1 Lecture 1 – overview

Supervised learning – given dataset (X_i, Y_i), find the mapping $X \to Y$.

Regression – Y is a continuous variable. Classification – Y is a discrete variable.

Regression/classification problems; regression/classification algorithms to find mapping.

Practically speaking, X is usually multi-dimensional (consists of multiple *features*). For example, in a cancer classification example, $(X_{1,i}, X_{2,i}) \to Y$ where X_1 is age, X_2 is tumor size, and Y is $(\text{malignant}, \text{benign})$. X_1 and X_2 are the input features, so X is two-dimensional.

The outputs Y_i are called the labels (in other words, supervised learning uses labeled data).

Other topics:

Learning strategy – systematic ways to improve algorithms

Unsupervised learning – you're given dataset (X_i) without Y_i, and you need to find patterns in (X_i). This uses unlabeled data.

Deep learning

Reinforcement learning – like training a pet (reward when pet does something good, punish when pet does something bad)

2 Lecture 2 – linear regression and gradient descent

2.1 Batch gradient descent

Supervised learning – take a training set (of data), run it through the training algorithm, which produces a hypothesis h s.t. $h(x) = \hat{y}$. Example training set consisting of sizes and prices of houses: x is the size of the house and $\hat{y}$ is the predicted price.

In linear regression,

$$h(x) = \theta_0 + \theta_1 x$$

Let's say you have an additional input feature, no. of bedrooms. Then denote the input features size and no. bedrooms by x_1 and x_2, and

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

This can be compactly written as

$$h(x) = \sum_{j=0}^{2} \theta_j x_j$$

$$x_0 \triangleq 1$$

In matrix form,

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}, x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow h = \theta^T x$$

θ is known as the parameters of the learning algorithm, m is the no. of training examples (data sample size), x is the inputs or features, n is the number of input features, y is the output or target variable.

(x, y) denotes one training example; $\left(x^{(i)}, y^{(i)}\right)$ denotes the i^{th} training example.

Then

$$h(x) = \sum_{j=0}^{n} \theta_j x_j$$

And the dimensions of θ and x are $n + 1$.

Simply, we want to choose θ s.t. $h(x) = \hat{y} \approx y$ for the training examples.

Linear regression minimizes the squared error (least-squared error):

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} \left(h\left(x^{(i)}\right) - y^{(i)}\right)^2$$

$\frac{1}{2}$ term is added just to make the math of later derivations easier (it changes nothing). $J(\theta)$ is known as the cost function, and our goal is to choose θ to minimize $J(\theta)$.

$$\underset{\theta}{\text{minimize}}\, J(\theta)$$

We'll use an algorithm called gradient descent:

1. Start with some θ (say $\theta = \vec{0}$)
2. Keep changing θ to reduce $J(\theta)$

Depending on your initial θ, you can reach different local minima/optima for a general cost function. However, for linear regression, the cost function is quadratic so there is only one optimum/minimum (which is great for convergence).

Update equation:

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

$:=$ means assignment, j is the index for the input features, and α is the learning rate.

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta_j} &= \frac{\partial}{\partial \theta_j}\left[\frac{1}{2}\sum_{i=1}^{m}\left(h\left(x^{(i)}\right) - y^{(i)}\right)^2\right] = \sum_{i=1}^{m}\left[\frac{\partial h\left(x^{(i)}\right)}{\partial \theta_j} \times \left(h\left(x^{(i)}\right) - y^{(i)}\right)\right] \\ &= \sum_{i=1}^{m}\left[\frac{\partial\left(\sum_{j=0}^{n}\theta_j x_j\right)}{\partial \theta_j} \times \left(h\left(x^{(i)}\right) - y^{(i)}\right)\right] = \sum_{i=1}^{m}\left[\left(h\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}\right] \\ &= \sum_{i=1}^{m}\left[\left(\sum_{j=0}^{n}\theta_j x_j^{(i)} - y^{(i)}\right)x_j^{(i)}\right]\end{aligned}$$

Then

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m}\left[\left(h\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}\right]$$

Repeat this until convergence. Each iteration updates θ_j for all j.

The direction of the gradient (steepest descent) is always orthogonal to the contour (see 35:30).

When α is too large, you can overshoot the optimum. This is usually the problem if $J(\theta)$ increases with iterations.

When α is too small, you need too many iterations to converge.

In practice, you try a few different α (that is, learning rate is empirical). Ng recommends a doubling or tripling scale (e.g. 0.01, 0.02, 0.04, ...).

Because the update sums over all training samples, this algorithm is also called batch gradient descent. The main disadvantage of batch gradient descent is the update gets computationally expensive (perhaps prohibitively) as your dataset size increases.

2.2 Stochastic gradient descent

For $i = 1,2, \ldots m,$

$$\theta_j := \theta_j - \alpha\left(h\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

As in batch gradient processing, every iteration updates for all j, but the difference is that each iteration uses only one training example to update. If you reach the end of the dataset, start over from $i = 1$. Stochastic gradient descent is much faster and more practical than batch for large datasets. However, the path that stochastic gradient descent takes will be "noisy", and it'll never quite converge, unlike batch (47:00).

To get better convergence, you can reduce the learning rate over time so there's less "noise".

2.3 Normal equation (non-iterative solution for linear regression)

2.3.1 First, some notation (nabla operator)

As an example of the general notation we'll use in this class, let A be a 2x2 matrix and f is a function that maps a 2x2 matrix to a real number:

$$A \in \mathbb{R}^{2\times 2}$$

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

$$f: \mathbb{R}^{2\times 2} \to \mathbb{R}$$

e.g. let $A = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$ and $f(A) = A_{11} + A_{12}^2 = 5 + 6^2$.

Then the derivative of any function f w.r.t. its input argument A is a matrix, of the same shape as A, consisting of the partial derivatives of f w.r.t. the elements of A:

$$\nabla_A\left(f(A)\right) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \frac{\partial f}{\partial A_{12}} \\ \frac{\partial f}{\partial A_{21}} & \frac{\partial f}{\partial A_{22}} \end{bmatrix}$$

In our example,

$$\nabla_A\left(f(A)\right) = \begin{bmatrix} 1 & 2A_{12} \\ 0 & 0 \end{bmatrix}$$

2.3.2 Minimizing the cost function

For the cost function $J(\theta)$ that maps $\theta \in \mathbb{R}^{n+1}$ to $\mathbb{R}$, the derivative of $J(\theta)$ w.r.t. θ is

$$\nabla_\theta\big(J(\theta)\big) = \begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{bmatrix}$$

We want to minimize the cost function, so

$$\nabla_\theta\big(J(\theta)\big) = \begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{bmatrix} \overset{\text{set}}{=} \vec{0}$$

If A is square, trace of A, $\operatorname{tr} A$, is equal to the sum of the diagonals. Some properties of trace:

1. $\operatorname{tr} A = \operatorname{tr} A^T$
2. If $f(A) = \operatorname{tr} AB$, where B is some fixed matrix, then $\nabla_A\big(f(A)\big) = B^T$.

This is easy to show with 2x2 matrices (and easy to see how it extends to any square matrices).

$$AB = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & \text{don't care} \\ \text{don't care} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$f(A) = \operatorname{tr} AB = a_{11}b_{11} + a_{12}b_{21} + a_{21}b_{12} + a_{22}b_{22}$$

$$\nabla_A\big(f(A)\big) = \begin{bmatrix} b_{11} & b_{21} \\ b_{12} & b_{22} \end{bmatrix} = B^T$$

3. $\operatorname{tr} AB = \operatorname{tr} BA$
4. $\operatorname{tr} ABC = \operatorname{tr} CAB$ (cyclic permutation property)
5. $\nabla_A(\operatorname{tr} AA^T C) = CA + C^T A$ (Ng likens this property to $\frac{d}{da}a^2 c = 2ac$, but with matrices)

For linear regression,

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

$x^{(i)}$ are column vectors containing the input features of the training examples. Transpose and stack them to form X (Ng calls this the design matrix):

$$X = \begin{bmatrix} {x^{(1)}}^T \\ {x^{(2)}}^T \\ \vdots \\ {x^{(m)}}^T \end{bmatrix} \in \mathbb{R}^{m \times (n+1)}$$

$$\theta \in \mathbb{R}^{(n+1) \times 1}$$

$$X\theta = \begin{bmatrix} {x^{(1)}}^T \theta \\ {x^{(2)}}^T \theta \\ \vdots \\ {x^{(m)}}^T \theta \end{bmatrix} = \begin{bmatrix} h\left(x^{(1)}\right) \\ h\left(x^{(2)}\right) \\ \vdots \\ h\left(x^{(m)}\right) \end{bmatrix} \in \mathbb{R}^{m \times 1}$$

Each element of $X\theta$ is the prediction of the training algorithm (h is the hypothesis).

Let $\vec{y}$ be the vector of labels:

$$\vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

Then $J(\theta)$ can be written in matrix notation as

$$J(\theta) = \frac{1}{2}(X\theta - y)^T (X\theta - y)$$

The errors between the labels and the predictions are

$$X\theta - y = \begin{bmatrix} h\left(x^{(1)}\right) - y^{(1)} \\ h\left(x^{(2)}\right) - y^{(2)} \\ \vdots \\ h\left(x^{(m)}\right) - y^{(m)} \end{bmatrix}$$

Then $(X\theta - y)^T (X\theta - y)$ is the sum of squared errors.

Now let's find the θ that minimizes $J(\theta)$:

$$\nabla_\theta\big(J(\theta)\big) = \nabla_\theta \left(\frac{1}{2}(X\theta - y)^T (X\theta - y)\right) = \frac{1}{2}\nabla_\theta\big((\theta^T X^T - y^T)(X\theta - y)\big)$$
$$= \frac{1}{2}\nabla_\theta(\theta^T X^T X\theta - \theta^T X^T y - y^T X\theta + y^T y) = \frac{1}{2}\nabla_\theta(\theta^T X^T X\theta - \theta^T X^T y - y^T X\theta)$$

At this point, recall that $\theta \in \mathbb{R}^{(n+1)\times 1}, X \in \mathbb{R}^{m\times(n+1)}, y \in \mathbb{R}^{m\times 1}$. All of the expressions inside the derivative reduce to one real value (as expected, since $J(\theta)$ maps to sum squared error). This means

$$(\theta^T X^T y)^T = y^T X\theta \in \mathbb{R}$$

Then

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

For the second expression, note that $\nabla_x(b^T x) = b$ (at least, if b and x are column vectors). For the first expression, note that $\nabla_x(x^T Ax) = 2Ax$ (see problem set 0).

Then

$$\frac{1}{2}\nabla_\theta(\theta^T X^T X\theta - 2(X^T y)^T\theta) = \frac{1}{2}[2X^T X\theta - 2X^T y] = X^T X\theta - X^T y$$

$$X^T X\theta - X^T y \overset{\text{set}}{=} \vec{0}$$

The normal equation is

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

Then the optimum θ is

$$\theta = (X^T X)^{-1} X^T y$$

If $X^T X$ is not invertible, then you have redundant features (features are not linearly independent). You can use pseudo-inverse to still get an answer, but it's best to figure out which features are repeated and get rid of them.

2.4 Linear regression summary

1. Fit θ to minimize the cost function $J(\theta) = \frac{1}{2}\sum_i \left(y^{(i)} - \theta^T x^{(i)}\right)^2$ (also known as least-squared error)
    a. This is equivalent to maximizing $\mathcal{L}(\theta)$, the likelihood of θ, also known as the maximum likelihood estimate (MLE), under the assumptions that the errors are Gaussian and iid.
2. Given x, make the prediction $h = \hat{y} = \theta^T x$

3 Lecture 3 – locally weighted and logistic regression

3.1 Fitting nonlinear curves

Let's say your dataset is housing price vs. size, where size is the one input feature. If the data isn't linear, then $\theta_0 + \theta_1 x_1$ may result in poor fitting. Some other options:

1. $\theta_0 + \theta_1 x + \theta_2 x^2$
2. $\theta_0 + \theta_1 x + \theta_2 \sqrt{x}$

x^2 will eventually result in a parabolic shape, so perhaps $\sqrt{x}$ may work better. You can rewrite these in the same format as linear regression, $\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2$, where $x_0 = 1$, x_1 is size, and $x_2 = x^2$ or $\sqrt{x}$. Then you can use the same process as linear regression to find the optimum parameters, θ. Later in the course, we'll talk about feature selection algorithm, which automatically decide which features to use.

