Stanford CS229: Machine Learning

JAMES ZHANG[*]

June 28, 2023

[*]Email: jzhang72@terpmail.umd.com

Contents

§1 Linear Regression

§1.1 Notation

- $x^{(i)}$ denotes input variables of features, and $y^{(i)}$ denotes the output or target variable
- $x^{(i)} \in \mathbb{R}^{n+1}$ because we have a fake feature $x_0 = 1$
- $y^{(i)} \in \mathbb{R}$ always in the case of regression
- $x_j^{(i)}$ represents the j-th feature of the i-th training example
- m denotes the number of training examples
- n denotes the number of features
- $h(x)$ denotes the hypothesis function, which is a linear function of the features x and $x_0 = 1$
- $J(\vec{w})$ denotes the cost function you're trying to minimize by finding the optimal set of parameters $\vec{w}$ to straight line fit the data
- X denotes the space of input values, and Y denotes the space of output values

§1.2 Hypothesis Function

Definition 1.1. We represent our **hypothesis function** as a linear function of x, such that our values w_i are weights

$$h(x) = \sum_{i=0}^{n} w_i x_i = w \cdot x = w^T x$$

such that w and x are both vectors, and n is the number of input variables.

§1.3 Cost Function

Definition 1.2. Now let us define the **least squares cost function** that gives rise to the ordinary least squares regression model.

$$J(\vec{w}) = \frac{1}{2m} \sum_{i=1}^{m} (h(x^{(i)} - y^{(i)})^2$$

and we want to choose $\vec{w}$ so as to minimize the cost function, and this is known as the **gradient descent algorithm**.

$$w_j := w_j - \alpha \frac{\partial}{\partial w_j} J(w)$$

and this is performed for all values of $j \in [0, n]$. Note that α is the learning rate.

§1.4 LMS (Widrow-Hoff) Learning Rule

Let us solve for $\frac{\partial}{\partial w_j} J(\vec{w})$ such that we can substitute it into our formula for gradient descent.

$$\begin{aligned}\frac{\partial}{\partial w_j} J(\vec{w}) &= \frac{\partial}{\partial w_j} \frac{1}{2}(h(x) - y)^2 \\ &= (h(x) - y) \cdot \frac{\partial}{\partial w_j}(h(x) - y) \\ &= (h(x) - y) \cdot \frac{\partial}{\partial w_j}(\sum_{i=0}^{n} w_i x_i - y) \\ &= (h(x) - y) x_j\end{aligned}$$

Definition 1.3. For a single training example, the update rule is

$$w_j := w_j + \alpha(y^{(i)} - h(x^{(i)}))x_j^{(i)}$$

and this is known as the LMS (Least Mean Squares) and the Widrow-Hoff Learning Rule and **stochastic gradient descent**.

§1.5 Stochastic vs. Batch Gradient Descent

Note 1.4. Parameters may keep oscillating around the local extreme, but it will get there way faster than batch gradient descent.

Note 1.5. We can ensure that this algorithm converges if as we get close to the local extreme, let $\alpha \to 0$.

Definition 1.6. To modify this for more than one example and this is **batch gradient descent**.

$$w_j := w_j + \alpha \sum_{i=1}^{m} (y^{(i)} - h(x^{(i)}))x_j^{(i)}$$

Note 1.7. When the training set m is large, stochastic gradient descent is preferred over batch gradient descent.

§1.6 The Normal Equations

Instead of using the iterative algorithm of gradient descent, we can minimize our cost function $J(\vec{w})$ by explicitly taking derivatives with respect to the w_j and set them to 0.

§1.6.1 Matrix Derivatives

Suppose we have a function $f : \mathbb{R}^{m\times n} \to \mathbb{R}$ that maps from an $m \times n$ matrix to the real numbers, the derivative of f with respect to matrix A is

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{m1}} & \cdots & \frac{\partial f}{\partial A_{mn}} \end{bmatrix}$$

Example 1.8

Suppose

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

is a 2×2 matrix, and the function $f : \mathbb{R}^{2\times 2} \to \mathbb{R}$ is given by

$$f(A) = \frac{3}{2}A_{11} + 5A_{12}^2 + A_{21}A_{22}$$

then

$$\nabla_A f(A) = \begin{bmatrix} \frac{3}{2} & 10A_{12} \\ A_{22} & A_{21} \end{bmatrix}$$

Now let us introduce the **trace** operator such that the trace of a matrix A is the sum of its diagonal entries.

$$\operatorname{Tr} A = \sum_{i=1}^{n} A_{ii}$$

Lemma 1.9

$$\operatorname{Tr} AB = \operatorname{Tr} BA$$
$$\operatorname{Tr} ABC = \operatorname{Tr} CAB = \operatorname{Tr} BCA$$
$$\operatorname{Tr} A = \operatorname{Tr} A^T$$
$$\operatorname{Tr}(A + B) = \operatorname{Tr} A + \operatorname{Tr} B$$
$$\operatorname{Tr} aA = a \operatorname{Tr} A$$

Note that the fourth equation only applies to non-singular square matrices.

