$$\mathbf{x}_i = \mathrm{Proj}_{\langle \overline{\mathbf{W}}\rangle}(\mathbf{x}_i) = \overline{\mathbf{W}}\,\overline{\mathbf{W}}^T\mathbf{x}_i. \tag{280}$$

On the other hand, note that $\overline{\mathbf{W}}[\mathbf{W}, \mathbf{O}]^T = [\mathbf{W}, \mathbf{W}^\perp][\mathbf{W}, \mathbf{O}]^T = \mathbf{W}\mathbf{W}^T$, so we have

$$\mathrm{Proj}_{\langle \mathbf{W}\rangle}(\mathbf{x}_i) = \mathbf{W}\mathbf{W}^T\mathbf{x}_i = \overline{\mathbf{W}}[\mathbf{W}, \mathbf{O}]^T\mathbf{x}_i. \tag{281}$$

Thus we get

$$\left\|\mathbf{x}_i - \mathrm{Proj}_{\langle \mathbf{W}\rangle}(\mathbf{x}_i))\right\|_F^2 = \left\|\overline{\mathbf{W}}\,\overline{\mathbf{W}}^T\mathbf{x}_i - \overline{\mathbf{W}}[\mathbf{W}, \mathbf{O}]^T\mathbf{x}_i\right\|_F^2 \tag{282}$$

$$= \left\| \overline{\mathbf{W}} \left([\mathbf{W}, \mathbf{W}^\perp]^T - [\mathbf{W}, \mathbf{O}]^T \right) \mathbf{x}_i \right\|_F^2 \tag{283}$$

$$= \left\| \left([\mathbf{W}, \mathbf{W}^\perp]^T - [\mathbf{W}, \mathbf{O}]^T \right) \mathbf{x}_i \right\|_F^2 \tag{284}$$

$$= \left\| (\mathbf{W}^\perp)^T \mathbf{x}_i \right\|_F^2, \tag{285}$$

where the third equality follows from the fact that $\overline{\mathbf{W}}$ is orthonormal.[12] Also note that $\left\| (\mathbf{W}^\perp)^T \mathbf{x}_i \right\|_F^2 = \left\| \mathbf{x}_i^T (\mathbf{W}^\perp) \right\|_F^2 = (\mathbf{W}^\perp) \mathbf{x}_i \mathbf{x}_i^T (\mathbf{W}^\perp)^T$. Hence

$$\frac{1}{n} \left\| \mathbf{X} - \mathrm{Proj}_{\langle \mathbf{W} \rangle}(\mathbf{X}) \right\|_F^2 = \frac{1}{n} \sum_{i=1}^n (\mathbf{W}^\perp) \mathbf{x}_i \mathbf{x}_i^T (\mathbf{W}^\perp)^T = \mathrm{tr}\left((\mathbf{W}^\perp) \mathbf{S} (\mathbf{W}^\perp)^T \right). \tag{286}$$

Lastly, note that $\mathrm{tr}\left(\overline{\mathbf{W}}^T \mathbf{S} \overline{\mathbf{W}} \right) = \mathrm{tr}\left(\mathbf{S} \overline{\mathbf{W}} \overline{\mathbf{W}}^T \right) = \mathrm{tr}(\mathbf{S}) = \mathrm{Var}(\mathbf{X})$. Hence we get

$$\mathrm{Var}(\mathbf{X}) = \mathrm{tr}\left(\overline{\mathbf{W}}^T \mathbf{S} \overline{\mathbf{W}} \right) = \mathrm{tr}\left([\mathbf{W}, \mathbf{W}^\perp]^T \mathbf{S} [\mathbf{W}, \mathbf{W}^\perp] \right) \tag{287}$$

$$= \mathrm{tr}\left(\mathbf{W}^T \mathbf{S} \mathbf{W} \right) + \mathrm{tr}\left((\mathbf{W}^\perp)^T \mathbf{S} (\mathbf{W}^\perp) \right) + \mathrm{tr}\left(\mathbf{W}^T \mathbf{S} (\mathbf{W}^\perp) \right) + \mathrm{tr}\left((\mathbf{W}^\perp)^T \mathbf{S} \mathbf{W} \right) \tag{288}$$

$$= \mathrm{tr}\left(\mathbf{W}^T \mathbf{S} \mathbf{W} \right) + \mathrm{tr}\left((\mathbf{W}^\perp)^T \mathbf{S} (\mathbf{W}^\perp) \right) + \mathrm{tr}\left(\mathbf{S} \underbrace{(\mathbf{W}^\perp \mathbf{W}^T)}_{\mathbf{O}} \right) + \mathrm{tr}\left(\mathbf{S} \underbrace{\mathbf{W} (\mathbf{W}^\perp)^T}_{\mathbf{O}} \right) \tag{289}$$

$$= \mathrm{tr}\left(\mathbf{W}^T \mathbf{S} \mathbf{W} \right) + \mathrm{tr}\left((\mathbf{W}^\perp)^T \mathbf{S} (\mathbf{W}^\perp) \right). \tag{290}$$

Thus, the claimed identity (278) follows from combining the above and (286). This finishes the proof. □

5. Applications of PCA and Matrix Factorization

In this section, we apply PCA to various image and text datasets for the purpose of dictionary learning and compare with matrix factorization (MF). See Jupyter Notebook for the experiments given in this section. Here we use sklearn's PCA implementation sklearn.decomposition.PCA.

5.1. MNIST. We first apply PCA to the MNIST dataset to learn 24 dictionary images (principal components) and their singular values. These are shown in Figures 54. In Figure 54, it appears that the principal components with larger singular values captures the overall shape of the digits whereas thowe with smaller singular values gives finer details of the shape.

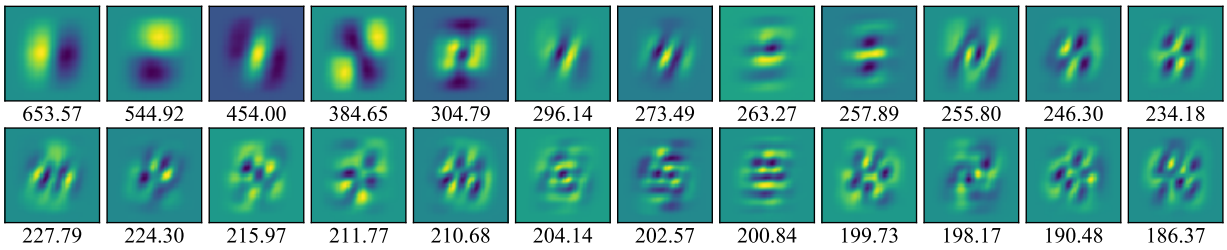


FIGURE 54. Examples of 24 dictionary images (principal components) learned from MNIST dataset by PCA. The numbers below each image are the corresponding singular values.

In Figure 55, we compare the MNIST image dictionary learned by PCA against those learned by MF. The numbers underneath the dictionary images for MF cases represent their 'total usage' computed by row sums of the code matrix $\mathbf{H}$. Namely, observe that from the factorization $\mathbf{X} \approx \mathbf{W}\mathbf{H} \in \mathbb{R}^{d \times n}$, $\mathbf{H}[i, j]$ equals the linear coefficient of $\mathbf{W}[:, i]$ in approximating the ith data point $\mathbf{X}[:, j]$. Hence we can define

$$\text{total usage of } \mathbf{W}[:, i] := \sum_{j=1}^n |\mathbf{H}[i, j]|. \tag{291}$$

[12] Suppose $\mathbf{A} \in \mathbb{R}^{d \times d}$ is orthonormal, i.e., $\mathbf{A}^T \mathbf{A} = \mathbf{I}$. Let $\mathbf{x} \in \mathbb{R}^d$. Then $\|\mathbf{A}\mathbf{x}\|_F^2 = (\mathbf{A}\mathbf{x})^T (\mathbf{A}\mathbf{x}) = \mathbf{x}^T \mathbf{A}^T \mathbf{A} \mathbf{x} = \mathbf{x}^T \mathbf{I} \mathbf{x} = \mathbf{x}^T \mathbf{x} = \|\mathbf{x}\|_F^2$.

Note that we take the absolute value of the code coefficient $\mathbf{H}[i, j]$ above in order to prevent cancelation in computing the total usage. Figure 56 shows similar experiments on MNIST with random padding of thickness 20. It is clear that NMF-learned dictionary are the easiest to interpret.

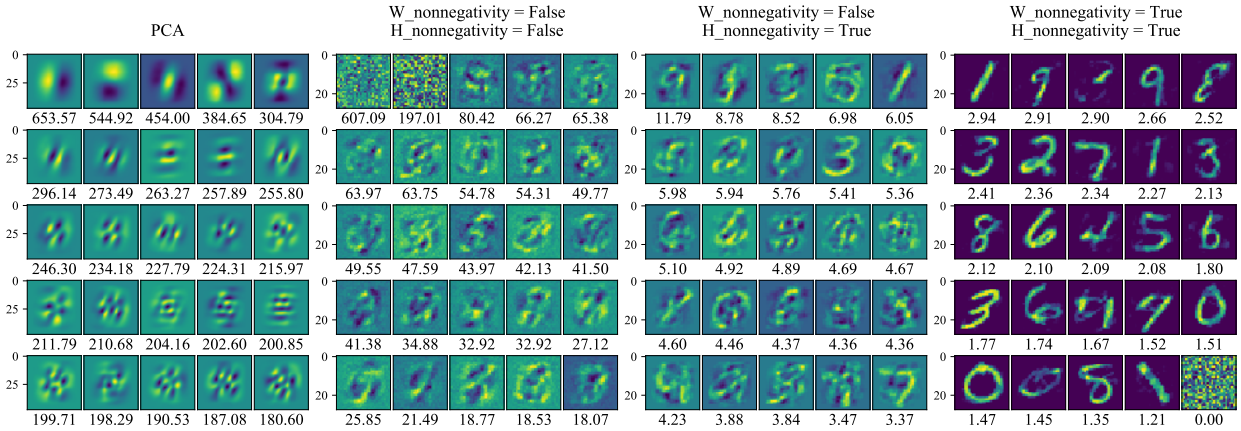


FIGURE 55. Examples of 25 dictionary images learned from MNIST dataset by Matrix Factorization with various nonnegativity constraints.

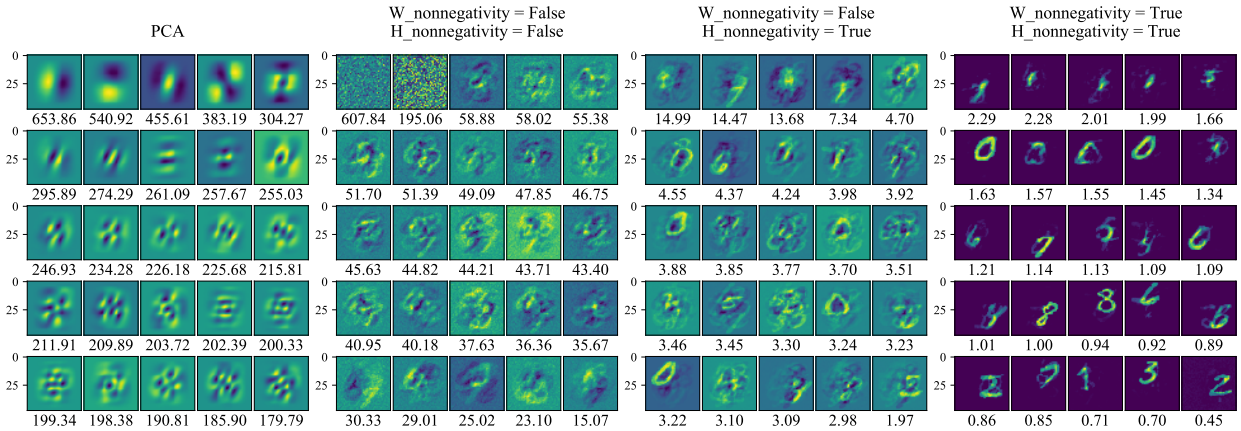


FIGURE 56. Examples of 25 dictionary images learned from MNIST dataset by Matrix Factorization with various nonnegativity constraints. Original 28×28 images are padded with zeros randomly so that they are 48×48 images.

5.2. Olivetti face images. We perform similar dictionary learning experiments for the Olivetti face dataset in Figure 57. Since the principal components may have negative entries, the PCA-learned dictioanry face images (a.k.a. 'eigenfaces') may have some shapes that are not part of any of the actual face images.

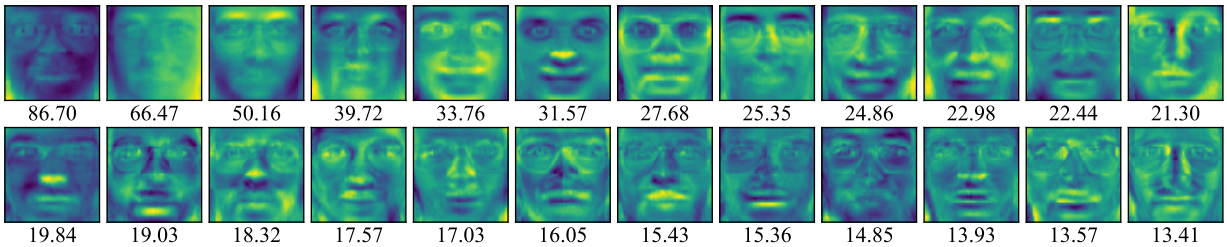


FIGURE 57. Examples of 24 dictionary images (principal components) learned from the Olivetti face dataset by PCA. The numbers below each image are the corresponding singular values.