This is like DPD.

3.2 Locally weighted regression

This is one way of fitting a nonlinear curve.

3.2.1 PLA vs. non-PLA

Sometimes we distinguish b/w "parametric" and "non-parametric" learning algorithms. Short-hand: PLA, non-PLA.

1. Parametric: fit a fixed set of parameters, θ_i, to the data, e.g. linear regression
2. Non-parametric: the amount of data/parameters you need to keep grows linearly with the size of the data, e.g. locally weighted regression

In PLA, once you find θ_i, you can erase the training set and make predictions using just θ_i. In non-PLA, the data you need to store grows linearly with training set size, so non-PLA is impractical for large datasets since you need the data to make predictions. The advantage of non-PLA is it's good for fitting nonlinear curves without worrying about feature selection.

3.2.2 Linear vs. locally weighted regression

In linear regression, you look at the whole dataset and

1. Fit θ to minimize $\frac{1}{2}\sum_i \left(y^{(i)} - \theta^T x^{(i)}\right)^2$
2. Given x, make the prediction $h = \hat{y} = \theta^T x$

In locally weighted regression, you look mainly at the datapoints close to your x of interest and draw a straight line based on those close-in data points. You'll also look at far-off points, but those will have lower weights. Your fit changes based on your x of interest.

Formally,

1. Fit θ to minimize $\sum_i w^{(i)}\left(y^{(i)} - \theta^T x^{(i)}\right)^2$, where $w^{(i)}$ is the weighting function
2. A default candidate for the weighting function is $w^{(i)} = \exp\left(-\frac{\left(x^{(i)}-x\right)^2}{2\tau^2}\right) \in (0,1]$
    a. When $\left|x^{(i)} - x\right|$ is small, $w^{(i)} \approx 1$
    b. When $\left|x^{(i)} - x\right|$ is large, $w^{(i)} \approx 0$
    c. This is also the shape of a Gaussian bell curve, but is unrelated to Gaussian density, which integrates to 1. Peak of the bell curve is at x.

Note that in locally weighted regression, you still look at the whole dataset, but your focus is set by the weighting function.

This is like weighting different parts of the desired filter response based on the ripple/rejection specifications.

How do you choose how wide of a Gaussian bell curve to use? We use the bandwidth parameter τ. Increasing τ means widening the curve. The choice of τ can cause underfitting (causes a very jagged overall fit) or overfitting (causes a very smooth overall fit).

Locally weighted regression is better for low-dimensional dataset (n small), and you don't want to think about feature selection.

3.3 Why do we use least squares for linear regression? The probabilistic interpretation

Assume $y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$, where $\epsilon^{(i)}$ is the error term that comprises unmodeled effects and random noise.

Assume that $\epsilon^{(i)} \sim N(0, \sigma^2)$, i.e. the error is normally distributed with mean 0 and variance σ^2. Then the probability density (or pdf, which integrates to 1) is

$$P\left(\epsilon^{(i)}\right) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{{\epsilon^{(i)}}^2}{2\sigma^2}\right)$$

Assume that $\epsilon^{(i)}$ are iid (independent and identically distributed). That is, the error terms for the data points are all independent and have the same pdfs. This usually won't be strictly true (e.g. house prices on the same street may not be independent).

This implies that the probability of $y^{(i)}$ given $x^{(i)}$ and parameterized by θ is

$$P\left(y^{(i)}\middle|x^{(i)};\theta\right)=\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{\left(y^{(i)}-\theta^{T}x^{(i)}\right)^{2}}{2\sigma^{2}}\right)$$

In other words, given $x^{(i)}$ and parameterized by θ, the random variable $y^{(i)}$ is normally distributed with mean $\theta^{T}x^{(i)}$ and variance σ^{2}:

$$\left(y^{(i)}\middle|x^{(i)};\theta\right)\sim N\left(\theta^{T}x^{(i)},\sigma^{2}\right)$$

We separate $x^{(i)}$ and θ by a semicolon because θ is not a random variable; we are conditioning $y^{(i)}$ on $x^{(i)}$ but not θ.

To get the true $y^{(i)}$, you take the estimate $\theta^{T}x^{(i)}$ and add some noise.

The likelihood of θ is given by

$$\mathcal{L}(\theta)=P(y|x;\theta)=\prod_{i=1}^{m}P\left(y^{(i)}\middle|x^{(i)};\theta\right)=\prod_{i=1}^{m}\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{\left(y^{(i)}-\theta^{T}x^{(i)}\right)^{2}}{2\sigma^{2}}\right)$$

Mathematically, the likelihood of θ is equivalent to the probability of $\left(y^{(i)}\middle|x^{(i)};\theta\right)$. Why do we use different terms then? If we view $P(y|x;\theta)$ as a function of both parameters and data, when we fix the data (as in a training set) while varying the parameters, we call it likelihood of the parameters. When we fix the parameters while varying the data, we call it probability of the data.

Let the "log likelihood" be given by

$$\begin{aligned}\ell(\theta)=\log\mathcal{L}(\theta)&=\log\left[\prod_{i=1}^{m}\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{\left(y^{(i)}-\theta^{T}x^{(i)}\right)^{2}}{2\sigma^{2}}\right)\right]\\&=\sum_{i=1}^{m}\left[\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right)+\log\left(\exp\left(-\frac{\left(y^{(i)}-\theta^{T}x^{(i)}\right)^{2}}{2\sigma^{2}}\right)\right)\right]\rightarrow\end{aligned}$$

$$\ell(\theta)=m\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right)+\sum_{i=1}^{m}\left(-\frac{\left(y^{(i)}-\theta^{T}x^{(i)}\right)^{2}}{2\sigma^{2}}\right)$$

Maximum likelihood estimation (MLE) is a popular parameter estimation technique in statistics – that is, choose θ to maximize $\mathcal{L}(\theta)$.

Since the log function is strictly monotonic, maximizing $\ell(\theta)$ is equivalent to maximizing $\mathcal{L}(\theta)$. Maximizing $\ell(\theta)$ is also easier than maximizing $\mathcal{L}(\theta)$. From the equation for $\ell(\theta)$, it's

obvious that maximizing $\ell(\theta)$ is equivalent to minimizing $\sum_{i=1}^{m}\left(y^{(i)} - \theta^T x^{(i)}\right)^2$, which is the cost function for linear regression.

In other words, minimizing the sum of the squared errors is equivalent to maximizing the likelihood of θ.

3.3.1 My explanation of MLE and application to classification

You're trying to choose θ s.t. you minimize the error b/w hypothesis and data (for linear regression, you minimize the sum squared error). In other words, you're trying to fit the model, which is defined by θ, as closely as possible to the training data.

From a probability perspective, for a given sample in the training data, if you assume $h^{(i)}$ is the "true" value of $y^{(i)}$, which is corrupted by Gaussian noise $\epsilon^{(i)}$, then the probability of $y^{(i)}$ decreases as $h^{(i)}$ and $y^{(i)}$ diverge. In other words, if your model gives a prediction $h^{(i)}$ very far from $y^{(i)}$, the probability of this happening is low since the chance of such a large error is low.

Assuming $\epsilon^{(i)}$ are iid, the joint distribution of the training data is the product of the individual distributions. The explanation for the entire training set is the same: if you assume your model's predictions are true but corrupted by noise, the bigger the differences between predictions and training data, the lower the probability. MLE chooses the model (parameterized by θ) to maximize the probability, which is equivalent to minimizing the sum squared error between prediction and data.

For classification where $y^{(i)}$ is a discrete output set, instead of predicting the value of $y^{(i)}$, $h^{(i)}$ predicts the probability of $y^{(i)}$. For a given sample $\left(x^{(i)}, y^{(i)}\right)$, $h^{(i)} = P\left(y^{(i)} \middle| x^{(i)}; \theta\right)$. In other words, how likely is this sample to happen based on your model?

For the entire training set, assuming the samples are iid, the joint distribution is the product of the individual distributions. In other words, how likely is this training set based on your model? As in regression, MLE chooses the model (parameterized by θ) to maximize this probability.

3.4 Logistic regression for classification

Start with binary classification:

$$y \in \{0,1\}$$

Ng says that while some people use linear regression for classification, he thinks it's a bad idea and never does it himself.

Ng says the two most common algorithms he uses are linear regression and logistic regression.

In logistic regression, we want $h_\theta(x) \in [0,1]$ (we want the hypothesis to output values in the real set between 0 and 1).

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

Where $g(z)$ is the sigmoid or logistic function (these mean exactly the same thing):

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

As $z \to -\infty, g(z) \to 0$. As $z \to \infty, g(z) \to 1$. $g(0) = 0.5$. So $g(z) \in (0,1)$.

It will be useful to calculate the first derivative of $g(z)$:

$$g'(z) = \frac{d}{dz}\left(\frac{1}{1 + e^{-z}}\right) = \frac{d}{dz}((1 + e^{-z})^{-1}) = -(1 + e^{-z})^{-2} * \frac{d}{dz}(e^{-z}) = e^{-z}(1 + e^{-z})^{-2}$$
$$= \frac{e^{-z}}{(1 + e^{-z})^2} = g^2(z)\frac{1 - g(z)}{g(z)} = g(z)\big(1 - g(z)\big)$$

In linear regression, the hypothesis takes the form of $\theta^T x$ because we assume the output can be modeled as a linear function of the inputs. However, $\theta^T x$ can take on any value. In logistic regression, we pass $\theta^T x$ through the logistic function to force the output to be between 0 and 1. Essentially, the sigmoid function maps $\mathbb{R}$ to a value between 0 and 1, and as we'll see below, this value is a probability of an event.

To develop the logistic regression algorithm, we'll use the same probability framework from linear regression – define an expression for probability/likelihood and apply MLE.

$$P(y = 1|x;\theta) = h_\theta(x)$$

$$P(y = 0|x;\theta) = 1 - h_\theta(x)$$

Since $y \in \{0,1\}$, we can merge these two equations (mathematical way of representing if/else):

$$P(y|x;\theta) = h(x)^y\big(1 - h(x)\big)^{1-y}$$

Assuming all data points are iid, we can write the expression for likelihood as

$$\mathcal{L}(\theta) = P(y|x;\theta) = \prod_{i=1}^{m} h_\theta\big(x^{(i)}\big)^{y^{(i)}}\left(1 - h_\theta\big(x^{(i)}\big)\right)^{1-y^{(i)}}$$

$$\ell(\theta) = \log \mathcal{L}(\theta) = \sum_{i=1}^{m} \left[y^{(i)} \log h_\theta\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right) \right]$$

Choose θ to maximize $\ell(\theta)$. We'll use batch gradient ascent. For each parameter in θ,

$$\theta_j := \theta_j + \alpha \frac{\partial \ell(\theta)}{\partial \theta_j}$$

Unlike linear regression, where we wanted to minimize the cost function, here we want to maximize the log likelihood. That's why it's $+$ instead of $-$.

Let's calculate $\frac{\partial \ell(\theta)}{\partial \theta_j}$.

$$\begin{aligned}
\frac{\partial \ell(\theta)}{\partial \theta_j} &= \frac{\partial}{\partial \theta_j} \left(\sum_{i=1}^{m} \left[y^{(i)} \log h_\theta\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right) \right] \right) \\
&= \sum_{i=1}^{m} \frac{\partial}{\partial \theta_j} \left[y^{(i)} \log h_\theta\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right) \right]
\end{aligned}$$

By log, Ng means ln, and $\frac{d(\ln x)}{dx} = \frac{1}{x}$, and recall that $h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$.