Lemma 1.10

$$\nabla_A \operatorname{Tr} AB = B^T$$
$$\nabla_{A^T} f(A) = (\nabla_A f(A))^T$$
$$nabla_A \operatorname{Tr}(ABA)^T C = CAB + C^T AB^T$$
$$\nabla_A \det(A) = \det(A)(A^{-1})^T$$

§1.6.2 Least Squares Revisited

We seek a closed form of $\vec{w}$ that minimizes $J(\vec{w})$. Let us rewrite $J(\vec{w})$ in matrix-vectorial notation.

Proof. Let X be an $m \times n$ matrix such that each column represents a training example.

$$X = \begin{bmatrix} (x^{(1)})^T & \cdots & (x^{(m)})^T \end{bmatrix}$$

and let

$$\vec{y} = \begin{bmatrix} y^{(i)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

be an $m \times 1$ vector. Recall that $h(x^{(i)}) = (x^{(i)})^T w$ and therefore

$$X\vec{w} - \vec{y} = \begin{bmatrix} h(x^{(1)}) - y^{(1)} \\ \vdots \\ h(x^{(m)}) - y^{(m)} \end{bmatrix}$$

Therefore,

$$\frac{1}{2}(X\vec{w} - \vec{y})^T (X\vec{w} - \vec{y}) - \frac{1}{2}\sum_{i=1}^{m}(h(x^{(i)} = y^{(i)})^2 = J(\vec{w})$$

Thus we've represented our cost function in terms of matrices, but now let us minimize it using the trace equations above.

$$\nabla_{\vec{w}J(\vec{w}} = \nabla_{\vec{w}}(X\vec{w} - \vec{w})^T (X\vec{w} - \vec{y})$$

$$= \frac{1}{2}\nabla_{\vec{w}}(\vec{w}^T X^T X\vec{w} - \vec{w}^T X^T \vec{y} - \vec{y}^T X\vec{w} + \vec{y}^T \vec{y})$$

by homogeneity, transpose of a matrix, and distributive property.

$$= \frac{1}{2}\nabla_{\vec{w}} \operatorname{Tr}(\vec{w}^T X^T X\vec{w} - \vec{w}^T X^T \vec{y} - \vec{y}^T X\vec{w} + \vec{y}^T \vec{y})$$

because the trace of a real number is just the real number

$$= \frac{1}{2}\nabla_{\vec{w}}(\operatorname{Tr} \vec{w}^T X^T X\vec{w} - 2\operatorname{Tr} \vec{y}^T X\vec{w})$$

using the property $\operatorname{Tr} A = \operatorname{Tr} A^T$.

$$= \frac{1}{2}(X^T X\vec{w} + X^T X\vec{w} - 2X^T\vec{y})$$

$$= X^T X - X^T\vec{y} \implies X^T X\vec{w} = X^T\vec{y} \implies \vec{w} = (X^T X)^{-1}X^T\vec{y}$$

□

§1.7 Probabilistic Interpretation: Why Squared Error

Suppose in our linear regression model

$$y^{(i)} = \vec{w}^T x^{(i)} + \epsilon^{(i)}$$

where $\epsilon^{(i)}$ is an error term. Let's now assume that $\epsilon^{(i)}$ follows a Gaussian distribution of mean $\mu = 0$ and variance σ^2, which is reasonable.

$$\epsilon^{(i)} \sim N(0, \sigma^2) \implies \mathbb{P}(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{(e^{(i)})^2}{2\sigma^2})$$

Now let us assume that each $\epsilon^{(i)}$ is **IID (independently and identically distributed)** from each other. This may not be entirely true, but it's good enough to get a good approximation for a model. Therefore, we can express the following

$$\mathbb{P}(y^{(i)} \mid x^{(i)}; \vec{w}) = \frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{(y^{(i)} - \vec{w}x^{(i)})^2}{2\sigma^2})$$

by substitution. Essentially, given $\vec{x}, \vec{w}$, what's the probability density of a particular house's price?

$$y^{(i)} \mid x^{(i)}; \vec{w} \sim N(\vec{w}^T x^{(i)}, \sigma^2)$$

The random variable $y^{(i)}$ given $x^{(i)}$ parameterized by $\vec{w}$ is that Gaussian. We're essentially modeling the price of a house using the random variable y.