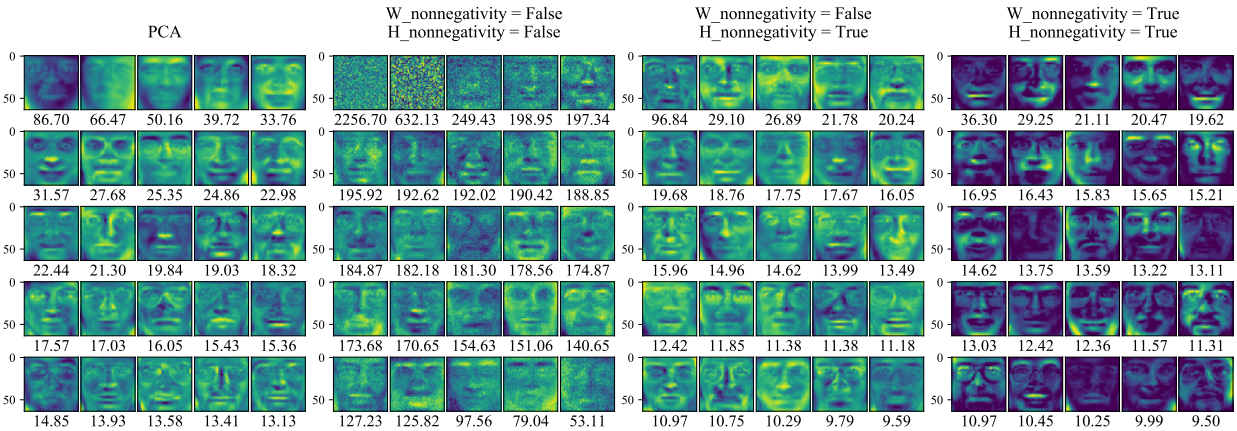


FIGURE 58. Examples of 25 dictionary images learned from the Olivetti face dataset by PCA and Matrix Factorization with various nonnegativity constraints.

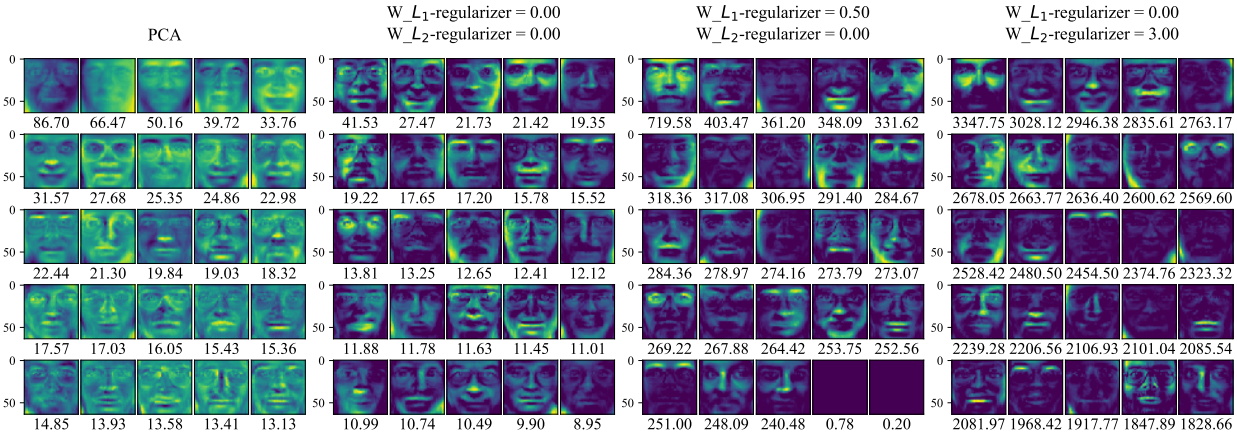


FIGURE 59. Examples of 25 dictionary images learned from the Olivetti face dataset by PCA and Matrix Factorization with various dictionary regularization.

6. Probabilistic PCA and EM algorithm

In this section, we will consider a probabilistic version of PCA and an iterative algorithm for solving the PCA optimization problem called the Expectation-Maximization (EM) algorithm. The discussion in this section is based on [Bis06, Sec. 9.3 and 12.2] and [Row98, TB99].

6.1. The EM algoirthm for PCA. The goal of this section is to derive the following *Expectation-Maximization* (EM) algorithm for PCA. In order to provide the full background and discussion, we need to introduce a probabistic version of PCA and formulate a maximum likelihood formulation of PCA. The EM algorithms form a general class of algorithms used to solve a complex maximum likelihood extimation problem in an efficient and iterative manner. We will discuss more details in the following section, and here we will state the algorithm and give some examples.

One might wonder what would be the point of using an iterative algorithm for PCA while we already have a closed-form solution (see Proposition 4.4.2). One of the main point here is the computational efficiency. Namely, suppose that our data matrix $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n]$ has dimension $d \times n$, so each observation has dimsion d. Say we are interested in learning r principal components, where $r \ll d$ (e.g., $r = 25$ and $d = 10^5$). The closed form solution in Proposition 4.4.2 involves computing the full $d \times d$ covariance

matrix $\mathbf{S} = n^{-1}\sum_{i=1}^{n}(\mathbf{x}_i - \overline{\mathbf{X}})^T(\mathbf{x}_i - \overline{\mathbf{X}})$ and computing *all* of its eigenvector-eigenvalue pairs and keeping only the top r of them. However, computing the full $d \times d$ covariance matrix takes $O(nd^2)$ operations and computing its full eigendecomposition takes $O(d^3)$ operations. But since we are only interested in extracting top $r \ll d$ principal components, it would me much more efficient if we can avoid computing the full covariance matrix and still compute the top r principal components.

In Algorithm 4, we state the EM algorithm for PCA due to [Row98]. The computational cost of this algorithm is $O(rdn)$, which is much less compared to the computational cost $O((n+d)d^2)$ of the full PCA solution.

Algorithm 4 EM algorithm for PCA [Row98, TB99]

Input: $\mathbf{X} \in \mathbb{R}^{d\times n}$ (data matrix); $r \geq 1$ (rank parameter); $\mathbf{W}_0 \in \mathbb{R}^{d\times r}$ (initial estimate); N (number of iterations)
  for $n = 1, \ldots, N$ **do**:
    $\mathbf{H}_n \leftarrow (\mathbf{W}_{n-1}^T\mathbf{W}_{n-1})^{-1}\mathbf{W}_{n-1}^T\mathbf{X} \in \mathbb{R}^{r\times n}$ (E-step)
    $\mathbf{W}_n \leftarrow \mathbf{X}\mathbf{H}_n^T(\mathbf{H}_n\mathbf{H}_n^T)^{-1} \in \mathbb{R}^{d\times r}$ (M-step)
  end for
output: $\mathbf{W}_N$

Here are some important properties of Algorithm 4.

1. (*Convergence to stationary points*) $\mathbf{W}_n$ converges to a stationary point of the PCA optimization problem (257) (and also the equivalent formulation (262)), but not guaranteed to converge to a global optimum.
2. (*Stability of global optimum*) If the initial estimate $\mathbf{W}_0$ is close to a global optimum of the PCA problem $\mathbf{W}^*$, then $\mathbf{W}_n$ converges to $\mathbf{W}^*$.
3. (*Instability of sub-optimal solutions*) If the initial estimate $\mathbf{W}_0$ is close to a stationary point $\mathbf{W}_*$ of the PCA problem and if $\mathbf{W}_*$ is not a global optimum, then $\mathbf{W}_n$ does not converge to $\mathbf{W}_*$.

Item 1 is due to the general convergence property of EM algorithm for maximum likelihood estimation problems. We will discuss more details for this aspect in later subsections. On the other hand, the local stability properties of stationary points of the PCA problem in items 2 and 3 are due to [TB99]. These are desired properties for iterative algorithms for nonconvex optimization problems, but they are not sufficient to guarantee convergence to a globally optimal solution (e.g., principal components for the PCA problem)[13]. For this reason, it is a general wisdom for EM algorithms to chose a proper initialization or try multiple runs for different initializations.

6.2. Olivetti Face images. Before we delve into theoretical details of EM algorithms for PCA, we first discuss some applications of it to the Olivetti face images. In Figure 60, we show the 24 components in $\mathbf{W}_N$ learned by the EM-PCA in Algorithm 4 (100 trials each with random initialization $\mathbf{W}_0$ and 100 iterations).

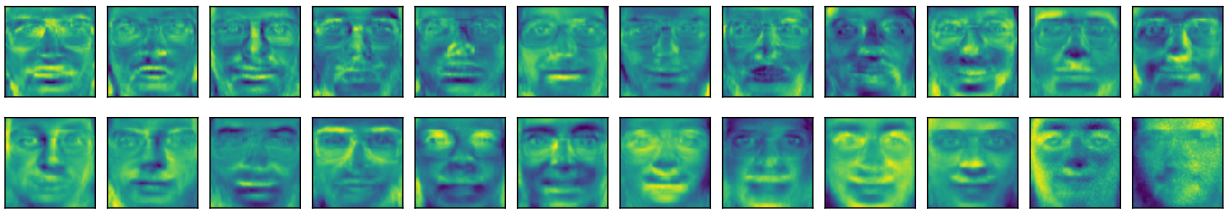


FIGURE 60. Examples of 24 dictionary images (principal components) learned from the Olivetti face dataset by the EM algorithm for PCA (Algorithm 4).

[13] In fact, the existence of a closed-form solution to PCA despite it being a non-convex problem is one of the reasons for the popularity of PCA.

The learned components are similar to the principal components in Figure 57, and if converged properly to a global optimum, these components should span a principal eigenspace of the covariance matrix $\mathbf{S}$ of the input data $\mathbf{X}$[14]. However, in our experiement, it turns out that the components in Figure 57 are almost eigenvectors of $\mathbf{S}$ but not quite. Indeed, in Figure 61, we see that the first column of $\mathbf{W}_N$ learned by the EM-PCA is almost an eigenvector of $\mathbf{S}$ but not as close as the exact principal compoenent. In order to see this, we plot the ratio of the ($64^2 = 4096$) coefficietns in $\mathbf{SW}[:,0]/\mathbf{W}[:,0]$. If $\mathbf{W}[:,0]$ is an eigenvector of $\mathbf{S}$, then these ratios should all equal to some constant λ, which is the eigenvalue of $\mathbf{W}[:,0]$. The standard deviation of these ratios are about 7.0567, while for the exact first principl component, the standard deviation is about 7.7190×10^{-5}.

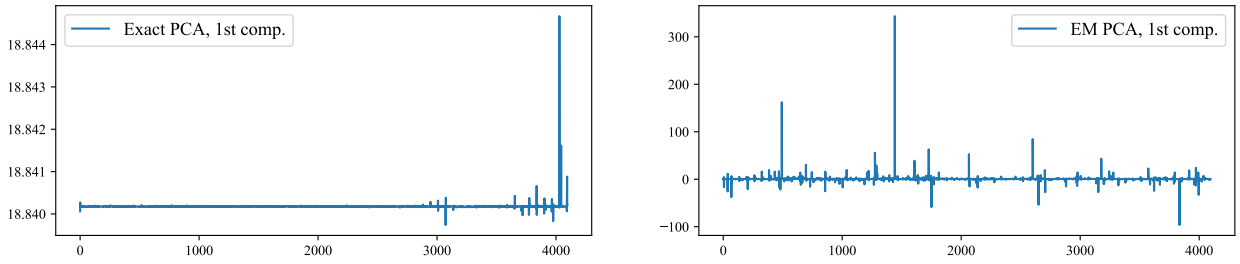


FIGURE 61. Plot of the ratio of coefficients $\mathbf{SW}[:,0]/\mathbf{W}[:,0]$ where $\mathbf{W}$ is either learned by the exact PCA or the EM PCA. $W[:,0]$ is an eigenvector of $\mathbf{S}$ if these ratios are constant.

6.3. The EM algorithm for MLE. The goal of the EM algorithm is to find maximum likelihood solutions for probabilistic models with latent variables. Suppose we have a probabilistic model for the pair $(\mathbf{x},\mathbf{h}) \in \mathbb{R}^p \times \mathbb{R}^r$ of random variables, given by a joint distribution $(x,h) \mapsto p(x,h\,|\,\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ denotes the vector of all parameters to be learned. We will refer to $\mathbf{x}$ as the 'observed variable' and $\mathbf{h}$ as the 'latent variable'. Suppose we are given with iobservations $x_1,\dots,x_n$, which we will denote as a single data matrices $X := [x_1,\dots,x_n] \in \mathbb{R}^{p\times n}$. Each observed data x_i may be associated with latent variable h_i. We will denote the matrix of all latent variables as $H := [h_1,\dots,h_n] \in \mathbb{R}^{r\times n}$. When these are random, we will use the notations $\mathbf{X} := [\mathbf{x}_1,\dots,\mathbf{x}_n] \in \mathbb{R}^{p\times n}$ and $\mathbf{H} := [\mathbf{h}_1,\dots,\mathbf{h}_n] \in \mathbb{R}^{r\times n}$.

The goal is to find a parameter $\boldsymbol{\theta}$ that maximizes the likelihood of observing $\mathbf{X}$ under the model, which can be written as (by using iterated expectation; see Exercise A.3.3)

$$\mathbb{P}_{\boldsymbol{\theta}}(\mathbf{X} = X) = \mathbb{E}_{\mathbf{H}}\left[\mathbb{P}(\mathbf{X} = X\,|\,\mathbf{H},\boldsymbol{\theta})\right] = \sum_H p(X,H\,|\,\boldsymbol{\theta}). \tag{292}$$

As always in MLE problems, we assume that the columns of $\mathbf{X}$ and $\mathbf{H}$ are i.i.d., and in this case they follow the joint distribution $p(x,h\,|\,,\theta)$. We may still denote the joint distribution of $(\mathbf{X},\mathbf{H})$ by p.