Then we have

$$\begin{aligned}
&\sum_{i=1}^{m} \frac{\partial}{\partial \theta_j} \left[y^{(i)} \log g\left(\theta^T x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - g\left(\theta^T x^{(i)}\right)\right) \right] \\
&= \sum_{i=1}^{m} \left(\frac{y^{(i)}}{g\left(\theta^T x^{(i)}\right)} - \frac{\left(1 - y^{(i)}\right)}{1 - g\left(\theta^T x^{(i)}\right)} \right) \frac{\partial \left(g\left(\theta^T x^{(i)}\right) \right)}{\partial \theta_j} \\
&= \sum_{i=1}^{m} \left(\frac{y^{(i)}}{g\left(\theta^T x^{(i)}\right)} - \frac{\left(1 - y^{(i)}\right)}{1 - g\left(\theta^T x^{(i)}\right)} \right) g\left(\theta^T x^{(i)}\right) \left(1 - g\left(\theta^T x^{(i)}\right)\right) \frac{\partial \left(\theta^T x^{(i)}\right)}{\partial \theta_j} \\
&= \sum_{i=1}^{m} \left(y^{(i)} \left(1 - g\left(\theta^T x^{(i)}\right)\right) - \left(1 - y^{(i)}\right) g\left(\theta^T x^{(i)}\right) \right) x_j^{(i)} \\
&= \sum_{i=1}^{m} \left(y^{(i)} - y^{(i)} g\left(\theta^T x^{(i)}\right) - g\left(\theta^T x^{(i)}\right) + y^{(i)} g\left(\theta^T x^{(i)}\right) \right) x_j^{(i)} \\
&= \sum_{i=1}^{m} \left(y^{(i)} - g\left(\theta^T x^{(i)}\right) \right) x_j^{(i)}
\end{aligned}$$

Then we have

$$\frac{\partial \ell(\theta)}{\partial \theta_j} = \sum_{i=1}^{m} \left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right) x_j^{(i)}$$

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right) x_j^{(i)}$$

$\ell(\theta)$ does not have local maxima; the function is always concave (face down). This is one reason why we use the logistic function, which guarantees this property of $\ell(\theta)$.

This update function looks exactly like linear regression, but keep in mind h_θ is different – $h_\theta(x) = g(\theta^T x)$.

Linear and logistic regression belong to a class of algorithms called generalized linear models that do not have local optima and have the same form of update function (will be explained in a later lecture).

Logistic regression does not have a normal equation equivalent (there's no way to find the optimal θ in one shot).

3.5 Newton's method

Gradient descent takes a long time to converge because it takes baby steps. Newton's method allows us to take larger jumps and converge more quickly (but each iteration is more complicated).

Let's say you have a function f and you want to find θ s.t. $f(\theta) = 0$. Newton's method solves this problem.

In logistic regression, we want to maximize $\ell(\theta)$. At the maximum, derivative is zero, so equivalently, we want $\ell'(\theta) = 0$. Therefore, we'll set $f(\theta) = \ell'(\theta)$.

Let $\theta^{(k)}$ denote the k^{th} iteration of θ starting from $k = 0$. Let f be the function.

1. Find the line tangent to $f\left(\theta^{(0)}\right)$
2. Find the x intercept of the line – this becomes $\theta^{(1)}$
3. Keep repeating 1 and 2

Let Δ be the difference b/w $\theta^{(1)}$ and $\theta^{(0)}$, i.e. $\theta^{(1)} := \theta^{(0)} - \Delta$.

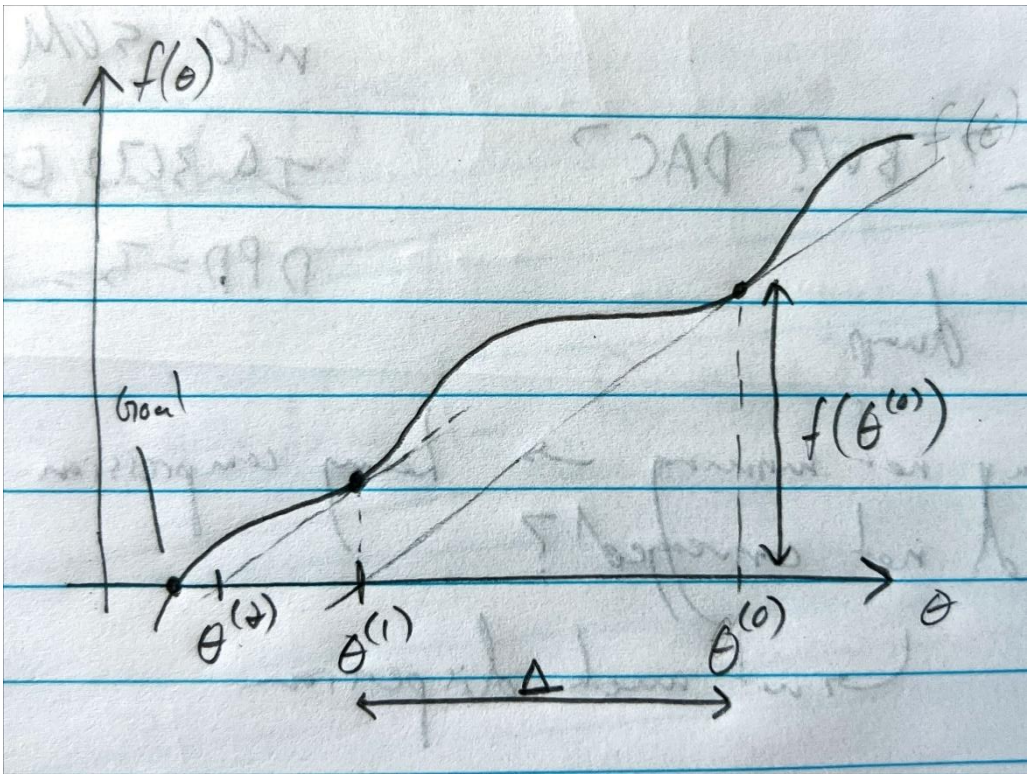
From calculus, we know the slope of the line at $\theta^{(0)}$ is equal to rise over run. The line is defined by $(\theta^{(0)}, f\left(\theta^{(0)}\right)$ and $\left(\theta^{(1)}, 0\right)$:

$$f'\left(\theta^{(0)}\right) = \frac{f\left(\theta^{(0)}\right) - 0}{\theta^{(0)} - \theta^{(1)}} = \frac{f\left(\theta^{(0)}\right)}{\Delta}$$

Then $\Delta = \frac{f(\theta^{(0)})}{f'(\theta^{(0)})}$, and

$$\theta^{(1)} := \theta^{(0)} - \frac{f(\theta^{(0)})}{f'(\theta^{(0)})}$$

Graphically,



For one-dimensional θ, each iteration of Newton's method is defined as

$$\theta^{(t+1)} := \theta^{(t)} - \frac{f(\theta^{(t)})}{f'(\theta^{(t)})}$$

Applying Newton's method to logistic regression, we have

$$\theta^{(t+1)} := \theta^{(t)} - \frac{\ell'(\theta^{(t)})}{\ell''(\theta^{(t)})}$$

Newton's method has the property of "quadratic convergence" – according to Ng, the name isn't important or meaningful. What this means: if the error at the current iteration is 0.01 (error from true $f(\theta) = 0$), then the error at the next iteration can be 0.0001, and then 0.00000001 after two iterations.

Under certain assumptions (smooth function that's close to quadratic), the number of significant digits in the error doubles per iteration. This is called quadratic convergence. When you get close to the optimum, Newton's method converges rapidly. This is why Newton's method requires few iterations.

When θ is a vector $\in \mathbb{R}^{(n+1)\times 1}$, Newton's method becomes

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

$\nabla_\theta \ell$ is a vector of partial derivatives $\in \mathbb{R}^{(n+1)\times 1}$, and H is the Hessian matrix $\in \mathbb{R}^{(n+1)\times(n+1)}$.

Let's calculate $\nabla_\theta \ell$.

$$\nabla_\theta \ell = \begin{bmatrix} \frac{\partial}{\partial \theta_1}\ell \\ \vdots \\ \frac{\partial}{\partial \theta_{n+1}}\ell \end{bmatrix}$$

From logistic regression, we've already calculated

$$\frac{\partial \ell(\theta)}{\partial \theta_j} = \sum_{i=1}^{m}\left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right)x_j^{(i)}$$

Then

$$\nabla_\theta \ell = \begin{bmatrix} \sum_{i=1}^{m}\left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right)x_1^{(i)} \\ \vdots \\ \sum_{i=1}^{m}\left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right)x_{n+1}^{(i)} \end{bmatrix}$$

H is the matrix of partial derivatives where $H_{ij} = \frac{\partial^2 \ell}{\partial \theta_i \partial \theta_j}$.

$$\frac{\partial^2 \ell}{\partial \theta_k \partial \theta_j} = \frac{\partial}{\partial \theta_k}\left(\frac{\partial}{\partial \theta_j}\ell\right) = \frac{\partial}{\partial \theta_k}\left(\sum_{i=1}^{m}\left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right)x_j^{(i)}\right) = \sum_{i=1}^{m} -x_j^{(i)}\frac{\partial}{\partial \theta_k}\left(g\left(\theta^T x^{(i)}\right)\right)$$

$$= \sum_{i=1}^{m} -x_j^{(i)} g\left(\theta^T x^{(i)}\right)\left(1 - g\left(\theta^T x^{(i)}\right)\right)\frac{\partial}{\partial \theta_k}\left(\theta^T x^{(i)}\right)$$

$$\frac{\partial^2 \ell}{\partial \theta_k \partial \theta_j} = \sum_{i=1}^{m} -x_j^{(i)} x_k^{(i)} g\left(\theta^T x^{(i)}\right)\left(1 - g\left(\theta^T x^{(i)}\right)\right)$$

In high-dimensional problems, each step in Newton's method becomes expensive since you need to invert H. Some rules of thumb: Newton's method is good for n around 10, around 15. For large n, gradient descent is preferred.

4 Lecture 4 – perceptron and generalized linear model

4.1 Perceptron

Perceptron algorithm isn't widely used in practice because it doesn't have a probabilistic interpretation, and its ability to classify is limited, but it's useful to understand.

In lecture 3, we saw that logistic regression uses the sigmoid function $g(z)$, where $z = \theta^T x^{(i)}$, to map $\mathbb{R}$ to a value between 0 and 1, and that value is a probability.

$$\text{sigmoid } g(z) = \frac{1}{1 + e^{-z}}$$

$$g(0) = 0.5$$

In perceptron algorithm, we use a step function:

$$\text{perceptron } g(z) = \begin{cases} 1, z \geq 0 \\ 0, z < 0 \end{cases}$$

Specifically, the perceptron hypothesis is

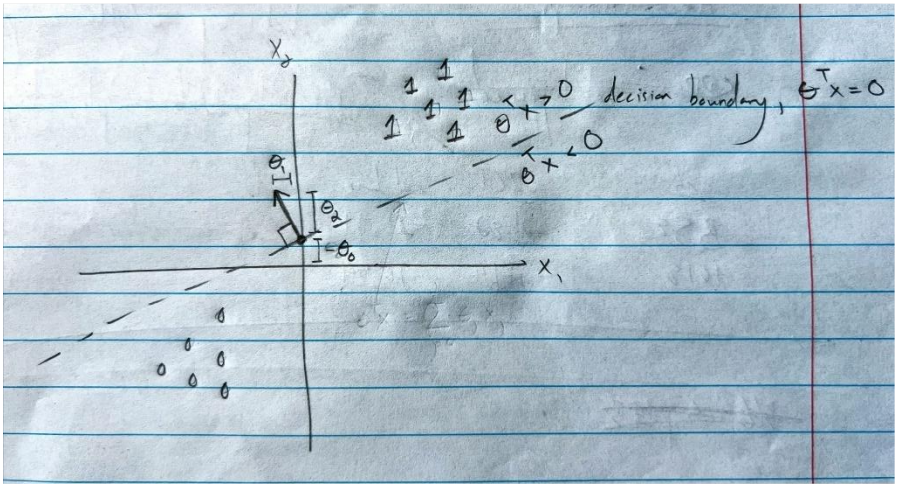
$$\text{perceptron } h_\theta(x) = g(\theta^T x) = \begin{cases} 1, \theta^T x \geq 0 \\ 0, \theta^T x < 0 \end{cases}$$

The perceptron update equation has the same form as logistic regression (stochastic gradient descent):

$$\theta_j \coloneqq \theta_j + \alpha \left(y^{(i)} - h_\theta\left(x^{(i)}\right) \right) x_j^{(i)}$$

$\left(y^{(i)} - h_\theta\left(x^{(i)}\right) \right)$ is the error between the sample and the hypothesis. For perceptron, this can take the values of 0 or ± 1. 0 means the algorithm got it right, and ± 1 means the algorithm got it wrong. θ only updates when the algorithm gets it wrong. From this we can see that logistic regression is a "softer" version of perceptron.