Definition 1.11. Below, we have the definition of a **likelihood function** in terms of $\vec{w}$.

$$L(\vec{w}) = L(\vec{w}; X, \vec{y}) = \mathbb{P}(\vec{y}|X; \vec{w})$$

Because we assumed that all $\epsilon^{(i)}$ are IID, the probability of all of the observations in the training set is simply the product

$$L(\vec{w}) = \mathbb{P}(\vec{y}|X; \vec{w}) = \prod_{i=1}^{m}\mathbb{P}(\vec{y}|X; \vec{w} = \prod_{i=1}^{m}\frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{(y^{(i)} - \vec{w}x^{(i)})^2}{2\sigma^2})$$

Our best guess of parameters $\vec{w}$ is by choosing $\vec{w}$ such that we maximize $L(\theta)$ which intuitively makes sense because we want to maximize the probability that given our x value, and parameterized by the weight, our probability that our ouputted value of y is as high as possible.

Definition 1.12. Instead of maximizing $L(\vec{w})$, we will maximize the **log likelihood**.

$$l(\vec{w}) = \log L(\vec{w}) = \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{(y^{(i)} - \vec{w}x^{(i)})^2}{2\sigma^2})$$

Note 1.13. The product rule of logarithms

$$\log xy = \log x + \log y$$

$$l(\vec{w}) = \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{(y^{(i)} - \vec{w}x^{(i)})^2}{2\sigma^2})$$

$$l(\vec{w}) = \sum_{i=1}^{m} \left[\log \frac{1}{\sqrt{2\pi}\sigma} + \log \exp(-\frac{(y^{(i)} - \vec{w}x^{(i)})^2}{2\sigma^2})\right]$$

$$l(\vec{w}) = m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \vec{w}^T x^{(i)})^2$$

Note 1.14. Note that the first term is constant, so we wish to only consider the second term. Now, observe that in the second term, we can ignore the $\frac{1}{\sigma^2}$ because it is also constant. Finally, since the second term is negative, note that maximizing the negative of a term is the same as minimizing the term. Hence, maximizing $l(\vec{w})$ is equivalent to minimizing

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

which is $J(\vec{w})$, our original least squares cost function.

§2 Locally Weighted Linear Regression

§2.1 Fitting Your Data

You can either fit

- Linear of the form $w_0 + w_1 x_1$
- Quadratic of the form $w_0 + w_1 x_1 + w_2 x_2^2$
- Some custom fit?

Whatever way, the linear regression naturally fits these functions of input features in your dataset.

§2.2 Locally Weighted Linear Regression

Definition 2.1. **Locally Weighted Linear Regression** is an algorithm that modifies linear regression to fit non-linear functions.

Note 2.2. The more features we add does not always correspond to better performance because it could lead to overfitting.

Recall that the original cost function was

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

To make a prediction at an input x using locally weighted linear regression

- Fit a straight line through the local neighborhood of training examples close to x.
- Fit a line passing through the outputs y in that neighborhood, and make a prediction at x.
- Note that by focusing on the local neighborhood of input points, we attribute these points to influence the output prediction the most, but the other points can also be weighted lesser and utilized for our prediction.

Definition 2.3. We can represent this in a **weighted cost function** such that

$$J(\vec{w}) = \sum_{i=1}^{m} w'^{(i)}(y^{(i)} - \vec{w}^T x^{(i)})^2$$

where w' is a weighting function with values $\in [0, 1]$. A common choice for w' is

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

Note 2.4. The closer the neighbor ie. $|x^{(i)} - x| \to 0$ the more heavily weighted as in $w' \approx e^0 = 1$. The counter logic applies as well. The further a neighbor, the smaller the weight.

Note 2.5. The weight function is not a normal distribution, and the weights are not random variables.

§2.3 Bandwidth Parameter

Definition 2.6. The **bandwidth parameter** τ decides the width of the neighborhood.

- τ controls how quickly the weight of a training example $x^{(i)}$ falls off as distance from the query point x increases, and τ can have a big impact on potentially underfitting and overfitting
- τ is a hyperparameter of this algorithm, and it doesn't have to be bell-shaped.