Hence we seek to find the maximum likelihood parameter

$$\hat{\boldsymbol{\theta}} := \arg\max_{\boldsymbol{\theta}} \left(\log\left(\mathbb{E}_{\mathbf{H}}\left[\mathbb{P}(\mathbf{X} = X\,|\,\mathbf{H},\boldsymbol{\theta})\right]\right) = \log\left(\sum_H p(X,H\,|\,\boldsymbol{\theta})\right)\right). \tag{293}$$

The key difficulty in solving the above optimization problem is that the expectation $\mathbb{E}_{\mathbf{H}}$ over the latent variable $\mathbf{H}$ (i.e., a sum over H) is inside the log, which does not easily simplify and result in complicated analytical expression.

For the moment, suppose that an 'oracle' told us the unknown (random) latent variable $\mathbf{H}$ is in fact a particular value H. Then we do not have to average over $\mathbf{H}$ and simply need to solve

$$\arg\max_{\boldsymbol{\theta}} \left(\log\left(\mathbb{P}(\mathbf{X} = X\,|\,H,\boldsymbol{\theta})\right) = \log\big(p(X,H\,|\,\boldsymbol{\theta})\big)\right). \tag{294}$$

[14]Still in this case the columns $\mathbf{W}_N$ need not be orthonormal, but one may use Gram-Schmidt to find an ONB for the column span of $\mathbf{W}_N$ and corresponding eigenvalues can be found by multiplying each basis vector to $\mathbf{S}$.

However, in reality $\mathbf{H}$ is random so we may prefer to maximize the expectation $\mathbb{E}_{\mathbf{H}}[\log(\mathbb{P}(\mathbf{X}=X\,|\,\mathbf{H},\boldsymbol{\theta}))]$ instead of solving the above one-point problem. Now note that the distribution of $\mathbf{H}$ conditional on X is given by

$$\mathbf{H} \sim p(H\,|\,X,\boldsymbol{\theta}). \tag{295}$$

Even though X is known at this point, the parameter $\boldsymbol{\theta}$ is not. In order to proceed, we pretend $\boldsymbol{\theta}$ equals our current best estimate $\boldsymbol{\theta}^{\text{old}}$. Then we know the distribution of $\mathbf{H}$ given X and $\boldsymbol{\theta}=\boldsymbol{\theta}^{\text{old}}$, so a better approximation of the original problem (293) than (294) would be the following:

$$\boldsymbol{\theta}^{\text{new}} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{H}}\left[\log(\mathbb{P}(\mathbf{X}=X\,|\,\mathbf{H},\boldsymbol{\theta}))\;\middle|\;X,\boldsymbol{\theta}^{\text{old}}\right] \tag{296}$$

$$= \arg\max_{\boldsymbol{\theta}} \left(\sum_{H} \log(\mathbb{P}(\mathbf{X}=X\,|\,H,\boldsymbol{\theta}))\; p(H\,|\,X,\boldsymbol{\theta}^{\text{old}})\right). \tag{297}$$

We can view (296) as an iterative algorithm that allows us to update the current best parameter $\boldsymbol{\theta}^{\text{old}}$ to a new one $\boldsymbol{\theta}^{\text{new}}$. We call this the *expectation-maximization* (EM) algorithm since we first take the expectation $\mathbb{E}_{\mathbf{H}}$ over the latent variable using the observed data X and the current best parameter $\boldsymbol{\theta}^{\text{old}}$, and then maximize the resulting function in $\boldsymbol{\theta}$ to get the next estimate $\boldsymbol{\theta}^{\text{new}}$.

Algorithm 5 EM algorithm for MLE with latent variable

1: **Input:** $\mathbf{X}\in\mathbb{R}^{d\times n}$ (data matrix); $r\geq 1$ (rank parameter); $\boldsymbol{\theta}_0\in\mathbb{R}^{d\times r}$ (initial estimate); N (number of iterations)
2: **for** $n=1,\ldots,N$ **do**:
3: $Q(\boldsymbol{\theta}\,|\,\boldsymbol{\theta}_{n-1}) \leftarrow \mathbb{E}_{\mathbf{H}}\left[\log(\mathbb{P}(\mathbf{X}=X\,|\,\mathbf{H},\boldsymbol{\theta}))\;\middle|\;X,\boldsymbol{\theta}_{n-1}\right]$ (E-step)
4: $\boldsymbol{\theta}_n \leftarrow \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}\,|\,\boldsymbol{\theta}_{n-1})$ (M-step)
5: **end for**
6: **output:** $\boldsymbol{\theta}_N$

Proposition 4.6.1 (Monotonicity of EM algorithm)**.** *Let $\boldsymbol{\theta}_n$ given by the EM algorithm (Algorithm 5). Then for all $k\geq 0$,*

$$p(X\,|\,\boldsymbol{\theta}_k) \leq p(X\,|\,\boldsymbol{\theta}_{k+1}). \tag{298}$$

In other words, EM algorithm improves (increases) the likelihood $p(X\,|\,\boldsymbol{\theta}_k)$ monotonimcally.

PROOF. We will show the following inequality holds for all $k\geq 1$:

$$\log p(X\,|\,\boldsymbol{\theta}) - p(X\,|\,\boldsymbol{\theta}_k) \geq Q(\boldsymbol{\theta}\,|\,\boldsymbol{\theta}_k) - Q(\boldsymbol{\theta}_k\,|\,\boldsymbol{\theta}_k), \tag{299}$$

where $Q(\boldsymbol{\theta}\,|\,\boldsymbol{\theta}_k) := \mathbb{E}_{\mathbf{H}}\left[\log\big(p(X\,|\,\mathbf{H},\boldsymbol{\theta})\big)\;\middle|\;X,\boldsymbol{\theta}_k\right]$. To see this, first note that $p(X,H\,|\,\boldsymbol{\theta}) = p(X\,|\,H,\boldsymbol{\theta})\cdot p(H\,|\,\boldsymbol{\theta})$. So we can write

$$\log p(X\,|\,\boldsymbol{\theta}) = \log p(X,H\,|\,\boldsymbol{\theta}) - \log p(H\,|\,X,\boldsymbol{\theta}) \tag{300}$$

Since the left hand side does not depend on H,

$$\log p(X\,|\,\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{H}}\left[\log p(X\,|\,\boldsymbol{\theta})\;\middle|\;X,\boldsymbol{\theta}_k\right]. \tag{301}$$

$$= \underbrace{\mathbb{E}_{\mathbf{H}}\left[\log p(X,\mathbf{H}\,|\,\boldsymbol{\theta})\;\middle|\;X,\boldsymbol{\theta}_k\right]}_{Q(\boldsymbol{\theta}\,|\,\boldsymbol{\theta}_k)} + \underbrace{\mathbb{E}_{\mathbf{H}}\left[-\log p(\mathbf{H}\,|\,X,\boldsymbol{\theta})\;\middle|\;X,\boldsymbol{\theta}_k\right]}_{H(\boldsymbol{\theta}\,|\,\boldsymbol{\theta}_k)}. \tag{302}$$

The above equation holds for all $\boldsymbol{\theta}$, including $\boldsymbol{\theta}=\boldsymbol{\theta}_k$. So subtracting the two resulting equations,

$$\log p(X\,|\,\boldsymbol{\theta}) - \log p(X\,|\,\boldsymbol{\theta}_k) = Q(\boldsymbol{\theta}\,|\,\boldsymbol{\theta}_k) - Q(\boldsymbol{\theta}_k\,|\,\boldsymbol{\theta}_k) + [H(\boldsymbol{\theta}\,|\,\boldsymbol{\theta}_k) - H(\boldsymbol{\theta}_k\,|\,\boldsymbol{\theta}_k)]. \tag{303}$$

It then remains to show the last term in the bracket is nonnegative. Indeed,

$$H(\boldsymbol{\theta}\,|\,\boldsymbol{\theta}_k) - H(\boldsymbol{\theta}_k\,|\,\boldsymbol{\theta}_k) = \mathbb{E}_{\mathbf{H}}\left[-\log\frac{p(\mathbf{H}\,|\,X,\boldsymbol{\theta})}{p(\mathbf{H}\,|\,X,\boldsymbol{\theta}_k)}\,\middle|\,X,\boldsymbol{\theta}_k\right] \tag{304}$$

$$= D_{KL}\Big(p(\cdot\,|\,X,\boldsymbol{\theta})\,\Big\|\,p(\cdot\,|\,X,\boldsymbol{\theta}_k)\Big) \geq 0, \tag{305}$$

where the last equality uses the definition of KL divergence and the inequality is due to Gibb's inequality (see Exercise 4.6.3). This shows the assertion. □

Exercise 4.6.2 (log-sum inequality)**.** Let $a_1,\ldots,a_n \geq 0$ and $b_1,\ldots,b_n \geq 0$. Show the following inequality holds:

$$\sum_{i=1}^{n} a_i \log\frac{a_i}{b_i} \geq \left(\sum_{i=1}^{n} a_i\right)\log\frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} b_i}. \tag{306}$$

Furthermore, show that the equality above holds if and only if a_i/b_i does not depend on $i = 1,\ldots,n$.

Exercise 4.6.3 (Gibb's inequality)**.** Let $\mathbf{p},\mathbf{q}$ denote probability distributions on the same probability space. Define the relative entropy (a.k.a. KullbackLeibler divergence) from $\mathbf{q}$ to $\mathbf{p}$ as

$$D_{KL}(\mathbf{p}\,\|\,\mathbf{q}) = \mathbb{E}_{X\sim\mathbf{p}}\left[-\log\big(\mathbf{q}(X)/\mathbf{p}(X)\big)\right]. \tag{307}$$

(i) Show that

$$\mathbb{E}_{X\sim\mathbf{p}}\left[-\log\big(\mathbf{q}(X)/\mathbf{p}(X)\big)\right] \geq -\log\mathbb{E}_{X\sim\mathbf{p}}\left[\big(\mathbf{q}(X)/\mathbf{p}(X)\big)\right] \tag{308}$$

$$= -\log\int q(X)\,dX = -\log 1 = 0. \tag{309}$$

(ii) Deduce Gibb's inequality from **(i)**: $D_{KL}(\mathbf{p}\,\|\,\mathbf{q}) \geq 0$, where the equality holds iff $\mathbf{p} = \mathbf{q}$.
(iii) Deduce the same result as in **(ii)** by using the log-sum inequality (see Exercise 4.6.2).

6.4. Probabilistic PCA. The *probabilistic PCA* [Row98, TB99] is the following *latent variable* model that explains a random observable $\mathbf{x} \in \mathbb{R}^p$ as a linear transform of some lower dimensional latent variable $\mathbf{h} \in \mathbb{R}^r$:

$$\mathbf{x} = \mathbf{W}\mathbf{h} + \boldsymbol{\mu} + \boldsymbol{\varepsilon}, \qquad \mathbf{h} \sim N(\mathbf{0},\mathbf{I}), \qquad \boldsymbol{\varepsilon} \sim N(\mathbf{0},\sigma^2\mathbf{I}), \tag{310}$$

where $\mathbf{h}$ and $\boldsymbol{\varepsilon}$ are independent. More explicitly, $\mathbf{x}$ is modeled as a Gaussian RV:

$$\mathbf{x} \sim N(\boldsymbol{\mu}, \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I}). \tag{311}$$

Indeed, it is easy to see that $\mathbf{x}$ is a Gaussian RV, so we only need to compute its mean and covariance matrix as follow:

$$\mathbb{E}[\mathbf{x}] = \mathbb{E}[\mathbf{W}\mathbf{h} + \boldsymbol{\mu} + \boldsymbol{\varepsilon}] = \mathbb{E}[\mathbf{W}\mathbf{h}] + \boldsymbol{\mu} = \mathbf{W}\mathbb{E}[\mathbf{h}] + \boldsymbol{\mu} = \boldsymbol{\mu}, \tag{312}$$

$$\mathrm{Cov}(\mathbf{x}) = \mathbb{E}\left[(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T\right] \tag{313}$$

$$= \mathbb{E}\left[(\mathbf{W}\mathbf{h}+\boldsymbol{\varepsilon})(\mathbf{W}\mathbf{h}+\boldsymbol{\varepsilon})^T\right] \tag{314}$$

$$= \mathbf{W}\mathbb{E}\left[\mathbf{h}\mathbf{h}^T\right]\mathbf{W}^T + \mathbb{E}[\boldsymbol{\varepsilon}]\mathbb{E}[\mathbf{h}^T]\mathbf{W}^T + \mathbf{W}]\mathbb{E}[\mathbf{h}]\mathbb{E}[\boldsymbol{\varepsilon}^T] = \mathbf{W}\mathbf{W}^T, \tag{315}$$

where for the covariance computation, we have used the fact that $\mathbb{E}[\mathbf{h}\mathbf{h}^T] = \mathrm{Cov}(\mathbf{h}) = \mathbf{I}$, the independence between $\mathbf{h}$ and $\boldsymbol{\varepsilon}$, and $\mathbb{E}[\boldsymbol{\varepsilon}] = \mathbf{0}$.