Graphically, the perceptron algorithm is



At the decision boundary, $\theta^T x = 0$. In this plot, we've drawn $\theta^T x > 0$ above the decision boundary and $\theta^T x < 0$ below the decision boundary, but this is arbitrary – these regions could be flipped. In the plot we've drawn above, this means our hypothesis is 1 above the boundary and 0 below the boundary, i.e. the square samples are 1 and the circle samples are 0.

There are two input features, i.e. $x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \end{bmatrix}$. For now, let's ignore the zeroth dimension and say $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$. The decision boundary is $\theta^T x$ which is also known as the dot product or correlation of θ and x. In other words,

$$\theta^T x = \theta \cdot x = \|\theta\| \|x\| \cos \phi$$

Where ϕ is the angle between θ and x. From this geometric definition, we can see that $\theta^T x = 0$ when θ and x are orthogonal (no correlation).

That's why the θ vector in the plot – consisting of $\begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$ – is normal to the decision boundary. It points up because $\theta^T x > 0$ above the decision boundary (if it pointed down, then $\theta^T x > 0$ below the boundary).

θ_0 simply shifts the decision boundary up and down to account for a DC offset. If we include θ_0, we have

$$\theta^T x = \sum_{j=0}^{2} \theta_j x_j = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

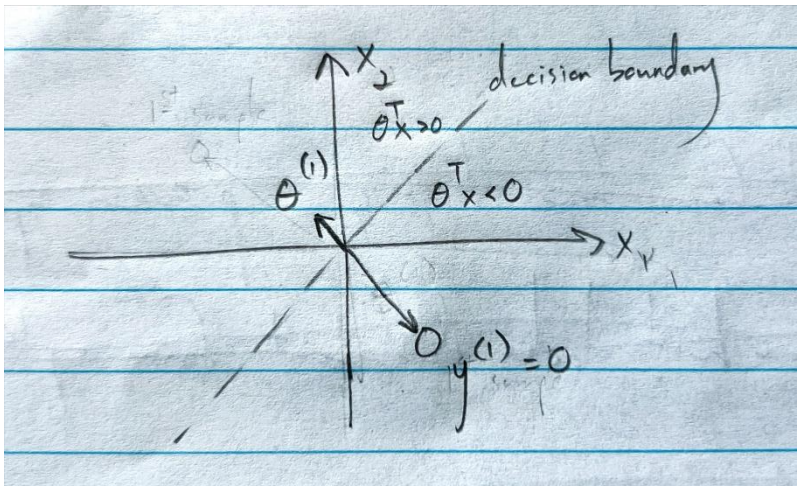
$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$$

$$\theta_1 x_1 + \theta_2 x_2 = -\theta_0$$

So the decision boundary is shifted by $-\theta_0$.

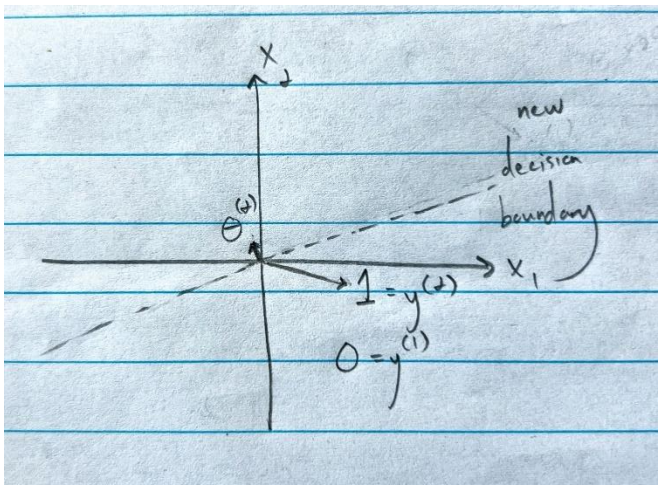
Let's go through some iterations to understand how the perceptron algorithm works with stochastic gradient descent (that is, we update the parameters based on one sample per iteration). Initialize the parameters as $\theta^{(0)} = \mathbf{0}$; then $\theta^{(0)} x^{(1)} = 0$, and $h_\theta\left(x^{(1)}\right) = 1$ is the first guess.

1. If $y^{(1)} = 1$, then we guessed correctly. Nothing happens, and $\theta^{(1)} = \mathbf{0}$.
2. If $y^{(1)} = 0$, then we guessed incorrectly, and $\theta^{(1)} = -\alpha x^{(1)}$, i.e. our first guess for θ points in the opposite direction of $x^{(1)}$ (see plot below).



θ wants to point towards $y^{(i)} = 1$ and point away from $y^{(i)} = 0$. That's what happens when there's an error. If $y^{(i)} = 1$ and $h_\theta\left(x^{(i)}\right) = 0$, then you add some of $x^{(i)}$ to θ. If $y^{(i)} = 0$ and $h_\theta\left(x^{(i)}\right) = 1$, then you add some of $-x^{(i)}$ to θ. Very simplistically, θ wants to be positively correlated with $y^{(i)} = 1$ and negatively correlated with $y^{(i)} = 0$.

For example, let's say $y^{(2)} = 1$, but ${\theta^{(1)}}^T x^{(2)} = 0$. Then you add some of $x^{(2)}$ to $\theta^{(1)}$, which causes θ and the decision boundary to rotate:



4.2 Exponential family

The exponential family is a probability distribution with pdf given by

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

1. y is the data (typically a scalar)
2. η is the natural parameter (of the distribution). η can be a vector; η and $T(y)$ have the same dimensions.
3. $T(y)$ is the sufficient statistic (for all the distributions we'll see in this class, $T(y) = y$)
4. $b(y)$ is the base measure (scalar)
5. $a(\eta)$ is the log partition function (scalar). You can think of $e^{a(\eta)}$ as a normalizing constant such that integrating the distribution yields 1. $\log e^{a(\eta)}$ yields $a(\eta)$, hence "log partition" function. The partition function is a technical term that indicates the normalizing constant of probability distributions.

As long as $\int_{-\infty}^{+\infty} P(y;\eta)dy = 1$, you have a distribution in the exponential family. Based on your choice of T, a, and b, you get different distributions (e.g. Gaussian). η may freely change without changing the type of distribution.

4.2.1 Bernoulli distribution example

Bernoulli distribution is used to model binary data. ϕ is the probability of the event ($y = 1$). ϕ is also known as the canonical parameter (vs. η, which is the natural parameter).

$$P(y;\phi) = \phi^y(1-\phi)^{1-y}$$

We used this for logistic regression.

A common technique for manipulating this expression is to take (natural) log and exp:

$$P(y;\phi) = \exp(\log P(y;\phi)) = \exp(\log(\phi^y(1-\phi)^{1-y})) = \exp\left(\log\left(\frac{\phi}{1-\phi}\right)y + \log(1-\phi)\right)$$

Pattern matching to exponential family expression:

$$P(y;\eta) = b(y)\exp\left(\eta^T T(y) - a(\eta)\right) = \exp\left(\log\left(\frac{\phi}{1-\phi}\right)y + \log(1-\phi)\right)$$

$$b(y) = 1$$

$$T(y) = y$$

$$\eta = \log\left(\frac{\phi}{1-\phi}\right) \rightarrow \phi = \frac{1}{1+e^{-\eta}}$$

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

In other words, we can rewrite the Bernoulli distribution as

$$P(y;\eta) = \exp\left(\eta y - a(\eta)\right)$$

$$\eta = \log\left(\frac{\phi}{1-\phi}\right)$$

This verifies that the Bernoulli distribution is a member of the exponential family.

4.2.2 Gaussian distribution example

Gaussian with fixed variance. Assume $\sigma^2 = 1$, so μ is the only canonical parameter.

$$P(y;\mu) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{(y-\mu)^2}{2}\right) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{y^2}{2}\right)\exp\left(\mu y - \frac{1}{2}\mu^2\right)$$

Pattern matching to exponential family expression:

$$P(y;\eta) = b(y)\exp\left(\eta^T T(y) - a(\eta)\right) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{y^2}{2}\right)\exp\left(\mu y - \frac{1}{2}\mu^2\right)$$

$$b(y) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{y^2}{2}\right)$$

$$T(y) = y$$

$$\eta = \mu$$

$$a(\eta) = \frac{1}{2}\mu^2 = \frac{1}{2}\eta^2$$

In other words, we can rewrite the Gaussian distribution as

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

$$b(y) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{y^2}{2}\right)$$

$$\eta = \mu$$

$$a(\eta) = \frac{1}{2}\eta^2$$

This verifies that the Gaussian distribution is a member of the exponential family.

If both mean and variance are canonical parameters, we have

$$\begin{aligned} P(y;\mu,\sigma^2) &= \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right) = \frac{1}{\sqrt{2\pi}}\frac{1}{\sigma}\exp\left(-\frac{y^2-2\mu y+\mu^2}{2\sigma^2}\right) \\ &= \frac{1}{\sqrt{2\pi}}\exp\left(\log\frac{1}{\sigma}\right)\exp\left(\frac{\mu}{\sigma^2}y - \frac{1}{2\sigma^2}y^2 - \frac{\mu^2}{2\sigma^2}\right) \\ &= \frac{1}{\sqrt{2\pi}}\exp\left(\frac{\mu}{\sigma^2}y - \frac{1}{2\sigma^2}y^2 - \frac{\mu^2}{2\sigma^2} - \log\sigma\right) \\ &= \frac{1}{\sqrt{2\pi}}\exp\left(\begin{bmatrix}\frac{\mu}{\sigma^2} & -\frac{1}{2\sigma^2}\end{bmatrix}\begin{bmatrix}y\\y^2\end{bmatrix} - \frac{\mu^2}{2\sigma^2} - \log\sigma\right) \end{aligned}$$

Pattern matching to exponential family expression:

$$P(y;\eta) = b(y)\exp\big(\eta^T T(y) - a(\eta)\big) = \frac{1}{\sqrt{2\pi}}\exp\left(\begin{bmatrix}\frac{\mu}{\sigma^2} & -\frac{1}{2\sigma^2}\end{bmatrix}\begin{bmatrix}y\\y^2\end{bmatrix} - \frac{\mu^2}{2\sigma^2} - \log\sigma\right)$$

$$b(y) = \frac{1}{\sqrt{2\pi}}$$

$$T(y) = \begin{bmatrix}y\\y^2\end{bmatrix}$$

$$\eta = \begin{bmatrix}\eta_1\\\eta_2\end{bmatrix} = \begin{bmatrix}\frac{\mu}{\sigma^2}\\-\frac{1}{2\sigma^2}\end{bmatrix}$$

$$\eta_2 = -\frac{1}{2\sigma^2} \to \sigma^2 = -\frac{1}{2\eta_2} = \left|-\frac{1}{2\eta_2}\right| = \left|\frac{1}{2\eta_2}\right| \to \sigma = \left|\frac{1}{2\eta_2}\right|^{\frac{1}{2}} \to \log\sigma = \frac{1}{2}\log\left|\frac{1}{2\eta_2}\right|$$

$$a(\eta) = \frac{\mu^2}{2\sigma^2} + \log \sigma = -\frac{\eta_1^2}{4\eta_2} + \frac{1}{2}\log\left|\frac{1}{2\eta_2}\right|$$

4.2.3 Properties of the exponential family

1. The MLE of the exponential family w.r.t. η is concave (has only one minimum at the global minimum). Equivalently, the NLL (negative log likelihood, i.e. take log of likelihood expression and multiply by -1) is convex.
2. $E(y;\eta) = \frac{\partial}{\partial \eta} a(\eta)$. The expectation, or mean, of the distribution is the partial derivative of the log partition function w.r.t. η.
3. $\mathrm{Var}(y;\eta) = \frac{\partial^2}{\partial \eta^2} a(\eta)$. The variance of the distribution is the second partial derivative of the log partition function w.r.t. η.