§2.4 Applications

The main benefit of this algorithm is that fitting a non-linear dataset doesn't require you to manually fiddle with features.

Note 2.7. This works best when the number of features isn't too large, say 2 or 3.

In terms of computation and space complexity, you need to solve a linear system of equations of dimension equal to m so we may need to resort to scaling algorithms.

§3 Newton's Method

Definition 3.1. Gradient ascent is very effective, but it needs many iterations before convergence. **Newton's method** takes bigger steps and converges earlier when maximizing the likelihood function $l(\vec{w})$. Although there are less iterations, each iteration tends to be more expensive.

Lemma 3.2

Suppose we have a function $f : \mathbb{R} \to \mathbb{R}$ and a scalar value w. The update rule to find the zero of a function is the following

$$w := w - \frac{f(w)}{f'(w)}$$

The nautral interpretation of this formula is we take our function f and approximate it with a linear function tangent to f at the current value of w, and solving for where this linear function equals 0 (the x-intercept) gives us our next guess for w.

Now suppose we wish to maximize the log likelihood function $l(w)$. The maxima of $l(w)$ corresponds to points where its first derivative is equal to 0. Thus, let

$$f(w) = l'(w) \implies w := w - \frac{l'(w)}{l''(w)}$$

In terms of indices,

$$w^{(t+1)} = w^{(t)} - \frac{l'(w)}{l''(w)}$$

§3.1 Quadratic Convergence

Note that when you apply Newton's method, the iterations represent quadratic steps such that the number of significant digits that you have converged to the minimum double on a single iteration.

§3.2 Newton's Method for Vectors

Definition 3.3. If w is a vector and not a scalar, we have to generalize Newton's method to a multi-dimensional setting, and this is known as the **Newton-Raphson method**

$$\vec{w} := \vec{w} - H^{-1}\nabla_0 l(\vec{w})$$

where

- $\nabla_0 l(\vec{w})$ is the vector of partial derivatives with respect to the $w_i's$
- H is the Hessian square matrix defined as

$$H_{ij} = \frac{\partial^2 l(\vec{w})}{\partial w_i \partial w_j}$$

§4 The Classification Problem

Definition 4.1. Let us take this framework that we have developed and apply it to a **classification** problem. This differs from regression because the y values can only take on a discrete number of values. Let us simplify this even further by beginning with the binary classification problem such that $y \in 0, 1$ where 0 is the negative class and 1 is the positive class.

Note 4.2. The problem with applying linear regression to a classification problem and applying a threshold of 0.5 is that if there is a single outlier, the regression algorithm will adjust its straight line, which could drastically alter the decision boundary. Furthermore, the decision boundary for classification problems may not be clear cut.

§4.1 Logistic Regression

Definition 4.3. If we know output values will be between 0 and 1, it makes sense to choose our hypothesis function $h(x)$ as the following

$$h(x) = g(w^T x) = \frac{1}{1 + e^{-w^T x}}$$

where $g(z) = \frac{1}{1+e^{-z}}$ is known as the **logistic function** or the sigmoid function.

Lemma 4.4

When we will later try to adjust w such as to minimize our cost function, we must be able to compute the derivative of the cost function, and thereby we must be able to compute the derivative of the sigmoid function.

$$g'(z) = \frac{d}{dz}\frac{1}{1 + e^{-z}} = g(z)(1 - g(z))$$

A rigorous derivation can be found online.

§4.2 Update Rule

Let's use a similar technique of making some probabilistic assumptions and then fitting the parameters via a a maximum likelihood. Note that because y only takes 2 values, 0, and 1

$$(y^{(i)} = 1|x^{(i);w}) = h(x^{(i)})$$
$$(y^{(i)} = 0|x^{(i)}; w) = 1 - h(x^{(i)})$$

Assuming that the m training examples were generated independently, we have the likelihood function as follows

$$L(w) = \mathbb{P}(\vec{y}|X; w) = \prod_{i=1}^{m} \mathbb{P}(y^{(i)}|x^{(i)}; w) = \prod_{i=1}^{m} h(x^{(i)})^{y^{(i)}}(1 - h(x^{(i)}))^{1-y^{(i)}}$$

Note that it's easier instead to maximumze the log likelihood rather than the actual likelihood, as local extreme will occur in the same points in both.

$$l(w) = \log L(w) = \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))$$

Now we will use batch gradient ascent to obtain the update rule

$$w := w + \alpha \nabla_w l(w)$$

Note 4.5. Note that there's a plus sign because we are using gradient ascent, not descent, and here we are maximizing the log-likelihood function instead of minimizng the squared cost function like in linear regression.