Denote $\widetilde{X} := X - [\boldsymbol{\mu},\ldots,\boldsymbol{\mu}]$ and $\mathbf{C} := \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I}$. The log likelihood of observing $X = [\vec{x}_1,\ldots,\vec{x}_n]$ from (311) is

$$\ell(X\,|\,\boldsymbol{\mu},\mathbf{W},\sigma) = \sum_{i=1}^{n} \log(2\pi)^{-d/2}|\mathbf{C}|^{-1/2} - \frac{1}{2}(\vec{x}_i-\boldsymbol{\mu})^T\mathbf{C}^{-1}(\vec{x}_i-\boldsymbol{\mu}) \tag{316}$$

$$= -\frac{n}{2}\left(d\log(2\pi) + \log|\mathbf{C}| + \mathrm{tr}\left(\widetilde{X}\mathbf{C}^{-1}\widetilde{X}\right)\right). \tag{317}$$

From this, by differentiating $\boldsymbol{\mu}$ and setting it equal zero, we find the maximum likelihood estimate of $\boldsymbol{\mu}$ is

$$\hat{\boldsymbol{\mu}} = \overline{X} = \frac{1}{n}\sum_{i=1}^{n} \vec{x}_i. \tag{318}$$

On the other hand, the maximum likelihood estimates of $\mathbf{W}$ and σ may be found by using EM algorithm. In order to do this, we first need the conditional distribution of the hidden variable $\mathbf{h}$ given the data $\mathbf{x}$ as in the following exercise.

Exercise 4.6.4. Let $(\mathbf{x}, \mathbf{h})$ denote the pair of observable and latent variables in the probabilistic PCA in (310). Show that the distribution of $\mathbf{h}$ given $\mathbf{x}$ is the following Gaussian distribution:

$$\mathbf{h} \,|\, \mathbf{x} \sim N\left(\mathbf{M}^{-1}\mathbf{W}^T(\mathbf{x}-\boldsymbol{\mu}), \sigma^{-2}\mathbf{M}^{-1}\right), \tag{319}$$

where $\mathbf{M} := \mathbf{W}^T\mathbf{W} + \sigma^2\mathbf{I} \in \mathbb{R}^{r\times r}$. In particular, the expected value of $\mathbf{h}$ given $\mathbf{x}$ is

$$\mathbb{E}[\mathbf{h} \,|\, \mathbf{x}] = \left(\mathbf{W}^T\mathbf{W} + \sigma^2\mathbf{I}\right)^{-1}\mathbf{W}^T(\mathbf{x}-\boldsymbol{\mu}). \tag{320}$$

(Hint: Use Bayes' Theorem and properties of multivariate Gaussian distribution, see [Bis06, Sec. 2.3.3].)

Now we are ready to derive the EM algorithm for probabilistic PCA.

Proposition 4.6.5 (EM algorithm for PPCA)**.** *Fix $X \in \mathbb{R}^{d\times n}$ and denote $\widetilde{X} = X - [\overline{X}, \ldots, \overline{X}]$ where $\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X[:, i]$. The EM algorithm for PPCA (Algorithm 5) equals the following iterative algorithm:*

$$\begin{cases} \mathbb{E}[\mathbf{H}_k] & \leftarrow \left(\mathbf{W}_{k-1}^T\mathbf{W}_{k-1} + \sigma_{k-1}^2\mathbf{I}\right)\mathbf{W}_{k-1}^T\widetilde{X}, \\ \mathbb{E}[\mathbf{H}_k\mathbf{H}_k^T] & \leftarrow \sigma_{k-1}^2\left(\mathbf{W}_{k-1}^T\mathbf{W}_{k-1} + \sigma_{k-1}^2\mathbf{I}\right) + \mathbb{E}[\mathbf{H}_k]\mathbb{E}[\mathbf{H}_k]^T \\ \mathbf{W}_k & \leftarrow \widetilde{X}\mathbb{E}[\mathbf{H}_k^T](\mathbb{E}[\mathbf{H}_k\mathbf{H}_k^T])^{-1}, \\ \sigma_k^2 & \leftarrow n^{-1}\mathrm{tr}\left((\widetilde{X} - \mathbf{W}_k\mathbb{E}[\mathbf{H}_k])^T(\widetilde{X} - \mathbf{W}_k\mathbb{E}[\mathbf{H}_k])\right). \end{cases} \tag{321}$$

PROOF. (Ref: [TB99]) For the E-step, first we compute the conditional likelihood of observing the data $X = [\vec{x}_1, \ldots, \vec{x}_n]$ given the latent variable $H = [\vec{h}_1, \ldots, \vec{h}_n]$. Denote $\boldsymbol{\theta} = [\mathbf{W}, \sigma]$ and suppose we are given $\boldsymbol{\theta}_{k-1} = [\mathbf{W}_{k-1}, \sigma_{k-1}]$. Then

$$\ell(X \,|\, H, \boldsymbol{\theta}) := \sum_{i=1}^{n} \log\left(p(\vec{x}_i \,|\, \vec{h}_i, \boldsymbol{\theta})\right) \tag{322}$$

$$= n\log(\sigma^{-1}(2\pi)^{-d/2}) - \frac{1}{2\sigma^2}\sum_{i=1}^{n} \|\vec{x}_i - \mathbf{W}\vec{h}_i - \boldsymbol{\mu}\|_F^2 \tag{323}$$

$$= n\log(\sigma^{-1}(2\pi)^{-d/2}) - \frac{1}{2\sigma^2}\mathrm{tr}\left((\widetilde{X} - \mathbf{W}H)^T(\widetilde{X} - \mathbf{W}H)\right) \tag{324}$$

$$= n\log(\sigma^{-1}(2\pi)^{-d/2}) - \frac{1}{2\sigma^2}\left[\mathrm{tr}(\widetilde{X}\widetilde{X}^T) - 2\mathrm{tr}(XH^T\mathbf{W}^T) + \mathrm{tr}(\mathbf{W}HH^T\mathbf{W}^T)\right]. \tag{325}$$

The E-step in Algorithm 5 amounts computing the following expectation:

$$Q(\boldsymbol{\theta} \,|\, \boldsymbol{\theta}_{k-1}) = \mathbb{E}_{\mathbf{H}}\left[\ell(X \,|\, \mathbf{H}, \boldsymbol{\theta}) \,\middle|\, X, \boldsymbol{\theta}_{k-1}\right] \tag{326}$$

$$= n\log(\sigma^{-1}(2\pi)^{-d/2}) - \frac{1}{2\sigma^2}\left[\mathrm{tr}(\widetilde{X}\widetilde{X}^T) - 2\mathrm{tr}(\widetilde{X}\mathbb{E}[\mathbf{H}]^T\mathbf{W}^T) + \mathrm{tr}(\mathbf{W}\mathbb{E}[\mathbf{H}\mathbf{H}^T]\mathbf{W}^T)\right], \tag{327}$$

where, using Exercise (4.6.4),

$$\mathbb{E}[\mathbf{H}] := \mathbb{E}_{\mathbf{H}}\left[\mathbf{H} \,\middle|\, X, \boldsymbol{\theta}_{k-1}\right] = \mathbf{M}^{-1}\mathbf{W}_n^T\widetilde{X} = \left(\mathbf{W}_{k-1}^T\mathbf{W}_{k-1} + \sigma_{k-1}^2\mathbf{I}\right)\mathbf{W}_{k-1}^T\widetilde{X} \tag{328}$$

$$\mathbb{E}[\mathbf{H}\mathbf{H}^T] := \mathbb{E}_{\mathbf{H}}\left[\mathbf{H}\mathbf{H}^T \,\middle|\, X, \boldsymbol{\theta}_{k-1}\right] = \mathrm{Cov}(\mathbf{H}) + \mathbb{E}[\mathbf{H}]\mathbb{E}[\mathbf{H}]^T = \sigma_{k-1}^2\left(\mathbf{W}_{k-1}^T\mathbf{W}_{k-1} + \sigma_{k-1}^2\mathbf{I}\right) + \mathbb{E}[\mathbf{H}]\mathbb{E}[\mathbf{H}]^T. \tag{329}$$

Now differentiating $Q(\boldsymbol{\theta}\,|\,\boldsymbol{\theta}_{k-1})$ above with respect to $\mathbf{W}$ and σ gives

$$\nabla_{\mathbf{W}} Q(\boldsymbol{\theta}\,|\,\boldsymbol{\theta}_{k-1}) = -\sigma^{-2}\left[\mathbf{W}\mathbb{E}[\mathbf{H}\mathbf{H}^T] - \widetilde{X}\mathbb{E}[\mathbf{H}]^T\right], \tag{330}$$

$$\frac{\partial}{\partial\sigma} Q(\boldsymbol{\theta}\,|\,\boldsymbol{\theta}_{k-1}) = -\frac{n}{\sigma} + \frac{1}{\sigma^3}\mathrm{tr}\left((\widetilde{X} - \mathbf{W}\mathbb{E}[\mathbf{H}])^T(\widetilde{X} - \mathbf{W}\mathbb{E}[\mathbf{H}])\right). \tag{331}$$

Setting these equal to zero, we get

$$\mathbf{W}_n = \widetilde{X}\mathbb{E}[\mathbf{H}]^T(\mathbb{E}[\mathbf{H}\mathbf{H}^T])^{-1}, \tag{332}$$

$$\sigma_n^2 = \frac{1}{n}\mathrm{tr}\left((\widetilde{X} - \mathbf{W}\mathbb{E}[\mathbf{H}])^T(\widetilde{X} - \mathbf{W}\mathbb{E}[\mathbf{H}])\right). \tag{333}$$

Denoting $\mathbb{E}[\mathbf{H}_n] = \mathbb{E}[\mathbf{H}]$ and $\mathbb{E}[\mathbf{H}_n\mathbf{H}_n^T] = \mathbb{E}[\mathbf{H}\mathbf{H}^T]$, we obtain the claimed iterative algorithm. □

Remark 4.6.6. Consider the special case of 'zero-noise limit' $\sigma \searrow 0$. Then taking $\sigma_k \searrow 0$ for each $k \geq 1$, the EM algorithm for PPCA in Proposition 4.6.5 reduces to

$$\begin{cases} \mathbf{H}_k & \leftarrow \left(\mathbf{W}_{n-1}^T\mathbf{W}_{n-1}\right)\mathbf{W}_{n-1}^T\widetilde{X}, \\ \mathbf{W}_k & \leftarrow \widetilde{X}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{H}_k^T)^{-1}, \end{cases} \tag{334}$$

which is exactly Algorithm 4 we gave earlier.

Bibliography

[Bis06] Christopher M Bishop, *Pattern recognition and machine learning*, springer, 2006.

[HK70] Arthur E Hoerl and Robert W Kennard, *Ridge regression: Biased estimation for nonorthogonal problems*, Technometrics **12** (1970), no. 1, 55–67.

[Hoy04] Patrik O Hoyer, *Non-negative matrix factorization with sparseness constraints.*, Journal of machine learning research **5** (2004), no. 9.

[LKVP21] Hanbaek Lyu, Yacoub H Kureh, Joshua Vendrow, and Mason A Porter, *Learning low-rank latent mesoscale structures in networks*, arXiv preprint arXiv:2102.06984 (2021).

[LNB20] Hanbaek Lyu, Deanna Needell, and Laura Balzano, *Online matrix factorization for markovian data and applications to network dictionary learning*, Journal of Machine Learning Research **21** (2020), no. 251, 1–49.

[LS99] Daniel D. Lee and H. Sebastian Seung, *Learning the parts of objects by non-negative matrix factorization*, Nature **401** (1999), no. 6755, 788.

[LXT+17] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein, *Visualizing the loss landscape of neural nets*, arXiv preprint arXiv:1712.09913 (2017).

[Lyu20] Hanbaek Lyu, *Convergence of block coordinate descent with diminishing radius for nonconvex optimization*, arXiv preprint arXiv:2012.03503 (2020).

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams, *Learning representations by back-propagating errors*, nature **323** (1986), no. 6088, 533–536.

[Row98] Sam Roweis, *Em algorithms for pca and spca*, Advances in neural information processing systems (1998), 626–632.

[SH94] Ferdinando S Samaria and Andy C Harter, *Parameterisation of a stochastic model for human face identification*, Proceedings of 1994 IEEE workshop on applications of computer vision, IEEE, 1994, pp. 138–142.

[SSY18] Tao Sun, Yuejiao Sun, and Wotao Yin, *On markov chain gradient descent*, arXiv preprint arXiv:1809.04216 (2018).

[TB99] Michael E Tipping and Christopher M Bishop, *Probabilistic principal component analysis*, Journal of the Royal Statistical Society: Series B (Statistical Methodology) **61** (1999), no. 3, 611–622.

[Tib96] Robert Tibshirani, *Regression shrinkage and selection via the lasso*, Journal of the Royal Statistical Society: Series B (Methodological) **58** (1996), no. 1, 267–288.

[ZL12] Songfeng Zheng and Weixiang Liu, *Functional gradient ascent for probit regression*, Pattern recognition **45** (2012), no. 12, 4428–4437.

APPENDIX A

Review of basic probability theory

1. Probability measure and probability space

Many things in life are uncertain. Can we 'measure' and compare such uncertainty so that it helps us to make a more informed decision? Probability theory provides a systematic way of doing so.

We begin with idealizing our situation. Let Ω be a finite set, called *sample space*. This is the collection of all possible outcomes that we can observe (think of six sides of a die). We are going to perform some experiment on Ω, and the outcome could be any subset E of Ω, which we call an *event*. Let us denote the collection of all events $E \subseteq \Omega$ by 2^Ω. A *probability measure* on Ω is a function $\mathbb{P} : 2^\Omega \to [0,1]$ such that for each event $E \subseteq \Omega$, it assigns a number $\mathbb{P}(E) \in [0,1]$ and satisfies the following properties:

(i) $\mathbb{P}(\emptyset) = 0$ and $\mathbb{P}(\Omega) = 1$.
(ii) If two events $E_1, E_2 \subseteq \Omega$ are disjoint, then $\mathbb{P}(E_1 \cup E_2) = \mathbb{P}(E_1) + \mathbb{P}(E_1)$.