In general, calculating mean and variance of probability distributions requires integration, but the exponential family only requires differentiation, which is much simpler.

When η is a vector, you use gradient and Hessian instead:

$$E(y;\eta) = \nabla_\eta\big(a(\eta)\big)$$

$$\mathrm{Var}(y;\eta) = \nabla_\eta^2\big(a(\eta)\big)$$

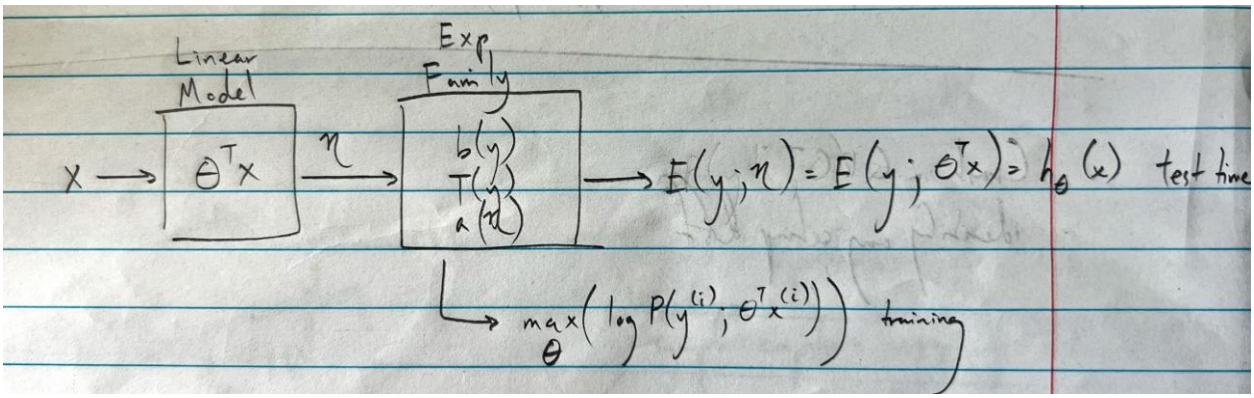
4.3 Generalized linear models (GLM)

A natural extension of the exponential family to include the input features in the model.

GLM assumptions/design choices:

1. Assumption: $(y|x;\theta) \sim \text{exponential family}(\eta)$. Assume that the distribution of y given x and parameterized by θ takes the form of an exponential family parameterized by η. Generally, we use Bernoulli distribution for binary data, Gaussian distribution for real data, Poisson distribution for discrete data (integers), Gamma or Exponential distribution for real positive data, Beta or Dirichlet distributions (they are ratios of distributions) for Distn (Bayesian machine learning or statistics). This applies to both regression and classification.
2. Design choice: $\eta = \theta^T x$ where $\theta, x \in \mathbb{R}^n$. We choose to make the natural parameter linear w.r.t. the input features.
3. Test time design choice: after training our model, given new input samples, the output of our model is $E(y|x;\theta)$. That is, the prediction generated by our model is $h_\theta(x) = E(y|x;\theta)$, or the mean of the distribution of y conditioned on x. If you plug in the Gaussian distribution or Bernoulli distribution, you'll see that the hypothesis function is equal to the one we used for linear regression and logistic regression.

Graphically, the GLM is



4.3.1 GLM training

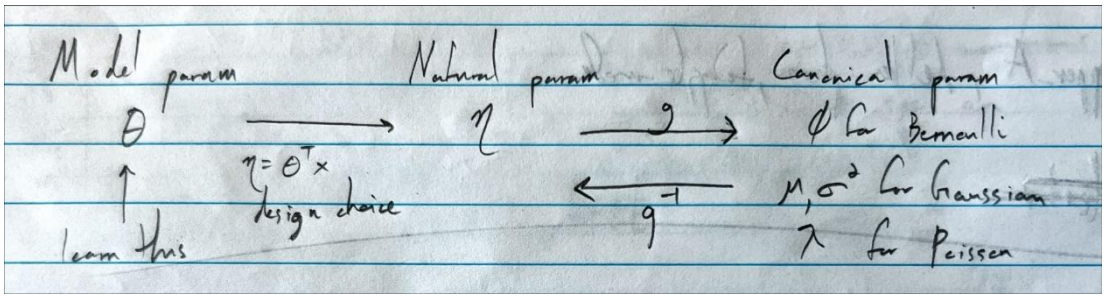
The learning update rule is the same for all GLMs (the only thing that changes is h_θ):

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

Terminology: η is the natural parameter, and $\mu = E(y;\eta) = g(\eta)$ is the canonical response function. The inverse of g is $\eta = g^{-1}(\mu)$ and is called the canonical link function.

Remember $g(\eta) = \frac{\partial}{\partial \eta} a(\eta)$.

We have 3 parameterizations for the exponential family/GLM:



I don't think g directly converts b/w natural and canonical params. It converts b/w η and the mean of the distribution calculated using the canonical params.

4.3.2 Bernoulli distribution

From before, we have

$$P(y;\phi) = \phi^y (1-\phi)^{1-y}$$

We showed that this falls under exponential families:

$$P(y;\eta) = \exp\left(\eta y - a(\eta)\right)$$

$$\eta = \log\left(\frac{\phi}{1-\phi}\right)$$

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

Extending to GLM, we choose $\eta = \theta^T x$ and $h_\theta(x) = E(y;\eta) = \frac{\partial a}{\partial \eta}$.

$$\frac{\partial a}{\partial \eta} = \frac{e^{\eta}}{1+e^{\eta}} = \frac{1}{1+e^{-\eta}} = \frac{1}{1+e^{-\theta^T x}} = g(\eta)$$

This is the exact expression we saw for logistic regression. The sigmoid function g is equal to the canonical response function.

From $P(y;\phi)$, we can also derive

$$E(y) = \phi$$

Then

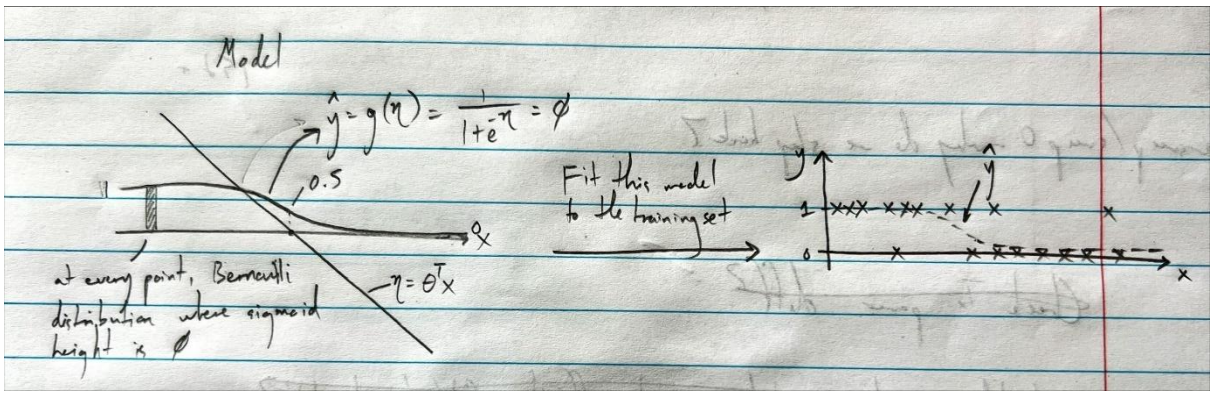
$$E(y) = \phi = g(\eta) = \frac{1}{1+e^{-\eta}}$$

Then

$$\eta = \log\left(\frac{\phi}{1-\phi}\right) = g^{-1}(\phi)$$

We already derived this expression when we went over exponential families.

Graphically,



4.3.3 Gaussian distribution

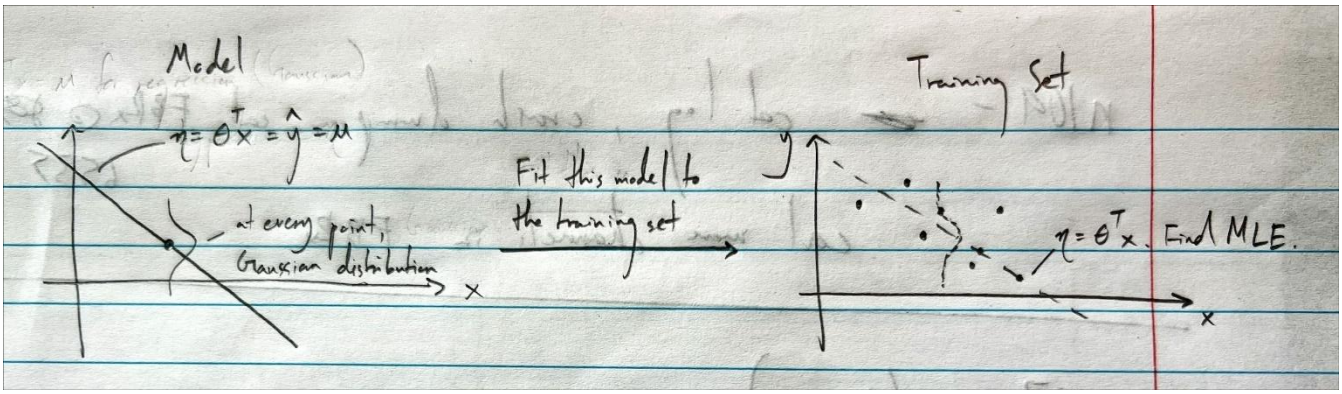
$$a(\eta) = \frac{1}{2}\eta^2$$

$$g(\eta) = \frac{\partial a}{\partial \eta} = \eta = \theta^T x = E(y)$$

We already derived $\eta = \mu$, so everything checks out.

$$\eta = \theta^T x = \mu$$

Graphically,



4.4 Softmax regression (aka cross-entropy minimization)

In the lecture notes, softmax regression is explained as a GLM, but in the delivered lecture, we'll take a non-GLM approach.