Now if we plug in our sigmoid function value of $h(x)$, and then solve for $\frac{\partial l(w)}{\partial w_j}$ we get the following

$$\frac{\partial l(w)}{\partial w_j} = (y - h(x^{(i)}))x_j^{(i)}$$

which yields the update rule for an entire training set m of examples

$$w_j = w_j + \alpha \sum_{i=1}^{m} (y^{(i)} - h(x^{(i)})x_j^{(i)})$$

While this appears to be the same as the LMS update rule, it is not because now our hypothesis function is now not a linear function of $w^T x$. However, this is not a coincidence, and this update rule is a property of a bigger class of algorithms called generalized linear models.

§5 Generalized Linear Models

- In linear regression, $y|x; \vec{w} \sim N(\mu, \sigma^2)$
- In logistic regression $y|x; \vec{w} \sim \text{Bernoulli}(p)$

Both of these methods are part of a broader family of models, known as Generalized Linear Models.

§5.1 The Exponential Family

Definition 5.1. Let us first define **exponential family** distributions. A class of distributions is in the exponential family if their PDF can be written in the form.

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

where

- y is the data that the PDF is trying to model
- η is called the **natural (canonical) parameter** of the distribution
- $T(y)$ is called the **sufficient statistic** (often $T(y) = y$)
- $b(y)$ is the **base measure**
- $a(\eta)$ is called the **log partition function**

Rewriting the form above as

$$\mathbb{P}(y;\eta) = b(y)\frac{\exp(\eta^T T(y))}{\exp(a(\eta))}$$

such that the denominator plays the role of a normalization constant to ensure that the PDF integrates over y to 1.

Note 5.2. Note that

- y is a scalar
- η is only a vector when considering softmax regression (multiclass classification)
- $T(y)$ is a vector
- $b(y)$ and $a(\eta)$ are scalars

Now let us show that Bernoulli Distribution and Normal Distributions are examples of exponential family distributions.

§5.1.1 Bernoulli Distribution

The Bernoulli Distribution is a distribution used to model binary data.

$$P(y = 1) = p, P(y = 0) = 1 - p$$

$$E(X) = p, \text{Var}(X) = p(1 - p)$$

$$P(y) = p^y(1 - p)^{1-p}$$

is the pdf. We seek to take th pdf and convert it to an equation of the form that defines the distribution to be a member in the exponential family. Recall that that form is

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

Proof.

$$P(y) = p^y(1-p)^{1-p}$$
$$P(y) = \exp(y \log p + (1-y)\log(1-p))$$
$$P(y) = \exp((\log(\frac{p}{1-p}))y + \log(1-p))$$

- Base measure $b(y)$ is given by 1
- Sufficient statistic $T(y)$ is given by y
- Log partition function $a(\eta)$ is given by $\log(1-p)$
- Natural parameter η is given by $-\log \frac{p}{1-p}$

Note that if we solve the last equation for p instead of η, we obtain the sigmoid function such that

$$p = \frac{1}{1+e^{-\eta}}$$

Note that our log partition function is currently not in terms of η but we can get it to be so by plugging in our sigmoid function for p. Thus,

$$a(\eta) = -\log(1-p) = -\log(1 - \frac{1}{1+e^{-\eta}}) = \log(1+e^{\eta})$$

Therefore, the Bernoulli Distribution is a member of the exponential family. □

§5.1.2 Gaussian Distribution

Proving that the Gaussian Distribution is a member of the exponential family is a similar process. Write out the pdf, and simply use algebra to massage it into our desired form. In this case, note that $\eta \in \mathbb{R}^2$.

§5.1.3 Other Members of the Exponential Family

- The mulitnomial distribution, which we'll use in your discussion on softmax regression
- Regression $\rightarrow$ you could model the $y's$ as a Gaussian
- Binary classification $\rightarrow$ Bernoulli Distribution
- Poisson distribution for modelling count data which consits of non-negative integers (number of people who will enter your store in an hour)
- Gamma and exponential distributions for modeling continuous values such as time intervals or volume of an object
- Beta and Dirichlet distributions show up in Bayesian ML or Bayesian statistics

§5.1.4 Properties of the Exponential Family

- Performing maximum likelihood estimation on a distribution parameterized by the natural parameter yields a concave function.
- Minimizing the negative log likelihood yields a convex function. Note that NLL is essentially the cost function equivalent of maximum likelihood estimation.