In words, $\mathbb{P}(E)$ is our quantization of how likely it is that the event E occurs out of our experiment.

Exercise A.1.1. Let $\mathbb{P}$ be a probability measure on sample space Ω. Show the following.

(i) Let $E = \{x_1, x_2, \cdots x_k\} \subseteq \Omega$ be an event. Then $\mathbb{P}(E) = \sum_{i=1}^k \mathbb{P}(\{x_i\})$.
(ii) $\sum_{x\in\Omega} \mathbb{P}(\{x\}) = 1$.

If $\mathbb{P}$ is a probability measure on sample space Ω, we call the pair $(\Omega, \mathbb{P})$ a *probability space*. This is our idealized world where we can precisely measure the uncertainty of all possible events. Of course, there could be many (in fact, infinitely many) different probability measures on the same sample space.

Exercise A.1.2 (coin flip)**.** Let $\Omega = \{H, T\}$ be a sample space. Fix a parameter $p \in [0,1]$, and define a function $\mathbb{P}_p : 2^\Omega \to [0,1]$ by $\mathbb{P}_p(\emptyset) = 0$, $\mathbb{P}_p(\{H\}) = p$, $\mathbb{P}_p(\{T\}) = 1-p$, $\mathbb{P}_p(\{H,T\}) = 1$. Verify that $\mathbb{P}_p$ is a probability measure on Ω for each value of p.

A typical way of constructing a probability measure is to specify how likely it is to see each individual element in Ω. Namely, let $f : \Omega \to [0,1]$ be a function that sums up to 1, i.e., $\sum_{x\in\Omega} f(x) = 1$. Define a function $\mathbb{P} : 2^\Omega \to [0,1]$ by

$$\mathbb{P}(E) = \sum_{\omega\in E} f(x). \tag{335}$$

Then this is a probability measure on Ω, and f is called a *probability distribution* on Ω. For instance, the probability distribution on $\{H,T\}$ we used to define $\mathbb{P}_p$ in Exercise A.1.2 is $f(H) = p$ and $f(T) = 1-p$.

Example A.1.3 (Uniform probability measure)**.** Let $\Omega = \{1, 2, \cdots, m\}$ be a sample space and let $\mathbb{P}$ be the *uniform probability measure* on Ω, that is,

$$\mathbb{P}(\{x\}) = 1/m \qquad \forall x \in \Omega. \tag{336}$$

Then for the event $A = \{1,2,3\}$, we have

$$\mathbb{P}(A) = \mathbb{P}(\{1\} \cup \{2\} \cup \{3\}) \tag{337}$$
$$= \mathbb{P}(\{1\}) + \mathbb{P}(\{2\}) + \mathbb{P}(\{3\}) \tag{338}$$
$$= \frac{1}{m} + \frac{1}{m} + \frac{1}{m} = \frac{3}{m} \tag{339}$$

Likewise, if $A \subseteq \Omega$ is any event and if we let $|A|$ denote the size (number of elements) of A, then

$$\mathbb{P}(A) = \frac{|A|}{m}. \tag{340}$$

For example, let $\Omega = \{1,2,3,4,5,6\}^2$ be the sample space of a roll of two fair dice. Let A be the event that the sum of two dice is 5. Then

$$A = \{(1,4),(2,3),(3,2),(4,1)\}, \tag{341}$$

so $|A| = 4$. Hence $\mathbb{P}(A) = 4/36 = 1/9$. ▲

2. Random variables

Given a finite probability space $(\Omega, \mathbb{P})$, a (discrete) *random variable* (RV) is any real-valued function $X : \Omega \to \mathbb{R}$. We can think of it as the outcome of some experiment on Ω (e.g., height of a randomly selected friend). We often forget the original probability space and specify a RV by is *probability mass function* (PMF) $f_X : \mathbb{R} \to [0,1]$,

$$f_X(x) = \mathbb{P}(X = x) = \mathbb{P}(\{\omega \in \Omega \mid X(\omega) = x\}). \tag{342}$$

Namely, $\mathbb{P}(X = x)$ is the likelihood that the RV X takes value x.

Example A.2.1. Say you win \$1 if a fair coin lands heads and lose \$1 if lands tails. We can set up our probability space $(\Omega, \mathbb{P})$ by $\Omega = \{H, T\}$ and $\mathbb{P}$ = uniform probability measure on Ω. The RV $X : \Omega \to \mathbb{R}$ for this game is $X(H) = 1$ and $X(T) = -1$. The PMF of X is given by $f_X(1) = \mathbb{P}(X = 1) = \mathbb{P}(\{H\}) = 1/2$ and likewise $f_X(-1) = 1/2$.

Exercise A.2.2. Let $(\Omega, \mathbb{P})$ be a probability space and $X : \Omega \to \mathbb{R}$ be a RV. Let f_X be the PMF of X, that is, $f_X(x) = \mathbb{P}(X = x)$ for all x. Show that f_X adds up to 1, that is,

$$\sum_x f_X(x) = 1, \tag{343}$$

where the summation runs over all numerical values x that X can take.

There are two useful statistics of a RV to summarize its two most important properties: It's average and uncertainty. First, if one has to guess the value of a RV X, what would be the best choice? It is the *expectation* (or mean) of X, defined as below:

$$\mathbb{E}[X] = \sum_x x\,\mathbb{P}(X = x). \tag{344}$$

Exercise A.2.3. For any RV X and real numbers $a, b \in \mathbb{R}$, show that

$$\mathbb{E}[aX + b] = a\mathbb{E}[X] + b. \tag{345}$$

On the other hand, say you play two different games where in the first game, you win or lose \$1 depending on a fair coin flip, and in the second game, you win or lose \$10. In both games, your expected winning is 0. But the two games are different in how much the outcome fluctuates around the mean. This notion if the fluctuation is captured by the following quantity called *variance*:

$$\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}(X))^2]. \tag{346}$$

Namely, it is the expected squared difference between X and its expectation $\mathbb{E}(X)$.

Exercise A.2.4. Let X be a RV. Show that $\mathrm{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$.

Here are some of the simplest and yet most important RVs.

Exercise A.2.5 (Bernoulli RV)**.** A RV X is a *Bernoulli* variable with (success) probability $p \in [0,1]$ if it takes value 1 with probability p and 0 with probability $1 - p$. In this case we write $X \sim \mathrm{Bernoulli}(p)$. Show that $\mathbb{E}(X) = p$ and $\mathrm{Var}(X) = p(1-p)$.

Exercise A.2.6 (Indicator variables)**.** Let $(\Omega, \mathbb{P})$ be a probability space and let $E \subseteq \Omega$ be an event. The *indicator variable* of the event E, which is denoted by $\mathbf{1}_E$, is the RV such that $\mathbf{1}_E(\omega) = 1$ if $\omega \in E$ and $\mathbf{1}_E(\omega) = 0$ if $\omega \in E^c$. Show that $\mathbf{1}_E$ is a Bernoulli variable with success probability $p = \mathbb{P}(E)$.

Example A.2.7 (Uniform RV)**.** Let $\Omega = \{x_1, x_2, \cdots x_N\}$ be a finite sample space of real numbers. A RV X is a *uniform* variable on Ω, denoted as $X \sim \text{Uniform}(\Omega)$, if it takes each of the element in Ω with equal probability. That is,

$$\mathbb{P}(X = x_i) = \frac{1}{n} = \frac{1}{|\Omega|} \qquad \text{for all } 1 \le i \le n. \tag{347}$$

Let $X \sim \text{Uniform}(\Omega)$ as above. Then

$$\mathbb{E}[X] = \sum_{i=1}^{n} x_i \cdot \frac{1}{n} = \frac{1}{n} \sum_{i=1}^{n} x_i. \tag{348}$$

Also,

$$\text{Var}(X) = \sum_{i=1}^{n} (x_i - \mathbb{E}[X])^2 \cdot \frac{1}{n} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mathbb{E}[X])^2. \tag{349}$$

The following exercise ties the expectation and the variance of a RV into a problem of finding a point estimator that minimizes the mean squared error.

Exercise A.2.8 (Variance as minimum MSE)**.** Let X be a RV. Let $\hat{x} \in \mathbb{R}$ be a number, which we consider as a 'guess' (or 'estimator' in Statistics) of X. Let $\mathbb{E}[(X - \hat{x})^2]$ be the *mean squared error* (MSE) of this estimation.

(i) Show that

$$\mathbb{E}[(X - \hat{x})^2] = \mathbb{E}[X^2] - 2\hat{x}\mathbb{E}[X] + \hat{x}^2 \tag{350}$$

$$= (\hat{x} - \mathbb{E}[X])^2 + \mathbb{E}[X^2] - \mathbb{E}[X]^2 \tag{351}$$

$$= (\hat{x} - \mathbb{E}[X])^2 + \text{Var}(X). \tag{352}$$

(ii) Conclude that the MSE is minimized when $\hat{x} = \mathbb{E}[X]$ and the global minimum is $\text{Var}(X)$. In this sense, $\mathbb{E}[X]$ is the 'best guess' for X and $\text{Var}(X)$ is the corresponding MSE.

To define a discrete RV, it was enough to specify its PMF. For a continuous RV, *probability distribution function* (PDF) plays an analogous role of PMF. We also need to replace summation $\sum$ with an integral $\int dx$. Namely, X is a *continuous RV* if there is a function $f_X : \mathbb{R} \to [0, \infty)$ such that for any interval $[a, b]$, the probability that X takes a value from an interval $(a, b]$ is given by integrating f_X over the interval $(a, b]$:

$$\mathbb{P}(X \in (a, b]) = \int_a^b f_X(x)\, dx. \tag{353}$$

The *cumulative distribution function* (CDF) of a RV X (either discrete or continuous), denoted by F_X, is defined by

$$F_X(x) = \mathbb{P}(X \le x). \tag{354}$$

By definition of PDF, we get

$$F_X(x) = \int_{-\infty}^{x} f_X(t)\, dt. \tag{355}$$

Conversely, PDFs can be obtained by differentiating corresponding CDFs.

Exercise A.2.9. Let X be a continuous RV with PDF f_X. Let a be a continuity point of f_X, that is, f_X is continuous at a. Show that $F_X(x)$ is differentiable at $x = a$ and

$$\frac{dF_X}{dx}\Big|_{x=a} = f_X(a). \tag{356}$$

The expectation of a continuous RV X with pdf f_X is defined by

$$\mathbb{E}(X) = \int_{-\infty}^{\infty} x f_X(x)\, dx, \tag{357}$$

and its variance $\mathrm{Var}(X)$ is defined by the same formula (346).

3. Conditional and iterated expectations

Let X, Y be discrete RVs. Recall that the expectation $\mathbb{E}(X)$ is the 'best guess' on the value of X when we do not have any prior knowledge on X. But suppose we have observed that some possibly related RV Y takes value y. What should be our best guess on X, leveraging this added information? This is called the *conditional expectation of X given $Y = y$*, which is defined by

$$\mathbb{E}[X|Y = y] = \sum_x x\, \mathbb{P}(X = x|Y = y). \tag{358}$$

This best guess on X given $Y = y$, of course, depends on y. So it is a function in y. Now if we do not know what value Y might take, then we omit y and $\mathbb{E}[X|Y]$ becomes a RV, which is called the *conditional expectation of X given Y*.

Proposition A.3.1 (Iterated expectation)**.** *Let X, Y be RVs. Then $\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X|Y]]$.*

PROOF. We only show the assertion for the case when both X and Y are discrete, and the general case can be argued similarly.

We are going to write the iterated expectation $\mathbb{E}[\mathbb{E}[X|Y]]$ as a double sum and swap the order of summation (Fubini's theorem, as always).

$$\begin{aligned}
\mathbb{E}[\mathbb{E}[X|Y]] &= \sum_y \mathbb{E}[X|Y = y]\mathbb{P}(Y = y) && (359)\\
&= \sum_y \left(\sum_x x\, \mathbb{P}(X = x|Y = y)\right)\mathbb{P}(Y = y) && (360)\\
&= \sum_y \sum_x x\, \mathbb{P}(X = x|Y = y)\mathbb{P}(Y = y) && (361)\\
&= \sum_y \sum_x x\, \mathbb{P}(X = x,\, Y = y) && (362)\\
&= \sum_x \sum_y x\, \mathbb{P}(Y = y|X = x)\mathbb{P}(X = x) && (363)\\
&= \sum_x x \left(\sum_y \mathbb{P}(Y = y|X = x)\right)\mathbb{P}(X = x) && (364)\\
&= \sum_x x\, \mathbb{P}(X = x) = \mathbb{E}(X). && (365)
\end{aligned}$$

□

Remark A.3.2. Here is an intuitive reason why the iterated expectation works. Suppose you want to make the best guess $\mathbb{E}(X)$. Pretending you know Y, you can improve your guess to be $E(X\,|\,Y)$. Then you admit that you didn't know anything about Y and averaged over all values of Y. The result is $\mathbb{E}[\mathbb{E}[X\,|\,Y]]$, and this should be the same best guess on X when we don't know anything about Y.

All our discussions above hold for continuous RVs as well: We simply replace the sum by integral and PMF by PDF. To summarize how we compute the iterated expectations when we condition on discrete and continuous RV:

$$\mathbb{E}[\mathbb{E}[X\,|\,Y]] = \begin{cases} \sum_y \mathbb{E}[X\,|\,Y = y]\mathbb{P}(Y = y) & \text{if } Y \text{ is discrete} \\ \int_{-\infty}^{\infty} \mathbb{E}[X\,|\,Y = y] f_Y(y)\, dy & \text{if } Y \text{ is continuous.} \end{cases} \tag{366}$$

Exercise A.3.3 (Iterated expectation for probability)**.** Let X, Y be RVs.