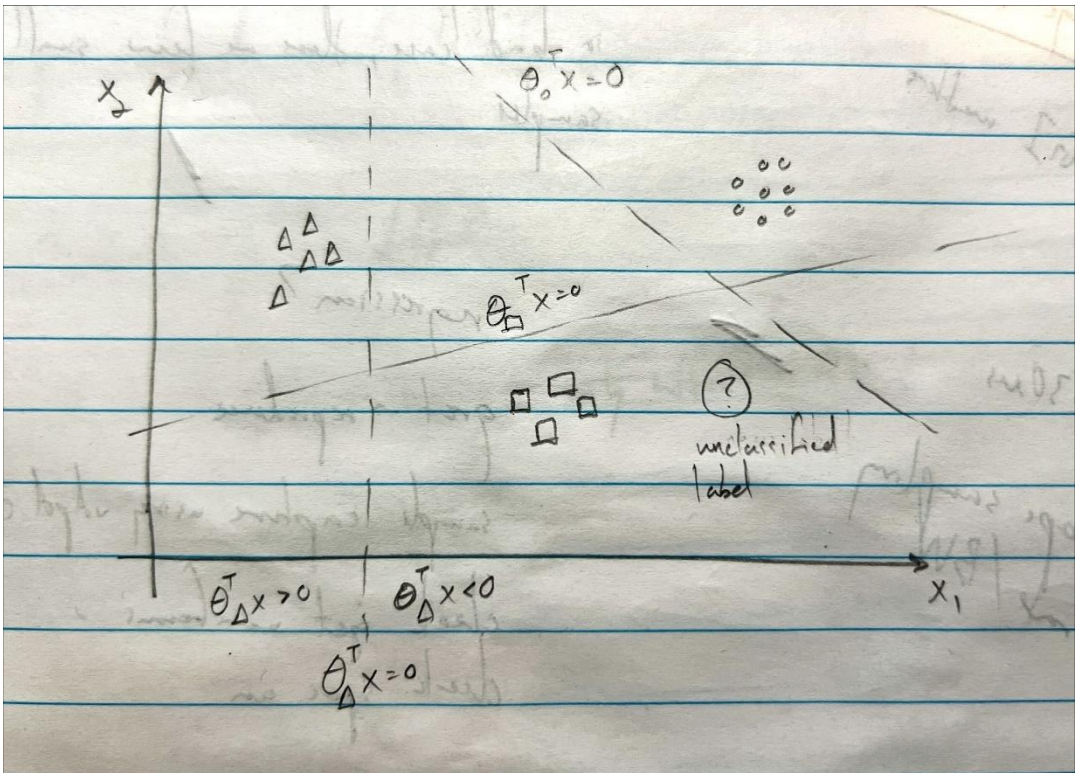
Softmax regression is used for multi-class classification. Let $x^{(i)} \in \mathbb{R}^n$, k be the number of classes, and the labels y are stored in a "one-hot" vector (or as integers of the form 2^i).

Each class has its own set of parameters, i.e. there is one θ per class. $\theta_{\text{class}} \in \mathbb{R}^n$, and there are k θ_{class}. We can store these θ_{class} into one matrix $\Theta^{\text{T}} \in \mathbb{R}^{k \times n}$:

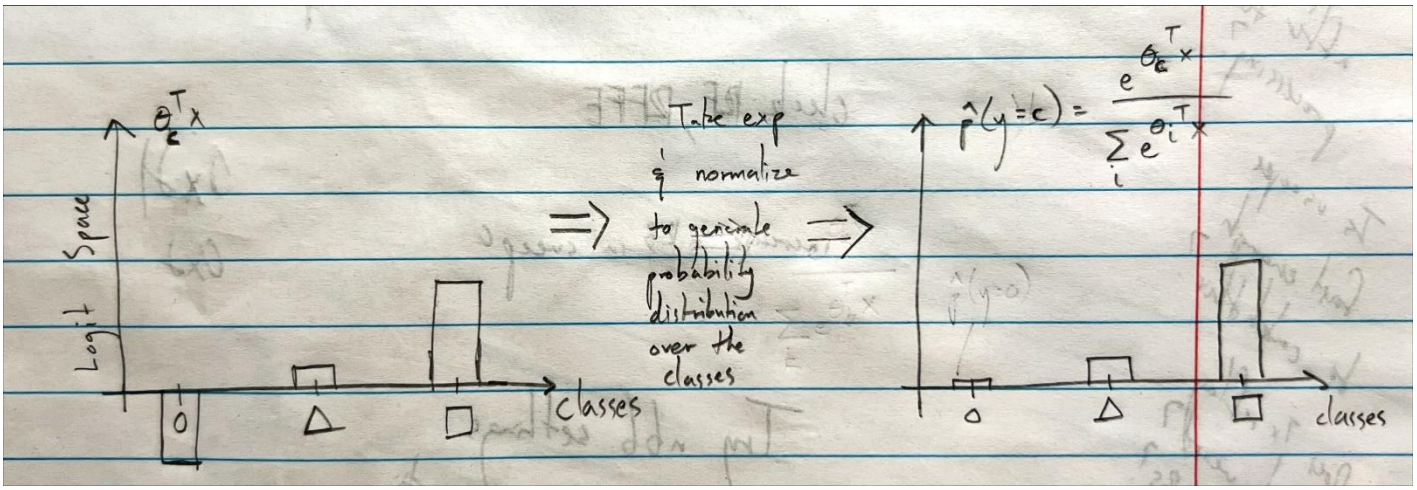
$$\Theta^{\text{T}} = \begin{bmatrix} \theta_{\text{class}1}^T \\ \theta_{\text{class}2}^T \\ \vdots \\ \theta_{\text{class}k}^T \end{bmatrix}$$

It's a generalization of logistic regression.

Let's take 3 classes as an example (triangles, squares, and circles) with two input features:



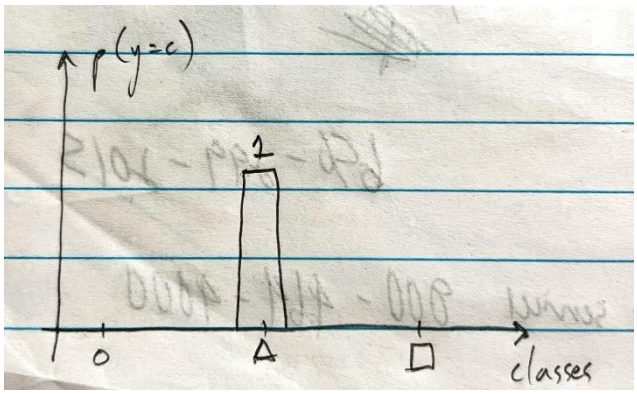
For the unclassified label, the $\theta_c^T x$ would evaluate to a logit space that looks something like this:



We take the exp of $\theta_c^T x$ and normalize to one to generate a probability distribution over the classes:

$$\hat{p}(y = c) = \frac{e^{\theta_c^T x}}{\sum_i e^{\theta_i^T x}}$$

This is our hypothesis probability distribution. Let's say the unclassified label is actually a triangle. Then we can represent the "real" "probability distribution" like this:



The goal of cross-entropy minimization is to minimize the difference between $\hat{p}$ and p:

$$\text{cross entropy}(p, \hat{p}) = -\sum_{y} p(y) \log \hat{p}(y)$$

In our example, this evaluates to

$$-\log \hat{p}(y = \text{triangle}) = -\log \frac{e^{\theta_{\text{triangle}}^T x}}{\sum_i e^{\theta_i^T x}}$$

This is the loss function; use gradient descent to update θ.

5 Lecture 5/6 – GDA and naïve Bayes

The algorithms up to this point have been discriminative learning algorithms. Today, we'll learn about generative learning algorithms, particularly Gaussian discriminant analysis (GDA). GDA is simpler than logistic regression and may be more computationally efficient. Sometimes it works better for very small datasets.

In logistic regression, you're trying to find the decision boundary between 0 and 1, separating the two classes. In generative learning algorithms, you look at each class individually and build one model per class. At test time, you evaluate each new sample against all models and find the one that matches most closely.

Formally,

1. Discriminative learning algorithm: learn $P(y|x) = P(y|x; \theta)$, or equivalently, learn $h_\theta(x)$. Based on the input features, pick the most likely y.

2. Generative learning algorithm: learn $P(x|y)$ – for each class, learn its features. Learn $P(y)$, also known as the class prior – learn the probability of each class regardless of input features.

Bayes rule: $P(y = 1, x) = P(y = 1|x)P(x) = P(x|y = 1)P(y = 1)$

Then

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

$$P(x) = P(x|y = 1)P(y = 1) + P(x|y = 0)P(y = 0)$$

5.1 GDA

5.1.1 Multivariate Gaussian distribution

Suppose $x \in \mathbb{R}^n$ – unlike discriminant learning algorithms, we're going to drop the $x_0 = 1$ convention.

Assume $P(x|y)$ is Gaussian.

Let's define a multivariate Gaussian random variable, $z \in \mathbb{R}^n$. Multivariate refers to the fact that z is a vector of univariate Gaussian random variables.

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

$$\mu \in \mathbb{R}^n, \Sigma \in \mathbb{R}^{n \times n}$$

$$E(z) = \mu$$

Σ is the covariance matrix. Σ is always symmetric.

$$\Sigma = \text{cov}(z) = E[(z - \mu)(z - \mu)^T] = E(zz^T) - E(z)E(z)^T = E(zz^T) - \mu\mu^T$$

Simpler notation, where we sub Ez for $E(z)$:

$$\Sigma = Ezz^T - (Ez)(Ez)^T$$

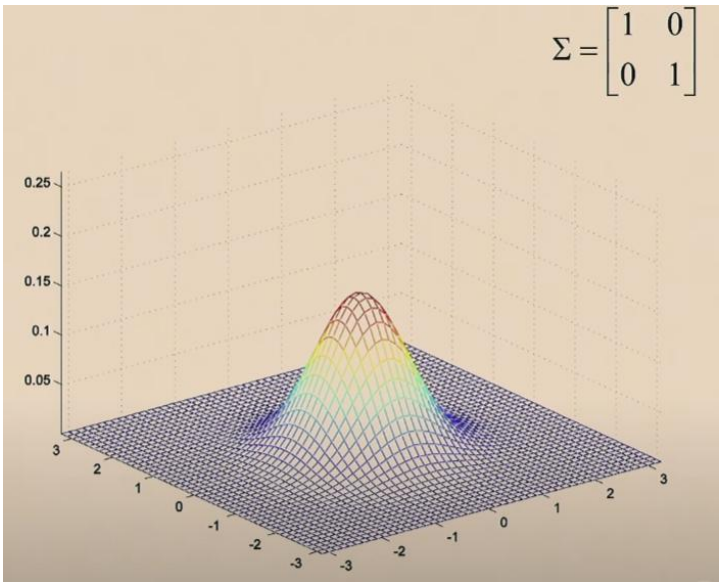
The diagonal elements of Σ are the variances of z_i, and the off-diagonal elements are the covariances of z_i, z_j. Positive values = positively correlated, negative values = negatively correlated, 0 = uncorrelated.

The probability distribution is

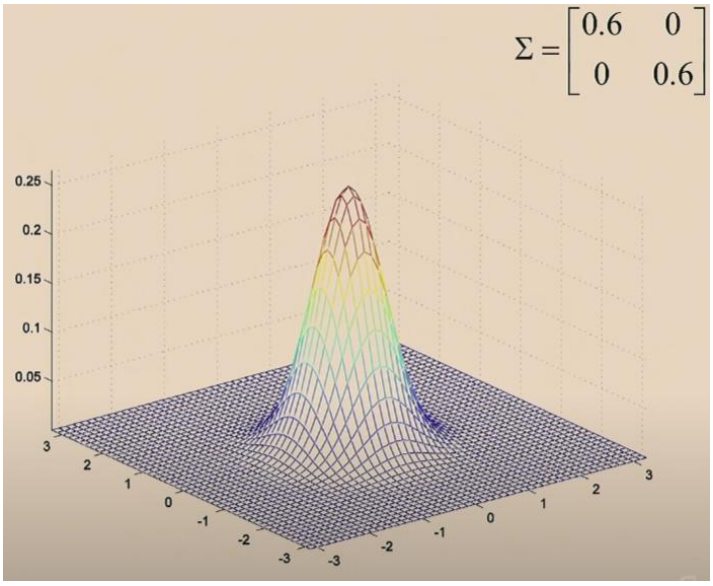
$$P(z) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(z - \mu)^T\Sigma^{-1}(z - \mu)\right)$$

5.1.1.1 Example plots for two-variate Gaussian

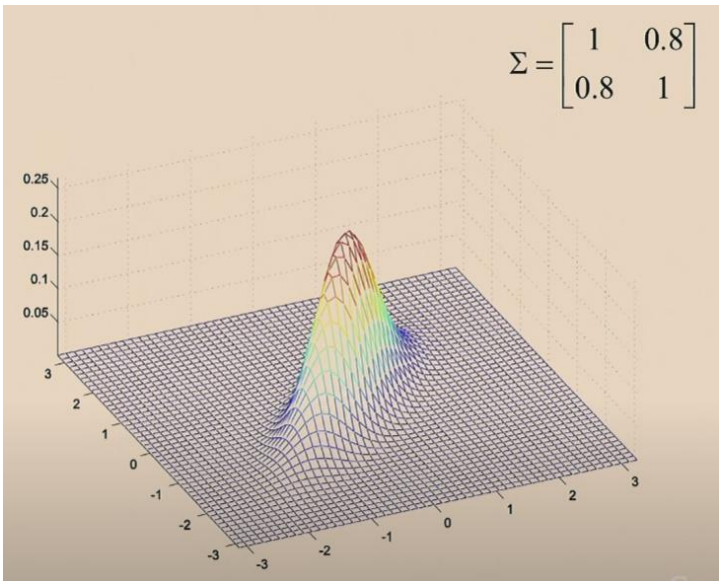
Radially symmetric “bump”:

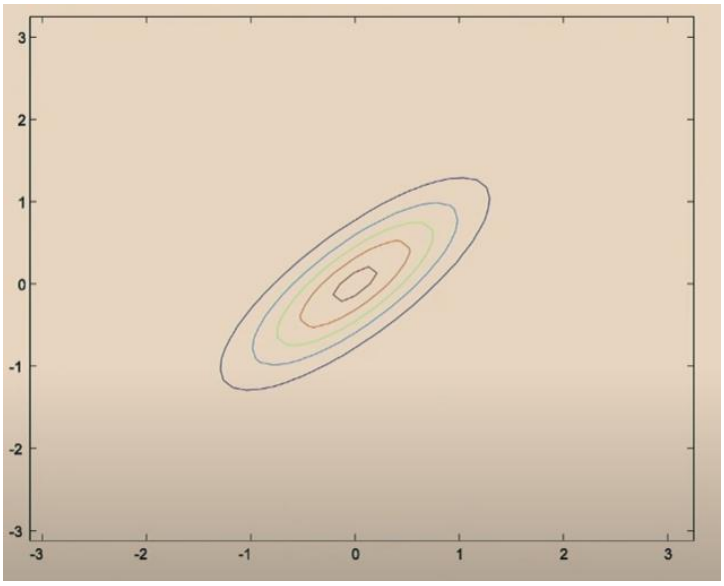


Radially symmetric “bump” with lower variance:



Nonzero covariance:





5.1.2 GDA training for binary data

The parameters of the GDA model are $\mu_0, \mu_1, \Sigma, \phi$. We use the same Σ for both classes in order to obtain a linear decision boundary (you can define Σ_0 and Σ_1, but that's not usually done because it adds an additional parameter – however, Ng says this is also a reasonable approach). (Remember that x, μ_0, μ_1 are vectors in general). Then the multivariate Gaussian distributions are

$$P(x|y=0) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right)$$

$$P(x|y=1) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right)$$

$$P(y) = \phi^y (1-\phi)^{1-y}$$

$$x, \mu_0, \mu_1 \in \mathbb{R}^n$$

$$\Sigma \in \mathbb{R}^{n \times n}$$

$$\phi \in \mathbb{R}$$

Graphically, we're trying to fit the contour of the multivariate Gaussian to each of our classes.

If you know all the parameters, then at test time, through Bayes rule, you can calculate $P(y|x)$.

Let's say you have a training set $\left(x^{(i)}, y^{(i)}\right), i = 1 \ldots m$. To fit the parameters, we want to maximize the joint likelihood:

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

Let's drop the parameters to simplify the notation. Then

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

Recall that likelihood in discriminative learning algorithms (GLMs) is defined as

$$\mathcal{L}(\theta) = \prod_{i=1}^{m} P\left(y^{(i)} \middle| x^{(i)}; \theta\right)$$

Since it's conditioned on the input parameters (y given x), it's also known as conditional likelihood. In contrast, in generative learning algorithms, we try to maximize the probability of x and y, not y given x.

Maximum likelihood estimation:

$$\ell(\phi, \mu_0, \mu_1, \Sigma) = \log \mathcal{L}(\phi, \mu_0, \mu_1, \Sigma)$$

$$\max_{\phi, \mu_0, \mu_1, \Sigma} \ell(\phi, \mu_0, \mu_1, \Sigma)$$

Taking the partial derivatives of ℓ w.r.t. the parameters and setting to 0 yields

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

We can rewrite this using the indicator function ($\mathcal{I}\{\text{true}\} = 1, \mathcal{I}\{\text{false}\} = 0$):

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

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

$$\mu_1 = \frac{\sum_{i=1}^{m} \mathcal{I}\{y^{(i)} = 1\} x^{(i)}}{\sum_{i=1}^{m} \mathcal{I}\{y^{(i)} = 1\}}$$

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} \left(x^{(i)} - \mu_{y^{(i)}}\right) \left(x^{(i)} - \mu_{y^{(i)}}\right)^T$$

5.1.3 GDA testing/prediction

By Bayes rule,

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

In the equation above, we're trying to find the maximum probability. We can use arg max instead of max, which explicitly says we're trying to find the input argument (in this case, y), that maximizes the function (in this case $P(y|x)$):

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

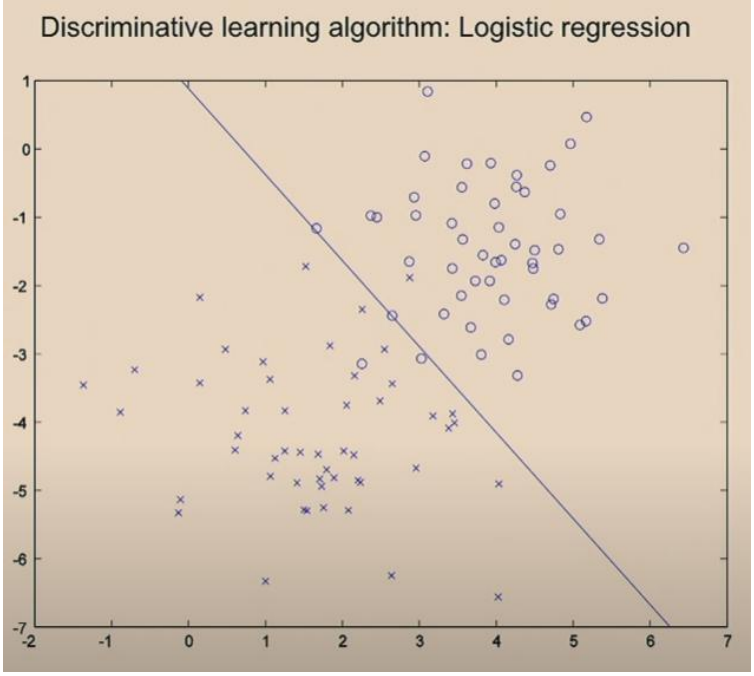
$P(x)$ is independent of y, so we can simplify to

$$\arg\max_y P(y|x) = \arg\max_y P(x|y)P(y)$$

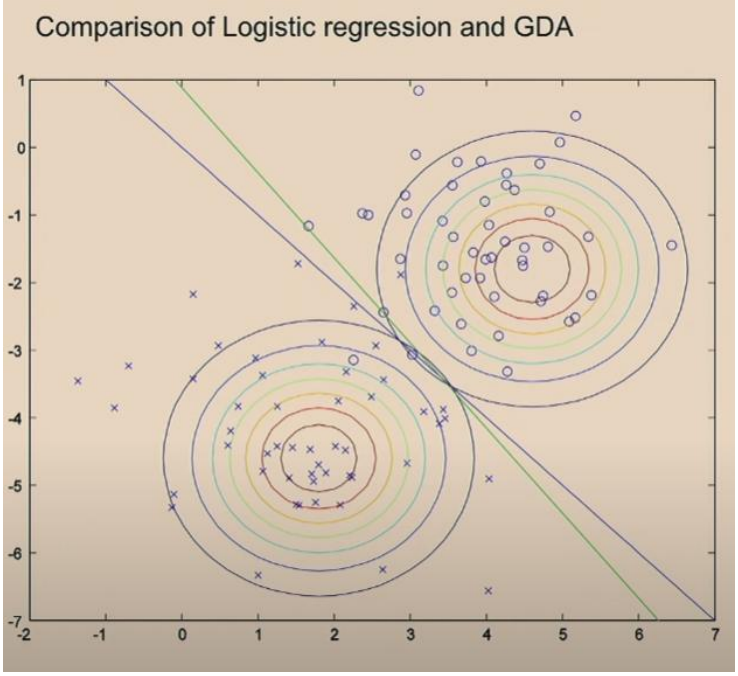
Of course, you'll still need $P(x)$ if you want to calculate the maximum probability and not just the arg max.

5.1.4 Graphical comparison of logistic regression and GDA

Logistic regression is an example of discriminative learning algorithm. We iterate on the line ($\theta^T x$) until we reach the maximum conditional likelihood, e.g.



GDA is an example of generative learning algorithm. We fit Gaussian curves to each set of classes (technically, we don't really look at each class separately since they share Σ), and then for each point, we predict the label using Bayes rule, e.g.



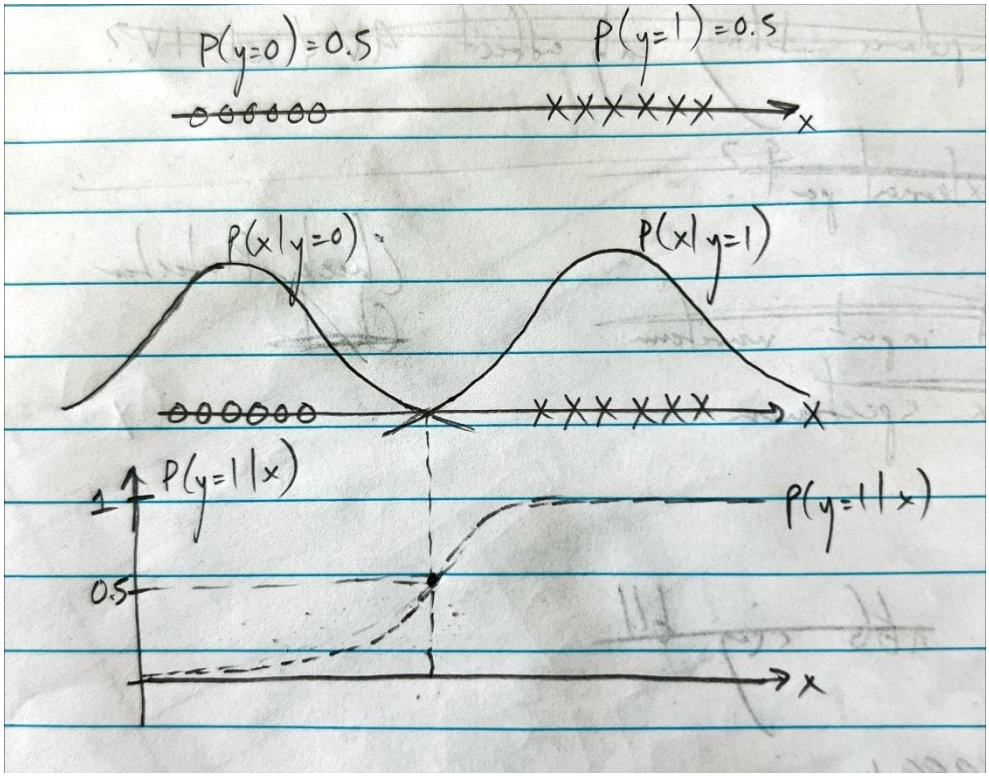
Blue line is from GDA, green line is from logistic regression.

5.1.5 GDA vs. logistic regression

Compare GDA to logistic regression. For a fixed set of parameters $\phi, \mu_0, \mu_1, \Sigma$, let's plot $P(y = 1|x;\ \phi, \mu_0, \mu_1, \Sigma)$ as a function of x.

$$P(y = 1|x;\ \phi, \mu_0, \mu_1, \Sigma) = \frac{P(x|y = 1; \mu_1, \Sigma)P(y = 1; \phi)}{P(x; \phi, \mu_0, \mu_1, \Sigma)}$$

$P(x|y = 1)$ is parameterized only by μ_1 and Σ. $P(y)$ is parameterized only by ϕ and in this example, $P(y = 1; \phi) = \phi$. $P(x) = P(x, y = 1) + P(x, y = 0) = P(x|y = 1)P(y = 1; \phi) + P(x|y = 0)P(y = 0; \phi)$, so it's parameterized by all parameters.