Note 5.3. Note that

$$E(y;\eta) = \frac{\partial}{\partial \eta} a(\eta) \text{ and } \mathrm{Var}(y;\eta) = \frac{\partial^2}{\partial \eta^2} a(\eta)$$

These last two properties are especially noteworthy because normally when finding mean and variance, they require integration, but here we can find both with just differentation.

§5.2 Generalized Linear Models (GLMs)

GLMs are a natural extension of the exponential family. They build powerful models using a distribution that belongs to the exponential family and plugging it in at the output of a linear model, which models the output y as a linear function of the input features x.

- Depending on the task at hand, choose parameters $b, a, T(\cdot)$ based on the distribution that you'd like to model your output as.
- During training time, note that we're learning and tuning the parameters of $\vec{w}$. The output of thias model will become η, the natural parameter to our distribution.
- During test time, the mean of our distribution which is the first derivative of the log partition function with respect to η will become our prediction y.

§5.2.1 Update Rule

No matter the choice of the exponential family distribution that you decide to model your output y values as, the update rule remains the same

$$w_j := w_j + \alpha(y^{(i)} - h(x^{(i)})x_j^{(i)}$$

Note that this for stochastic gradient descent. For batch, just sum over all of the examples before updating. Simply initialize random weights and plug them into an appropriate hypothesis function and start learning right away.

Note 5.4. Newton's method is the most common MLE algorithm you would use with GLMs, assuming the number of futures is fewer than a thousand.

Observe that most of time, our sufficient statistic $T(y) = y$ and this implies that

$$g(\eta) = E(T(y);\eta) = E(y;\eta) = \mu = \frac{\partial}{\partial \eta} a(\eta)$$

where this function pair g, g^{-1} is known as the canonical link function, and g is a function that yields the distribution's mean as a function of the natural parameter.

§5.2.2 Parameterization Spaces

Note that now we have three parameterization spaces

1. $\vec{w} \to$ during the training phase of a GLM, we learn w.
2. $\eta \to$ where $\eta = \vec{w}^T \vec{x}$
3. Canonical parameters such as $p, (\mu, \sigma^2), \lambda$

§5.2.3 Constructing GLMs

Now we will actually discuss constructing GLMs to solve problems. Before we do, however, we must identify three assumptions about our conditional distribution y given x.

- $y|x; \vec{w} \sim$ Exponential Family
- $\eta = \vec{w}^T x$ and if η is vector valued then $\eta_i = \vec{w}_i^T x$ where $\vec{w} \in \mathbb{R}^n$ and $x \in \mathbb{R}^n$. This isn't necessarily a needed assumption, but more of a design pattern.
- Our goal is to output a prediction given by the mean of our distribution, in other words, $h(x) = E(y|x; \vec{w})$

§5.3 Linear Regresion (Gaussian Distribution)

Our goal use our assumptions and properties to begin from the fact that we wish to output a prediction given by the mean of our distribution. Working backwards will allow us to verify our hypothesis function. For example, in linear regression

$$h(x) = E(y|x; \vec{w}) = \mu = \eta = \vec{w}^T x$$

§5.4 Logistic Regression (Bernoulli Distribution)

$$h(x) = E(y|x; \vec{w}) = p = \frac{1}{1 + e^{-\eta}} = \frac{1}{1 + e^{-\vec{w}^T}}$$

§5.5 Softmax Regression (Multinomial Distribution)

Definition 5.5. Note that softmax regression is to multiclass regression in the same way that logistic regression is to binary classification. Let's derive a GLM for this problem, which we will start by expressing a multinomial as an exponential family distribution.

Let us denote k parameters $\phi_i, \cdots, \phi_k$. Note that $\sum_{i=1}^k \phi_i = 1$, and as such we will only parameterize $k-1$ parameters such that $\phi_k = 1 - \sum_{i=1}^{k-1} \phi_i$, but ϕ_k is not a parameter.
Now let us denote $T(y) \in \mathbb{R}^{k-1}$ such that they form a standard basis set

$$T(1) = e_1, T(2) = e_2, \cdots, T(k-1) = e_{k-1}, T(k) = \text{zero vector}$$

Now let us also define an indictor function $1\{\cdot\}$ which returns 1 if its argument is true and 0 otherwise.