(i) For any $x \in \mathbb{R}$, show that $\mathbb{P}(X \le x) = \mathbb{E}[\mathbf{1}(X \le x)]$.

(ii) By using iterated expectation, show that

$$\mathbb{P}(X \le x) = \mathbb{E}[\mathbb{P}(X \le x \mid Y)], \tag{367}$$

where the expectation is taken over for all possible values of Y.

Example A.3.4. Let $Y \sim \text{Uniform}([0,1])$ and $X \sim \text{Binomial}(n, Y)$. Then $X|Y = y \sim \text{Binomial}(n, y)$ so $\mathbb{E}[X|Y = y] = ny$. Hence

$$\mathbb{E}[X] = \int_0^1 \mathbb{E}[X \mid Y = y] f_Y(y)\, dy = \int_0^1 ny\, dy = n/2. \tag{368}$$

▲

Example A.3.5. Let $X_1 \sim \text{Exp}(\lambda_1)$ and $X_2 \sim \text{Exp}(\lambda_2)$ be independent exponential RVs. We will show that

$$\mathbb{P}(X_1 < X_2) = \frac{\lambda_1}{\lambda_1 + \lambda_2} \tag{369}$$

using the iterated expectation. Using iterated expectation for probability,

$$\begin{aligned}
\mathbb{P}(X_1 < X_2) &= \int_0^\infty \mathbb{P}(X_1 < X_2 \mid X_1 = x_1) \lambda_1 e^{-\lambda_1 x_1}\, dx_1 && (370)\\
&= \int_0^\infty \mathbb{P}(X_2 > x_1) \lambda_1 e^{-\lambda_1 x_1}\, dx_1 && (371)\\
&= \lambda_1 \int_0^\infty e^{-\lambda_2 x_1} e^{-\lambda_1 x_1}\, dx_1 && (372)\\
&= \lambda_1 \int_0^\infty e^{-(\lambda_1+\lambda_2) x_1}\, dx_1 = \frac{\lambda_1}{\lambda_1 + \lambda_2}. && (373)
\end{aligned}$$

▲

4. Conditional variance

As we have defined conditional expectation, we may define the variance of a RV Y given that another RV X takes a particular value. Recall that the (unconditioned) variance of Y is defined by

$$\text{Var}(Y) = \mathbb{E}[(Y - \mathbb{E}(Y))^2]. \tag{374}$$

Note that there are two places where we take expectation. Given X, we should improve both expectations so the *conditional variance of Y given X is defined by*

$$\text{Var}(Y \mid X) = \mathbb{E}[(Y - \mathbb{E}[Y \mid X])^2 \mid X]. \tag{375}$$

Proposition A.4.1. *Let X and Y be RVs. Then*

$$\text{Var}(Y \mid X) = \mathbb{E}[Y^2 \mid X] - \mathbb{E}[Y \mid X]^2. \tag{376}$$

PROOF. Using linearity of conditional expectation and the fact that $\mathbb{E}[Y \mid X]$ is not random given X,

$$\begin{aligned}
\text{Var}(Y \mid X) &= \mathbb{E}[Y^2 - 2Y\mathbb{E}[Y \mid X] + \mathbb{E}[Y \mid X]^2 \mid Y] && (377)\\
&= \mathbb{E}[Y^2 \mid X] - \mathbb{E}[2Y\mathbb{E}[Y \mid X] \mid X] + \mathbb{E}[\mathbb{E}[Y \mid X]^2 \mid X] && (378)\\
&= \mathbb{E}[Y^2 \mid X] - \mathbb{E}[Y \mid X]\mathbb{E}[2Y \mid X] + \mathbb{E}[Y \mid X]^2\mathbb{E}[1 \mid X] && (379)\\
&= \mathbb{E}[Y^2 \mid X] - 2\mathbb{E}[Y \mid X]^2 + \mathbb{E}[Y \mid X]^2 && (380)\\
&= \mathbb{E}[Y^2 \mid X] - \mathbb{E}[Y \mid X]^2. && (381)
\end{aligned}$$

□

The following exercise explains in what sense the conditional expectation $\mathbb{E}[Y \mid X]$ is the best guess on Y given X, and that the minimum possible mean squared error is exactly the conditional variance $\mathrm{Var}(Y \mid X)$.

Exercise A.4.2. Let X, Y be RVs. For any function $g : \mathbb{R} \to \mathbb{R}$, consider $g(X)$ as an estimator of Y. Let $\mathbb{E}_Y[(Y - g(X))^2 \mid X]$ be the *mean squared error*.

(i) Show that

$$\mathbb{E}_Y[(Y - g(X))^2 \mid X] = \mathbb{E}_Y[Y^2 \mid X] - 2g(X)\mathbb{E}_Y[Y \mid X] + g(X)^2 \tag{382}$$

$$= (g(X) - \mathbb{E}_Y(Y \mid X))^2 + \mathbb{E}_Y[Y^2 \mid X] - \mathbb{E}_Y[Y \mid X]^2 \tag{383}$$

$$= (g(X) - \mathbb{E}_Y(Y \mid X))^2 + \mathrm{Var}(Y \mid X). \tag{384}$$

(ii) Conclude that the mean squared error is minimized when $g(X) = \mathbb{E}_Y[Y \mid X]$ and the global minimum is $\mathrm{Var}(Y \mid X)$.

Next, we study how we can decompose the variance of Y by conditioning on X.

Proposition A.4.3 (Law of total variance)**.** *Let X and Y be RVs. Then*

$$\mathrm{Var}(Y) = \mathbb{E}_X[\mathrm{Var}(Y \mid X)] + \mathrm{Var}(\mathbb{E}[Y \mid X]). \tag{385}$$

PROOF. Using previous result, iterated expectation, and linearity of expectation, we have

$$\begin{aligned}
\mathrm{Var}(Y) &= \mathbb{E}(Y^2) - (\mathbb{E}(Y))^2 && (386)\\
&= \mathbb{E}_X(\mathbb{E}(Y^2|X)) - (\mathbb{E}_X(\mathbb{E}(Y|X)))^2 && (387)\\
&= \mathbb{E}_X(\mathrm{Var}(Y|X) + (\mathbb{E}(Y|X))^2) - (\mathbb{E}_X(\mathbb{E}(Y|X)))^2 && (388)\\
&= \mathbb{E}_X(\mathrm{Var}(Y|X)) + \left[\mathbb{E}_X(\mathbb{E}(Y|X))^2) - (\mathbb{E}_X(\mathbb{E}(Y|X)))^2\right] && (389)\\
&= \mathbb{E}_X(\mathrm{Var}(Y|X)) + \mathrm{Var}(\mathbb{E}(Y|X)). && (390)
\end{aligned}$$

□

Here is a handwavy explanation of why the above is true. Given Y, we should measure the fluctuation of $Y \mid X$ from the conditional expectation $\mathbb{E}[Y \mid X]$, and this is measured as $\mathrm{Var}(Y \mid X)$. Since we don't know Y, we average over all Y, giving $\mathbb{E}[\mathrm{Var}(Y \mid X)]$. But the reference point $\mathbb{E}[Y \mid X]$ itself varies with Y, so we should also measure its own fluctuation by $\mathrm{Var}(\mathbb{E}[Y \mid X])$. These fluctuations add up nicely like Pythagorian theorem because $\mathbb{E}[Y \mid X]$ is an optimal estimator so that these two fluctuations are 'orthogonal'.

Example A.4.4. Let $X \sim \mathrm{Uniform}([0,1])$ and $Y \sim \mathrm{Binomial}(n, X)$. Since $Y \mid X = x \sim \mathrm{Binomial}(n, x)$, we have $\mathbb{E}[Y \mid X = x] = nx$ and $\mathrm{Var}(Y \mid X = x) = nx(1-x)$. Also, since $X \sim \mathrm{Uniform}([0,1])$, we have

$$\mathrm{Var}(\mathbb{E}[Y \mid X]) = \mathrm{Var}(nX) = \frac{n^2}{12}. \tag{391}$$

So by iterated expectation, we get

$$\mathbb{E}[Y] = \mathbb{E}_X[\mathbb{E}[Y \mid X]] = \int_0^1 nx\,dx = \frac{n}{2}. \tag{392}$$

On the other hand, by law of total variance,

$$\mathrm{Var}(Y) = \mathbb{E}[\mathrm{Var}(Y \mid X)] + \mathrm{Var}(\mathbb{E}[Y \mid X]) \tag{393}$$

$$= \int_0^1 nx(1-x)\,dx + \mathrm{Var}(nX) \tag{394}$$

$$= n\left[\frac{x^2}{2} - \frac{x^3}{3}\right]_0^1 + \frac{n^2}{12} = \frac{n^2}{12} + \frac{n}{6}. \tag{395}$$

▲

5. Joint probability distributions

Let X and Y be discrete RVs defined on sample spaces Ω_1 and Ω_2, which we think of the outcome of two random experiments. We can think of the pair (X, Y) of the outcomes as a single observable that describes the two experiments jointly. This object is called a *random vector*, being a vector of random variables. Namely, (X, Y) is a vector-valued function $\Omega_1 \times \Omega_2 \to \mathbb{R}^2$ defined on the joint sample space $\Omega_1 \times \Omega_2$. As for the PDF of a random variable, which is the probability that a RV takes a particular value, we can also ask the probability that the random vector (X, Y) takes a particular vector (x, y). This is called the *joint PMF* of X and Y:

$$f_{X,Y}(x, y) = \mathbb{P}((X, Y) = (x, y)) = \mathbb{P}(X = x,\ Y = y). \tag{396}$$

Given a joint PMF of X and Y, we can determine the PMFs of X and Y simply by 'integrating out' the other variable. Namely, note that for any $x \in \mathbb{R}$,

$$\mathbb{P}(X = x) = \sum_{y \in Y[\Omega_2]} \mathbb{P}(X = x,\ Y = y) = \sum_{y \in Y[\Omega_2]} f_{X,Y}(x, y), \tag{397}$$

where $Y[\Omega_2]$ denotes the set of all possible values of Y. Hence, we sum $f_{X,Y}(x, y)$ over all y to get $f_X(x)$; likewise, we sum $f_{X,Y}(x, y)$ over all x to get $f_Y(y)$. We also call the PMFs of each X and Y the *marginal PMFs* of $f_{X,Y}$.

Proposition A.5.1. *Let X, Y be discrete RVs. Then they are independent if and only if there are functions f and g such that*

$$\mathbb{P}(X = x,\ Y = y) = f(x)g(y) \tag{398}$$

for all possible values (x, y) for (X, Y).

PROOF. If X and Y are independent, then

$$\mathbb{P}(X = x,\ Y = y) = \mathbb{P}(X = x)\mathbb{P}(Y = y) = f_X(x) f_Y(y). \tag{399}$$

Hence the conclusion holds. Conversely, suppose (398) holds for some functions f and g. Then

$$\mathbb{P}(X = x) = \sum_y \mathbb{P}(X = x,\ Y = y) = \sum_y f(x)g(y) = f(x) \sum_y g(y). \tag{400}$$

Since sum of the probabilities $\mathbb{P}(X = x)$ is 1, we have

$$\left(\sum_x f(x)\right)\left(\sum_y g(y)\right) = 1. \tag{401}$$

Hence if we define $\bar{f}(x) = f(x)/\sum_x f(x)$ and $\bar{g}(y) = \sum_y g(y)$, then

$$\mathbb{P}(X = x) = f(x) \sum_y g(y) = \frac{f(x) \sum_y g(y)}{\left(\sum_x f(x)\right)\left(\sum_y g(y)\right)} = \bar{f}(x) \sum_y \bar{g}(y) = \bar{f}(x). \tag{402}$$

Similarly, we have $\mathbb{P}(Y = y) = \bar{g}(y)$. This yields

$$\mathbb{P}(X = x,\ Y = y) = f(x)g(y) = \frac{f(x)g(y)}{\left(\sum_x f(x)\right)\left(\sum_y g(y)\right)} = \bar{f}(x)\bar{g}(y) = \mathbb{P}(X = x)\mathbb{P}(Y = y). \tag{403}$$

Since (x, y) was arbitrary, this shows X and Y are independent. □

Let X and Y be continuous RVs defined on sample spaces Ω_1 and Ω_2. We say X and Y are *jointly continuous* if the random vector (X, Y) has a function $f_{X,Y} : \mathbb{R}^2 \to [0, \infty)$ such that for each subset $\mathscr{A} \subseteq \mathbb{R}^2$,

$$\mathbb{P}((X, Y) \in \mathscr{A}) = \int\int_{(x,y) \in \mathscr{A}} f_{X,Y}(x, y)\, dx\, dy. \tag{404}$$

For instance, if $\mathscr{A}$ is a rectangle of the form $\mathscr{A} = [a, b] \times [c, d]$, then by Fubini's theorem, (404) reads

$$\mathbb{P}(a \le X \le b,\ c \le Y \le d) = \int_a^b \int_c^d f_{X,Y}(x, y)\, dx\, dy. \tag{405}$$

Such a function $f_{X,Y}$ is called the *joint PDF* of X and Y.

As for the joint PMFs, we can determine the PDF of X and Y simply by 'integrating out' the other variable. We also call the PMFs of each X and Y the *marginal PMFs* of $f_{X,Y}$.