$$\begin{aligned} P(y;\phi) &= \phi_1^{1\{y=1\}}\phi_2^{1\{y=2\}}\cdots\phi_k^{1\{y=k\}} \\ &= \phi_1^{T(1)}\cdots\phi_k^{1-\sum_{i=1}^{k=1}T(i)} \\ &= \exp(T(1)\log(\frac{\phi_1}{\phi_k}+\cdots+T(k-1)\log(\frac{\phi_{k-1}}{\phi_k}+\log(\phi_k)) \\ &= b(y)\exp(\eta^T T(y)-a(\eta)) \end{aligned}$$

where

$$n = \begin{pmatrix} \log(\phi_1/\phi_k) \\ \vdots \\ \log(\phi_{k-1}/\phi_k) \end{pmatrix}, a(\eta) = -\log(\phi_k), b(y) = 1$$

A little bit more algebraic manipulation gets you the formula

$$P(y=i|x;\vec{w}) = \frac{e^{\vec{w}_i^T x}}{\sum_{j=1}^k e^{\vec{w}_j^T x}}$$

§6 Generative Learning Algorithms

We have mainly observed learning algorithms that model the conditional distribution $P(y|x;w)$. For instance, with logistic regression, we have $h(x) = g(w^T x)$ where g is the sigmoid function. In this section, we'll focus on binary classification to talk about a different learning algorithm.

Consider a classification problem where we want to distinguish between malignant $y = 1$ and benign $y = 0$ tumors.

- All learning algorithms such as logistic regression, perceptron algorithm, or generalized learning model starts with randomly initialized parameters. Over the course of learning, the algorithm performs gradient descent where the decision boundary evolves until you obtain a boundary that separates the two classes.
- To classify a new sample, it checks on which side the of the decision boundary the new sample falls in, and makes an appropriate prediction.

Definition 6.1. These algorithms that try to learn $P(y|x)$ directly or map from the input space to the labels are known as **discriminative learning algorithms**.

The alternative method is to

- First, look at the malignant tumors and see what the features look like. Then look at benign tumors, and we can build a separate model of what benign tumors look like.

- To classify a new tumor, match the new sample against the malignant tumor model and the benign tumor model, and see whether the new sample is more similar to which model.

Definition 6.2. In this section, we'll talk about algorithms that model $P(x|y)$ and $P(y)$. These are called **generative learning algorithms**.

Note 6.3. In this algorithm, we observe x given the label y. Therefore, $P(x|y=0)$ models the distribution of benign tumors' features, and $P(x|y=1)$ models the distribution of malignant tumors' features.

Definition 6.4. The model also learns the **class prior**, or the independent probability $P(y)$. For example, when a patient walks into a hospital, the doctor already knows the probability that a patient will have a malignant vs benign tumor.

After modeling $P(y)$ (the class priors) and $P(x|y)$, the algorithm can use **Bayes' rule** to derive $P(y|x)$

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

Note 6.5. Note that you don't actually need to calculate the denominator $P(x) = P(x|y=1)P(y=1) + P(x|y=0)P(y=0)$ to make a prediction because it is constant with respect to y, but if you want to calculate probability then you need to calculate it.

The above equation will set the framework for **Gaussian discriminant analysis** and **Naive Bayes**.

§6.1 Gaussian Discriminant Analysis

Definition 6.6. The first generative learning algorithm we will observe is **Gaussian Discriminant Analysis** which can be used for continuous-valued features, like in tumor classification.

We'll assume that $P(x|y)$ is distributed according to a multivariate normal distribution.

§6.2 The Multivariate Gaussian Distribution

Definition 6.7. The **multivariate Gaussian distribution** seeks to model multiple random variables as a Gaussian.

Suppose X is distributed as a multivariate Gaussian such that $X \in \mathbb{R}^n$, parameterized by a mean vector $\mu \in \mathbb{R}^n$ and a covariance matrix $\Sigma \in \mathbb{R}^{n \times n}$ such that $\Sigma \geq 0$ is symmetric and positive definite.

$$X \sim N(\mu, \Sigma)$$

Lemma 6.8

The PDF of X is given by

$$P(X;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)\right)$$

where $|\Sigma| = \det\Sigma$ The expected value of X is given as

$$E(X) = \int_x xP(x;\mu,\Sigma)dx = \mu$$

$$\text{Cov}(X) = E(XX^T) - (E(X)(E(X))^T = \Sigma$$

Note 6.9. Note that a Gaussian with zero mean and identity covariance is known as the **standard normal distribution**.

Note 6.10. As the covariance Σ becomes larger, the Gaussian becomes wider and shorter, as the distribution must adapt because the PDF must always integrate to 1.

Note 6.11. By varying μ, we can shift the center of the Gaussian density.

§6.2.1 The Gaussian Discriminant Analysis (GDA) Model

When the input features x are continuous random variables, we can model $P(x|y)$ as a multivariate normal distribution.

$$x|y = 0 \sim N(\mu_0, \Sigma)$$

$$x|y = 1 \sim N(\mu_1, \Sigma)$$

$$y \sim \text{Bernoulli}(\phi)$$

Thus, the parameters of our GDA model are $\mu_0, \mu_1, \Sigma, \phi$.

Note 6.12. Note that we use the same covariance matrix Σ for both classes, but different mean vectors.

Fitting these parameters to our data will define $P(x|y)$ and $P(y)$. With sufficient estimations of parameters, then we can easily ascertain a tumor as benign or malignant because we know $P(x|y=0), P(x|y=1), P(y=0), P(y=1)$.
Now let us maximize our joint likelihood.

$$L(\phi,\mu_0,\mu_1,\Sigma) = \prod_{i=1}^{m} P(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma) = \prod_{i=1}^{m} P(x^{(i)|y^{(i)};\mu_0,\mu_1,\Sigma})P(y^{(i)};\phi)$$

Note 6.13. The difference between **cost functions** of a generative learning algorithm and a discriminative learning algorithm is that you're trying in generative, you're trying to choose parameters $\phi, \mu_0, \mu_1, \Sigma$ that maximize the joint likelihood $P(x, y; \phi, \mu_0, \mu_1, \Sigma)$ whereas in discriminative, you choose w to maximize the conditional likelihood that $P(y|x; w)$.

§7 Naive Bayes

In Gaussian Discriminant Analysis, where the feature vectors x were continuous and real valued, **Naive Bayes** deals with $x_i's$ that are discrete-valued, for example in the problem of spam classification. The given features will not be continuous. Let us begin by supposing we have a training dataset, now we would like to specify the feature vector using word embeddings and removing stop words. This set of words is called the vocabulary, so the dimension of x is equal to the size of the vocabulary.

Note 7.1. Note that we are using a generative model, so we wish to model $P(x|y)$. Suppose our vocabulary is 50000 words, then x is a 50000 dimensional binary vector, and modelling x explicitly with a multinomial distribution would yield far too many parameters.

Thus, we will assume that all $x_i's$ are conditionally independent given $y \implies$ and this is called the **Naive Bayes Assumption** and the resulting algorithm is the **Naive Bayes Classifier**.

$$P(x_1|y) = P(x_1|y, x_9)$$

and this is NOT the same as

$$P(x_1) = P(x_1|x_9)$$

Therefore,

$$P(x_1, \cdots, x_{50000}|y) = \prod_{i=1}^{m} P(x_i|y)$$

Observe that our model is parameterized by $\phi_{i|y=1} = P(x_i = 1|y = 1), \phi_{i|y=0} = P(x_1 = 1|y = 0), \phi_y = P(y = 1)$. Using joint likelihood once more, we can find the maximum likelihood estimates for the parameters below.

$$\phi_{j|y=1} = \frac{\sum_{i=1}^{m} 1\{x_j^{(i)} = 1 \ \ \& \ \ y^{(i)} = 1\}}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}}$$

which is the fraction of the spam in which the word j appears

$$\phi_{j|y=0} = \frac{\sum_{i=1}^{m} 1\{x_j^{(i)} = 1 \ \ \& \ \ y^{(i)} = 0\}}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}}$$

which is the fraction of the nonspam in which the word j appears

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

which is the fraction that is spam. To make a new prediction, we seek

$$P(y = 1|x) = \frac{P(x|y = 1)P(y = 1)}{P(x)}$$

by Bayes rule which simplifies to

$$P(y = 1|x) = \frac{(\prod_{i=1} P(x_i|y = 1))P(y = 1)}{(\prod_{i=1}^{n} P(x_i|y = 1))P(y = 1) + (\prod_{i=1}^{n} P(x_i|y = 0))P(y = 0)}$$

and then we pick whichever class is more likely.

§7.1 Laplace Smoothing

Assume your Naive Bayes model has never seen a word before, but it must estimate $P(x_i|y)$. It will estimate 0, simply because it's never seen the word, and this would cause $\prod_{i=1}^{n}(P(x_i)|y)$ to be zero, and when making predictions, this would yield $\frac{0}{0}$.

Note 7.2. It is statistically a bad idea to estimate the probability of an event to be 0 just because you have never seen it before in your finite training set.

§8 Support Vector Machines