Proposition A.5.2. *Let X and Y be continuous RVs with joint PDF $f_{X,Y}$. Then for any $x, y \in \mathbb{R}$, we have*

$$f_X(x) = \int_{-\infty}^{\infty} f_{X,Y}(x,y)\,dy, \qquad f_Y(y) = \int_{-\infty}^{\infty} f_{X,Y}(x,y)\,dx. \tag{406}$$

PROOF. Note that for any interval $[a,b] \subseteq \mathbb{R}$, by definition of joint PDF and Fubini's theorem,

$$\mathbb{P}(a \le X \le b) = \int_a^b \int_{-\infty}^{\infty} f_{X,Y}(x,y)\,dx\,dy \tag{407}$$

$$= \int_a^b \left(\int_{-\infty}^{\infty} f_{X,Y}(x,y)\,dy \right) dx. \tag{408}$$

Then by the definition of PDF of a continuous RV, it follows that the PDF f_X of X is given the the first equation in (406). A Similar argument applies to f_Y as well. □

Example A.5.3. Let X and Y be continuous RVs with joint PDF given by

$$f_{X,Y}(x,y) = \lambda^2 e^{-\lambda(x+y)} \mathbf{1}(x,y \ge 0). \tag{409}$$

Then for each $x \in \mathbb{R}$,

$$f_X(x) = \int_{-\infty}^{\infty} \lambda^2 e^{-\lambda(x+y)} \mathbf{1}(x,y \ge 0)\,dy \tag{410}$$

$$= \lambda e^{-\lambda x} \mathbf{1}(x \ge 0) \int_{-\infty}^{\infty} \lambda e^{-\lambda y} \mathbf{1}(y \ge 0)\,dy \tag{411}$$

$$= \lambda e^{-x} \mathbf{1}(x \ge 0). \tag{412}$$

Note that for the last equality, we have recognized the function $y \mapsto \lambda e^{-\lambda y}\mathbf{1}(y \ge 0)$ as the PDF of $\mathrm{Exp}(\lambda)$ RV. This shows

$$f_X(x) = \lambda e^{-\lambda x} \mathbf{1}(x \ge 0). \tag{413}$$

Hence $X \sim \mathrm{Exp}(\lambda)$. Similarly, we obtain $Y \sim \mathrm{Exp}(\lambda)$. ▲

6. Definition and Examples of MLE

Maximum Likelihood Estimation (MLE) is a classical notion in statistics, which has a prominent use in the context of modern machine learning. A core problem in statistics is to infer an unknown RV X from its sample values $x_1, \cdots, x_n$. In many cases, we can narrow down our inference problem by imposing a probabilistic (statistical) model for X. Namely, say we know X is a Bernoulli RV with unknown success probability p. We then only need to estimate the parameter p using our sample, not the entire distribution of X. MLE gives a systematic approach to this parameter estimation problem. Below we give the pipeline of MLE.

Pipeline of MLE.
Input: An unknown RV X with distribution $f_{X;\theta}$, parameterized by θ in a parameter space Ω. Also have sample values $x_1, x_2, \cdots, x_n$.
Objective: Obtain an estimation $\hat{\theta}$ of θ using the sample.
Method: Choose $\hat{\theta} \in \Omega$ so that it maximizes the following *likelihood function*

$$L(x_1, \cdots, x_n; \theta) = \prod_{i=1}^{n} f_{X,\theta}(x_i). \tag{414}$$

Here the RV $\hat{\theta}(X_1,\cdots,X_n)$ is called the *maximum likelihood estimator* of θ, and its observed value $\hat{\theta}(x_1,\cdots,x_n)$ is called the *maximum likelihood estimate* of θ.

What is the reasoning behind maximizing the likelihood function as above? The underlying assumption is that, *you are seeing the sample values $x_1,\cdots,x_n$ because it was the most likely to see them!* Namely, suppose we have designed the sampling procedure well-enough so that we get i.i.d. samples $X_1,\cdots,X_n$ each with distribution $f_{X;\theta}$. Then we have

$$L(x_1,\cdots,x_n;\theta)=\mathbb{P}(X_1=x_1,\cdots,X_n=x_n;\theta). \tag{415}$$

Namely, the likelihood function on the LHS is the probability of observing a sequence of specific sample values $x_1,\cdots,x_n$ under the i.i.d. assumption and assuming the parameter value θ. Hence, MLE chooses $\hat{\theta}$ to be the value of the parameter in Ω under which the probability of obtaining the current sample is maximized.

Example A.6.1 (MLE for Bernoulli RVs)**.** Suppose $X\sim$ Bernoulli(p) for some unknown $p\in[0,1]$. Suppose we have sample values $x_1,\cdots,x_n$ obtained from an i.i.d. sampling for X. Denote $\bar{x}=n^{-1}\sum_{i=1}^n x_i$. We will show that the MLE for p is $\bar{X}$, that is,

$$\hat{p}=\bar{X}. \tag{416}$$

Let $X_1,\cdots,X_n$ be i.i.d. with distribution Bernoulli(p). Note that

$$\mathbb{P}(X_i=x_i;p)=\begin{cases}p & \text{if } x_i=1\\ 1-p & \text{if } x_i=0\end{cases} \tag{417}$$

$$=p^{x_i}(1-p)^{1-x_i}. \tag{418}$$

Hence the likelihood function is given by

$$L(x_1,\cdots,x_n;p)=\prod_{i=1}^n\mathbb{P}(X_i=x_i;p) \tag{419}$$

$$=\prod_{i=1}^n p^{x_i}(1-p)^{1-x_i} \tag{420}$$

$$=p^{\sum_{i=1}^n x_i}(1-p)^{n-\sum_{i=1}^n x_i}. \tag{421}$$

In order to maximize the likelihood function, it suffices to maximize its logarithm [1]. The *log likelihood function* is given by

$$l(x_1,\cdots,x_n;p):=\log L(x_1,\cdots,x_n;p)=n\bar{x}\log p+(n-n\bar{x})\log(1-p). \tag{422}$$

To maximize the log likelihood function, we take partial derivative in p:

$$\frac{\partial l(x_1,\cdots,x_n;p)}{\partial p}=\frac{n\bar{x}}{p}-\frac{n-n\bar{x}}{1-p}. \tag{423}$$

Setting this equal to zero, we obtain

$$\frac{\bar{x}}{p}=\frac{1-\bar{x}}{1-p}. \tag{424}$$

Rearranging, we obtain $p=\bar{x}$. Hence we have (416) as desired. ▲

Example A.6.2 (MLE for Exponential RVs)**.** Suppose $X\sim\text{Exp}(\lambda)$ for some unknown $\lambda>0$. We have sample values $x_1,\cdots,x_n$ obtained from an i.i.d. sampling for X. Denote $\bar{x}=n^{-1}\sum_{i=1}^n x_i$. We will show that the MLE for λ is $1/\bar{X}$, that is,

$$1/\hat{\lambda}=\bar{X}. \tag{425}$$

[1] since log is a strictly increasing function.

Let $X_1, \cdots, X_n$ be i.i.d. with distribution $\text{Exp}(\lambda)$, which have the following PDF

$$f_X(x;\lambda) = \lambda e^{-\lambda x}\mathbf{1}(x \geq 0). \tag{426}$$

Noting that $x_1, \cdots, x_n \geq 0$, it follows that the likelihood function is given by

$$L(x_1, \cdots, x_n; \lambda) = \prod_{i=1}^{n} f_X(x_i;\lambda) \tag{427}$$

$$= \lambda^n \prod_{i=1}^{n} e^{-\lambda x_i} = \lambda^n e^{-\lambda \sum_{i=1}^n \bar{x}} = \lambda^n e^{-\lambda n \bar{x}}. \tag{428}$$

So the log likelihood function is given by

$$l(x_1, \cdots, x_n; \lambda) = n \log \lambda - \lambda n \bar{x}. \tag{429}$$

To maximize the log likelihood function, we take partial derivative in λ:

$$\frac{\partial l(x_1, \cdots, x_n; \lambda)}{\partial \lambda} = \frac{n}{\lambda} - n\bar{x}. \tag{430}$$

Setting this equal to zero, we obtain $1/\lambda = \bar{x}$. Hence $1/\hat{\lambda} = \bar{X}$, as desired. ▲

Example A.6.3 (MLE for Normal RVs)**.** Suppose $X \sim N(\mu, \sigma^2)$ for some unknown $\mu \in \mathbb{R}$ and $\sigma \geq 0$. Let $X_1, \cdots, X_n$ be i.i.d. with distribution $N(\mu, \sigma^2)$. Denote the sample mean and variance of the empirical distribution by

$$\bar{X} = n^{-1} \sum_{i=1}^{n} X_i, \qquad V = n^{-1} \sum_{i=1}^{n} (X_i - \bar{X})^2. \tag{431}$$

We will show that the MLE for μ and σ^2 are given by sample mean and variance of the empirical distribution:

$$\hat{\mu} = \bar{X}, \qquad \hat{(\sigma^2)} = V. \tag{432}$$

Recall that $N(\mu, \sigma^2)$ has the following PDF

$$f_X(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \tag{433}$$

It follows that the likelihood function is given by

$$L(x_1, \cdots, x_n; \mu, \sigma^2) = \prod_{i=1}^{n} f_X(x_i;\mu,\sigma^2) \tag{434}$$

$$= (2\pi\sigma^2)^{-n/2} \prod_{i=1}^{n} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma^2}\right) \tag{435}$$

$$= (2\pi\sigma^2)^{-n/2} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (x_i-\mu)^2\right). \tag{436}$$

So the log likelihood function is given by

$$l(x_1, \cdots, x_n; \mu, \sigma^2) = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (x_i-\mu)^2. \tag{437}$$

To maximize the log likelihood function, we take partial derivatives in μ and σ^2:

$$\frac{\partial l(x_1, \cdots, x_n; \mu, \sigma^2)}{\partial \mu} = \frac{1}{\sigma^2} \sum_{i=1}^{n} (x_i - \mu) = \frac{n}{\sigma^2} \left(\bar{x} - \mu\right), \tag{438}$$

$$\frac{\partial l(x_1, \cdots, x_n; \mu, \sigma^2)}{\partial \sigma^2} = -\frac{n}{2} \frac{1}{\sigma^2} + \frac{1}{2\sigma^4} \sum_{i=1}^{n} (x_i - \mu)^2. \tag{439}$$

Setting these equations to be zero, we obtain

$$\begin{cases} \bar{x} = \mu \\ -n + \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2 = 0. \end{cases} \tag{440}$$

Using the first equation for the second, we obtain

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2. \tag{441}$$

This shows (432). ▲

7. Bayesian Inference

In this section, we discuss another widely used method of parameter estimation, called the *Bayesian estimation*. Just like in MLE, we would like to estimate an unknown parameter θ in a distribution $f_{X;\theta}$ from analyzing an i.i.d. sample $X_1, \cdots, X_n$ drawn from that distribution. One of the key aspects in Bayesian estimation is to quantify our prior knowledge on the unknown parameter θ as yet another probabilistic model, which we call as *belief*. After observing a new *data* $\{X_1 = x_1, \cdots, X_n = x_n\}$, our current best belief on θ (*prior distribution*) will be updated to a new belief (posterior distribution) according to Bayes' Theorem. Since we are making some additional modeling assumptions on θ, we will be able to draw more quantitative inference on θ using Bayesian estimation.

One of the early bottlenecks in Bayesian estimation was the computational overhead in computing posterior distributions. Due to the recent explosion in our computational capacity, coupled with advanced sampling techniques such as MCMC, Bayesian methods have become one of the indispensable tools in modern statistics and machine learning.

We begin by recalling the following basic observation, from which Bayes' theorem can be easily derived:

$$\mathbb{P}(B|A)\mathbb{P}(A) = \mathbb{P}(A\cap B) = \mathbb{P}(A|B)\mathbb{P}(B). \tag{442}$$

In the case of the events involving continuous random variables taking a certain value, we interpret the above probabilities as probability densities. The following is a random variable version of Bayes' Theorem discussed in the previous subsection.

Theorem A.7.1 (Bayes' Theorem)**.** *Let X and Θ be random variables. Then*

$$\overbrace{\mathbb{P}(\Theta=\theta\,|\,X=x)}^{\text{posterior}} = \frac{\mathbb{P}(X=x\,|\,\Theta=\theta)\mathbb{P}(\Theta=\theta)}{\mathbb{P}(X=x)} \propto \overbrace{\mathbb{P}(X=x\,|\,\Theta=\theta)}^{\text{likelihood}}\overbrace{\mathbb{P}(\Theta=\theta)}^{\text{prior}} \tag{443}$$

PROOF. Follows from (442). □

Remark A.7.2. Depending on whether X and Θ are discrete or continuous, we interpret the probabilities in the above theorem as PDF or PMF, accordingly.

Bayesian inference is usually carried out in the following procedure.

Bayesian inference:

(i) To explain an observable **x**, we choose a *probabilistic model* $p(x\,|\,\theta)$, which is a probability distribution on the possible values x of **x**, depending on a parameter θ.

(ii) Choose a probability distribution $\pi(\theta)$, called the *prior distribution*, that expresses our beliefs about a parameter θ to the best of our current knowledge.

(iii) After observing data $\mathscr{D} = \{x_1, \cdots, x_n\}$, we update our beliefs and compute the *posterior distribution* $p(\theta\,|\,\mathscr{D})$ according to Bayes' Theorem.

When we generate data $\mathscr{D} = \{x_1, \cdots, x_n\}$, we assume that each x_i is obtained by an independent sample X_i of the observable $\mathbf{x}$ from the true distribution of $\mathbf{x}$. According to Bayes' Theorem, we can compute the posterior distribution as

$$\mathbb{P}(\Theta = \theta \,|\, X_1 = x_1, \cdots, X_n = x_n) = \frac{\mathbb{P}(X_1 = x_1, \cdots, X_n = x_n \,|\, \Theta = \theta)\mathbb{P}(\Theta = \theta)}{\mathbb{P}(X_1 = x_1, \cdots, X_n = x_n)}, \tag{444}$$

or in a more compact form,

$$p(\theta \,|\, x_1, \cdots, x_n) = \frac{p(x_1, \cdots, x_n \,|\, \theta)\pi(\theta)}{p(x_1, \cdots, x_n)}. \tag{445}$$

By a slight abuse of notation, we use lowercase p to denote either PMF or PDF, depending on the context.

The conditional probability $p(x_1, \cdots, x_n \,|\, \theta)$ is called the *likelyhood function*, which is the probability of obtaining independent data sample $\mathscr{D} = \{x_1, \cdots, x_n\}$ according to our probability model $p(x \,|\, \theta)$ assuming model parameter $\Theta = \theta$. By the independence between samples, the likelihood function factors into the product of marginal likelihood of each sample:

$$p(x_1, \cdots, x_n \,|\, \theta) = \mathbb{P}(X_1 = x_1, \cdots, X_n = x_n \,|\, \Theta = \theta) \tag{446}$$

$$= \prod_{i=1}^{n} \mathbb{P}(X_i = x_i \,|\, \Theta = \theta) = \prod_{i=1}^{n} p(x_i \,|\, \theta). \tag{447}$$

On the other hand, the joint probability $\mathbb{P}(X_1 = x_1, \cdots, X_n = x_n)$ of obtaining the data sample $\mathscr{D} = \{x_1, \cdots, x_n\}$ from our probability model can be computed by conditioning on the values of model parameter Θ:

$$\mathbb{P}(X_1 = x_1, \cdots, X_n = x_n) = \mathbb{E}\left[\mathbb{P}(X_1 = x_1, \cdots, X_n = x_n \,|\, \Theta) \,\middle|\, \Theta\right] \tag{448}$$

$$= \int_{-\infty}^{\infty} p(x_1, \cdots, x_n \,|\, \theta)\pi(\theta)\, d\theta. \tag{449}$$

What's so important about Bayes' theorem is its interpretation as a means of *inference*, which is one of the fundamental tools in modern machine learning.

Example A.7.3. Suppose Bob has a coin with an unknown probability of heads, which we denote by Θ. This is called the *parameter*. Suppose Alice knows that Bob has one of the three kinds of coins A, B, and C, with the probability of heads being $p_A = 0.2$, $p_B = 0.5$, and $p_C = 0.8$, respectively. This piece of information is called the *model*. Since Alice has no information, she initially assumes that Bob has one of the three coins equally likely. Namely, she assumes the uniform distribution over the sample space $\Omega = \{0.2, 0.5, 0.8\}$. This knowledge is called *prior*.

Now Bob flips his coin 10 times and got 7 heads, and reports this information, which we call `Data`, to Alice. Now that Alice has more information, she needs to update her *prior* to *posterior*, which is the probability distribution on Ω that best explains the `Data`. Namely, it is the conditional probability distribution $\mathbb{P}(\Theta = \theta \,|\, \texttt{Data})$.

First, using our prior, we compute $\mathbb{P}(\texttt{Data})$, the probability of seeing this particular data at hand. This can be done by partitioning:

$$\mathbb{P}(\texttt{Data}) = \mathbb{P}(\texttt{Data} \,|\, \Theta = 0.2)\mathbb{P}(\Theta = 0.2) + \mathbb{P}(\texttt{Data} \,|\, \Theta = 0.5)\mathbb{P}(\Theta = 0.5) \tag{450}$$

$$+ \mathbb{P}(\texttt{Data} \,|\, \Theta = 0.8)\mathbb{P}(\Theta = 0.8) \tag{451}$$

$$= \mathbb{P}(7/10 \text{ heads} \,|\, \Theta = 0.2)\frac{1}{3} + \mathbb{P}(7/10 \text{ heads} \,|\, \Theta = 0.5)\frac{1}{3} + \mathbb{P}(7/10 \text{ heads} \,|\, \Theta = 0.8)\frac{1}{3} \tag{452}$$

$$= \frac{1}{3}\left(\binom{10}{7}(0.2)^7(0.8)^3 + \binom{10}{7}(0.5)^7(0.5)^3 + \binom{10}{7}(0.8)^7(0.2)^3\right) \approx 0.1064, \tag{453}$$

where $\binom{10}{7} = 120$ is the number of ways to choose 7 out of 10 objects.

Second, we reformulate the first equality of Theorem A.7.1 as

$$\mathbb{P}(\Theta \,|\, \texttt{Data}) = \frac{\mathbb{P}(\texttt{Data} \,|\, \Theta)\mathbb{P}(\Theta)}{\mathbb{P}(\texttt{Data})}. \tag{454}$$

Hence we can compute the posterior distribution by

$$\mathbb{P}(\Theta = 0.2 \,|\, \texttt{Data}) = \frac{\mathbb{P}(\texttt{Data} \,|\, \Theta = 0.2)\mathbb{P}(\Theta = 0.2)}{\mathbb{P}(\texttt{Data})} = \frac{\binom{10}{7}(0.2)^7(0.8)^3\frac{1}{3}}{0.1064} \approx 0.0025 \tag{455}$$

$$\mathbb{P}(\Theta = 0.5 \,|\, \texttt{Data}) = \frac{\mathbb{P}(\texttt{Data} \,|\, \Theta = 0.5)\mathbb{P}(\Theta = 0.5)}{\mathbb{P}(\texttt{Data})} = \frac{\binom{10}{7}(0.5)^7(0.5)^3\frac{1}{3}}{0.1064} \approx 0.3670 \tag{456}$$

$$\mathbb{P}(\Theta = 0.8 \,|\, \texttt{Data}) = \frac{\mathbb{P}(\texttt{Data} \,|\, \Theta = 0.8)\mathbb{P}(\Theta = 0.8)}{\mathbb{P}(\texttt{Data})} = \frac{\binom{10}{7}(0.8)^7(0.2)^3\frac{1}{3}}{0.1064} \approx 0.6305. \tag{457}$$

Note that according to the posterior distribution, $\Theta = 0.8$ is the most likely value, which is natural given that we have 7 heads out of 10 flips. However, our knowledge is always incomplete so our posterior knowledge is still a probability distribution on the sample space.

What if Bob flips his coin another 10 times and reports only 3 heads to Alice? Then she will have to use her current prior $\pi = [0.0025, 0.3060, 0.6305]$ (which was obtained as the posterior in the previous round) to compute yet another posterior using the new data. This will be likely to give a higher weight to $\Theta = 0.2$. ▲

Exercise A.7.4. Suppose we have a prior distribution $\pi = [0.0025, 0.3680, 0.6305]$ on the sample space $\Omega = \{0.2, 0.5, 0.8\}$ for the inference problem of unknown parameter Θ. Suppose we are given the data that ten independent flips of probability Θ coin come up heads twice. Compute the posterior distribution using this data and Bayesian inference.

Exercise A.7.5. A test for pancreatic cancer is assumed to be correct 95% of the time: if a person has cancer, the test results in positive with a probability 0.95, and if the person does not have cancer, then the test results in negative with probability 0.95. From recent medical research, it is known that only 5% of the population has pancreatic cancer. Given that the person just tested positive, what is the probability of having cancer?

Example A.7.6 (Bernoulli model and uniform prior)**.** Bob has a coin with unknown probability Θ of heads. Alice has no information whatsoever, so her prior distribution π for Θ is the uniform distribution Uniform($[0,1]$). Bob flips his coin independently n times, and let $X_1, \cdots, X_n$ be the outcome, X_i's are i.i.d. Bernoulli(Θ) variables. Let x_i be the observed value of X_i, and let $s_n = x_1 + \cdots + x_n$ be the number of heads in the n flips. Given data $\mathscr{D} = \{x_1, \cdots, x_n\}$, Alice wants to compute her posterior distribution on Θ.

The the likelyhood function is given by

$$p(x_1, \cdots, x_n \,|\, \theta) = \prod_{i=1}^{n} \theta^{x_i}(1-\theta)^{1-x_i} \tag{458}$$

$$= \theta^{s_n}(1-\theta)^{n-s_n}, \tag{459}$$

where we denote $s_n = x_1 + \cdots + x_n$. Since $\pi \equiv 1$, the posterior distribution is given by

$$p(\theta \,|\, x_1, \cdots, x_n) = \frac{\theta^{s_n}(1-\theta)^{n-s_n}}{p(x_1, \cdots, x_n)}, \tag{460}$$

Note that

$$p(x_1, \cdots, x_n) = \int_0^1 \theta^{s_n}(1-\theta)^{n-s_n}\, d\theta = \frac{1}{\binom{n}{s_n}(n+1)}, \tag{461}$$

where the last equality follows from Exercise A.7.8. Hence

$$p(\theta \,|\, x_1, \cdots, x_n) = \binom{n}{s_n}(n+1)\theta^{s_n}(1-\theta)^{n-s_n} \tag{462}$$

$$= \frac{(n+1)!}{s_n!(n-s_n)!}\theta^{(s_n+1)-1}(1-\theta)^{(n-s_n+1)-1}. \tag{463}$$

Hence, according to Exercise A.7.9, we can write

$$\Theta \,|\, x_1, \cdots, x_n \sim \text{Beta}(s_n+1, n-s_n+1). \tag{464}$$

▲

Exercise A.7.7. Bob has a coin with unknown probability Θ of heads. Alice has the following Beta prior (See Exercise A.7.9 for the definition of Beta distribution):

$$\pi = \text{Beta}(\alpha, \beta). \tag{465}$$

Suppose that Bob gives Alice the data $\mathscr{D}_n = \{x_1, \cdots, x_n\}$, which is the outcome of n independent coin flips. Denote $s_n = x_1 + \cdots + x_n$. Show that Alice's posterior distribution is $\text{Beta}(\alpha + s_n, \beta + n - s_n)$. Namely,

$$\Theta \,|\, \mathscr{D}_n \sim \text{Beta}(\alpha + s_n, \beta + n - s_n). \tag{466}$$

Exercise A.7.8. Let $Y \sim \text{Uniform}([0,1])$ and $X \sim \text{Binomial}(n, Y)$ be independent RVs.

(i) Use iterated expectation for probability to write

$$\mathbb{P}(X = k) = \binom{n}{k} \int_0^1 y^k (1-y)^{n-k}\, dy. \tag{467}$$

(ii) Write $A_{n,k} = \int_0^1 y^k (1-y)^{n-k}\, dy$. Use integration by parts and show that

$$A_{n,k} = \frac{k}{n-k+1} A_{n,k-1}. \tag{468}$$

for all $1 \le k \le n$. Conclude that for all $0 \le k \le n$,

$$A_{n,k} = \frac{1}{\binom{n}{k}} \frac{1}{n+1}. \tag{469}$$

(iii) Conclude that $X \sim \text{Uniform}(\{0, 1, \cdots, n\})$.

Exercise A.7.9 (Beta distribution)**.** A random varible X taking values from $[0,1]$ has Beta distribution of parameters α and β, which we denote by $\text{Beta}(\alpha, \beta)$, if it has PDF

$$f_X(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \tag{470}$$

where $\Gamma(z)$ is the Euler Gamma function defined by

$$\Gamma(z) = \int_0^\infty x^{z-1} e^{-x}\, dx. \tag{471}$$

(i) Use integration by parts to show the following recursion

$$\Gamma(z+1) = z\Gamma(z). \tag{472}$$

Deduce that $\Gamma(n) = (n-1)!$ for all integers $n \ge 1$.

(ii) Let $X \sim \text{Beta}(k+1, n-k+1)$. Use (i) to show that

$$f_X(x) = \frac{n!(n+1)}{k!(n-k)!} x^k (1-x)^{n-k} = \frac{x^k(1-x)^{n-k}}{1/\binom{n}{k}(n+1)}. \tag{473}$$

Use Exercise A.7.8 to verify that the above function is indeed a PDF (i.e., it integrates to 1).

(iii)* Show that if $X \sim \text{Beta}(\alpha, \beta)$, then

$$\mathbb{E}[X] = \frac{\alpha}{\alpha+\beta}, \quad \text{Var}(X) = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}. \tag{474}$$

Example A.7.10 (Bayesian estimator)**.** Recall Exercise A.7.7, where we want to infer an unknown parameter θ in Bernoulli(θ), starting from the beta prior $\pi = \text{Beta}(\alpha, \beta)$. After observing a data $\mathscr{D} = (x_1, \cdots, x_n)$ consisting of $s_n = \sum_{i=1}^n x_i$ successes, our posterior distribution is Beta distribution with parameters $s_n + 1$ and $\beta + n - s_n$. In other words,

$$\Theta | \mathscr{D} \sim \text{Beta}(\alpha + s_n, \beta + n - s_n). \tag{475}$$

Now suppose we want to get a point estimator $\hat{\theta}$ for the unknown parameter from our posterior distribution. Any choice of the estimator will result in some kind of error, so we would like our estimator to be minimizing an error function of choice. A standard choice is the mean squared error (MSE), which in our case would be under conditioning on the observed data. That is, we want our estimator $\hat{\theta}$ to be such that

$$\hat{\theta} = \text{argmin}\, \mathbb{E}[(\Theta - \hat{\theta})^2 \,|\, \mathscr{D}]. \tag{476}$$

According to Exercise A.4.2, the above MSE will be minimized when $\hat{\theta} = \mathbb{E}[\Theta \,|\, \mathscr{D}]$, and the minimum MSE is the conditional variance $\text{Var}(\Theta \,|\, \mathscr{D})$. Noting the mean of beta distribution in Exercise A.7.9, we obtain the following Bayesian estimator

$$\hat{\theta} = \mathbb{E}[\Theta \,|\, \mathscr{D}] = \mathbb{E}[\text{Beta}(\alpha + s_n, \beta + n - s_n)] = \frac{\alpha + s_n}{\alpha + \beta + n}. \tag{477}$$

▲