The points to the left almost certainly came from the $P(x|y = 0)$ distribution. The points to the right almost certainly came from the $P(x|y = 1)$ distribution. The points in the middle are in between the two distributions; the exact middle could have come from either with equal probability. It turns out that this curve is the sigmoid function.

Both logistic regression and GDA end up with $P(y = 1|x)$ as a sigmoid function, but the parameters are different, which leads to different decision boundaries.

GDA assumptions:

1. $(x|y = 0) \sim N(\mu_0, \Sigma)$
2. $(x|y = 1) \sim N(\mu_1, \Sigma)$

3. $y \sim \text{Ber}(\phi)$

Logistic regression assumption:

1. $P(y = 1|x) = \frac{1}{1+e^{-\theta^T x}}$ where $x_0 = 1$

As we saw above, the GDA assumptions lead $P(y = 1|x)$ being logistic, but the reverse is not true. That is, if you assume $P(y = 1|x)$ is logistic, the GDA assumptions aren't necessarily true.

This means GDA makes stronger (more specific) assumptions than logistic regression.

When you make stronger (and correct) assumptions, then your model will be better because it has more information. Of course, if those assumptions are wrong, then your model will be worse.

These assumptions also lead to $P(y = 1|x)$ being logistic:

1. $(x|y = 0) \sim \text{Poisson}(\lambda_0)$
2. $(x|y = 1) \sim \text{Poisson}(\lambda_1)$
3. $y \sim \text{Ber}(\phi)$

This is actually true for any GLM (exponential family) where the only difference between the two distributions is the natural parameter of the exponential family distribution.

This means that if you don't know whether your distribution is Poisson or Gaussian or some other exponential family, logistic regression will work fine either way.

Making weaker assumptions leads to a more robust model, but for small datasets, making stronger (and correct) assumptions will lead to better results. Larger datasets naturally give more information, so you can get away with making weaker assumptions. This is a general principle/tradeoff of machine learning.

Algorithm has two sources of information: the model assumptions (information that you give), and the data. When you have less data, your skill making the model assumptions is much more important.

However, one advantage of GDA over logistic regression is computational efficiency. It's easier to calculate mean and covariance than go through logistic regression iterations.

Softmax generalizes to multiple classes.

5.2 Naïve Bayes

This is another generative learning algorithm, but while GDA is for continuous, real-valued input features, naïve Bayes is for discrete-valued input features. Motivating example: email spam filtering – classify an email as spam or not spam (natural language processing).

Feature vector x? Start with an English dictionary and make a list of all the words in the dictionary, e.g.

-a

-aardvark

-aardwolf

…

-buy

…

-zygmurgy

In practice, you look at the top 10000 recurring words in your emails. When you encounter a word not in your dictionary, you can either ignore it or map it to "UNK" for unknown word.

Create a feature vector x of length 10000, each element is binary. For a given email, 1 indicates that the word is present, 0 indicates the word is absent. We can express this as

$$x \in \{0,1\}^n$$

$$x_i = \mathcal{I}\{\text{word } i \text{ appears in email}\}$$

$$n = 10000$$

Since this is generative learning algorithm, we want to model $P(x|y)$ and $P(y)$.

There are 2^{10000} possible values of x, and if you try to model x as a multinomial distribution, then you need 2^{10000} parameters (each parameter is the event probability of a specific x). Technically $2^{10000} - 1$ parameters since the probabilities add up to 1. This is too many parameters – you need some assumptions.

Start with an expression for the probability of x:

$$P(x_1, \dots, x_{10000}|y)$$

By the chain rule of probability, we have

$$P(x_1, \dots, x_{10000}|y) = P(x_1|y)P(x_2|x_1, y)P(x_3|x_1, x_2, y) \dots P(x_{10000}| \dots)$$

Assume x_i are conditionally independent given y, then we have

$$P(x_1, \dots, x_{10000}|y) = P(x_1|y)P(x_2|y)P(x_3|y) \dots P(x_{10000}|y) = \prod_{i=1}^{n} P(x_i|y)$$

This is called the conditional independence assumption and sometimes the naïve Bayes assumption. You're assuming that if you know y, then the probability of any word appearing in the email is independent of all other probabilities. This is definitely not true, but it may be true enough to get away with.

Then the parameters of this model are

$$\phi_{j|y=1} = P(x_j = 1|y = 1)$$

$$\phi_{j|y=0} = P(x_j = 1|y = 0)$$

$$\phi_y = P(y = 1)$$

The joint likelihood is

$$\begin{aligned}
\mathcal{L}(\phi_y, \phi_{j|y}) &= \prod_{i=1}^{m} P(x^{(i)}, y^{(i)}; \phi_y, \phi_{j|y}) \\
&= \prod_{i=1}^{m} \left[P(x^{(i)}|y^{(i)} = 1)P(y^{(i)} = 1)\right]^{y^{(i)}} \left[P(x^{(i)}|y^{(i)} = 0)P(y^{(i)} = 0)\right]^{1-y^{(i)}} \\
&= \prod_{i=1}^{m} \left[\phi_y \prod_{j=1}^{n} \phi_{j|y=1}\right]^{y^{(i)}} \left[(1 - \phi_y) \prod_{j=1}^{n} \phi_{j|y=0}\right]^{1-y^{(i)}}
\end{aligned}$$

The MLE for the parameters are

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

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

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

Once you estimate the parameters, you can calculate the posterior probability just like in GDA. By Bayes' rule,

$$P(y=1|x) = \frac{P(x|y=1)P(y=1)}{P(x|y=1)P(y=1) + P(x|y=0)P(y=0)}$$

5.2.1 When to use naïve Bayes?

Ng: This algorithm nearly works. With one fix (next lecture), this is a not-too-horrible spam classifier. Logistic regression is almost always better, but naïve Bayes is computationally efficient – estimating the parameters is simply counting, and calculating probabilities is simply multiplying the individual probabilities. It's also easy to update the model as more emails come in.

5.2.2 The problem

What happens when a never-before-seen word pops up in a new email? The probabilities predicted by the model are zero. Let's say the new word is "NIPS" (an annual AI conference) and $j = 6017$.

$$\phi_{j=6017|y=1} = \frac{\sum_{i=1}^{m} \mathcal{I}\left\{x_j^{(i)} = 1, y^{(i)} = 1\right\}}{\sum_{i=1}^{m} \mathcal{I}\{y^{(i)} = 1\}} = \frac{0}{\sum_{i=1}^{m} \mathcal{I}\{y^{(i)} = 1\}} = 0$$

$$\phi_{j=6017|y=0} = \frac{0}{\sum_{i=1}^{m} \mathcal{I}\{y^{(i)} = 0\}} = 0$$

Then $P(x|y=1)$ and $P(x|y=0)$ evaluate to zero:

$$P(x|y=1) = \prod_{j=1}^{10000} \phi_{j|y=1} = 0$$

$$P(x|y=0) = \prod_{j=1}^{10000} \phi_{j|y=0} = 0$$

Then

$$P(y=1|x) = \frac{P(x|y=1)P(y=1)}{P(x|y=1)P(y=1) + P(x|y=0)P(y=0)} = \frac{0}{0+0}$$

Apart from the divide-by-zero error, statistically it's a bad idea to estimate the probability of an event as zero just because it hasn't happened yet.

Laplace smoothing addresses this problem.

5.2.3 Laplace smoothing

Example: football team record

9/12 vs. Wake Forest. Won? 0.

10/10 vs. OSU. Won? 0.

10/17 vs. Arizona. Won? 0.

11/21 vs. Caltech. Won? 0.

12/31 vs. Oklahoma. Predict result?

If we use MLE, we'd predict

$$P(x=1) = \frac{\#\text{ of 1s}}{\#\text{ of 1s} + \#\text{ of 0s}} = \frac{0}{0+4} = 0$$

In Laplace smoothing, we add one to each of the numbers:

$$P(x=1) = \frac{(\#\text{ of 1s}) + 1}{(\#\text{ of 1s}) + 1 + (\#\text{ of 0s}) + 1} = \frac{1}{6}$$

This is an optimal estimate under certain assumptions.

More generally, when x can take on k different values,

$$x \in \{1, \dots, k\}$$

$$\text{MLE} = P(x=j) = \frac{\sum_{j=1}^{m} \mathcal{I}\{x^{(i)} = j\}}{m}$$

With Laplace smoothing, we have

$$P(x=j) = \frac{1 + \sum_{j=1}^{m} \mathcal{I}\{x^{(i)} = j\}}{m+k}$$

In the case of naïve Bayes, we update the MLE parameters to be

$$\phi_{j|y=1} = \frac{1 + \sum_{i=1}^{m} \mathcal{I}\{x_j^{(i)} = 1, y^{(i)} = 1\}}{2 + \sum_{i=1}^{m} \mathcal{I}\{y^{(i)} = 1\}}$$

$$\phi_{j|y=0} = \frac{1 + \sum_{i=1}^{m} \mathcal{I}\{x_j^{(i)} = 1, y^{(i)} = 0\}}{2 + \sum_{i=1}^{m} \mathcal{I}\{y^{(i)} = 0\}}$$

5.2.4 Input features with categorical distributions; discretizing input features

Up to this point, we've assumed that x_j are binary-valued, but what if x_j can take values in $\{1,2,\dots,k\}$?

Then we model x_j as a categorical distribution rather than Bernoulli – instead of ϕ, we have $p_1,\dots,p_k$ as the event probabilities. Everything else is the same.

This might come up in a scenario where you want to discretize input features that are continuous-valued. For example, let's say our input feature is the square footage of a house. Then we might discretize x_j as

1. $x_j = 1$ – square footage < 400
2. $x_j = 2$ – square footage between 400 and 800
3. $x_j = 3$ – square footage between 800 and 1200
4. $x_j = 4$ – square footage between 1200 and 1600
5. $x_j = 5$ – square footage > 1600

This describes the possible values of a single input feature, the square footage. When the original, continuous-valued features are not well-modeled by a multivariate normal distribution, discretizing the features and using naïve Bayes instead of GDA often results in a better classifier.

5.2.5 Variation on naïve Bayes for text classification

Let x be your input vector of words. Let's say x_1 corresponds to a, x_2 to aardvark, x_{800} to buy, x_{1600} to drugs, x_{6200} to now.

Let's say a new email comes in that says "Drugs! Buy drugs now!"

Standard naïve Bayes (aka multivariate Bernoulli event model) does not use the information that "drugs" appears twice – $x_j \in \{0,1\}$, that is, the input vector only indicates whether "drugs" appears and not the count.

Emails can vary greatly in length, but they all map to a feature vector of the same length.

Better representation for text (aka multinomial event model), using our example:

$$x = \begin{bmatrix} 1600 \\ 800 \\ 1600 \\ 6200 \end{bmatrix} \in \mathbb{R}^{n_i}$$

$$x_j \in \{1,\dots,10000\}$$

x_j, the elements of the feature vector, now indicate the dictionary index instead of True/False.

n_i is the length of email i (the i^{th} example).

Using the naïve Bayes assumption, where we assume the input features are conditionally independent on y, we have

$$P(x,y) = P(x|y)P(y) \overset{\text{assume}}{=} P(y)\prod_{j=1}^{n} P(x_j|y)$$

Note that this equation looks exactly the same as the one derived for multivariate Bernoulli event model. It's the x_j and n that are different.

Parameters:

$$\phi_y = P(y=1)$$

$$\phi_{k|y=0} = P(x_j = k|y=0)$$

$$\phi_{k|y=1} = P(x_j = k|y=1)$$

This reads as the chance of word j being k if $y=0$ or $y=1$. We've assumed that the probability of k is independent of position j – that is, the probability of k appearing in an email is independent of where it appears in the email. In other words, ϕ_k indicates the probability of any word being k.

MLE:

$$\phi_{k|y=0} = \frac{\sum_{i=1}^{m}\left(\mathcal{I}\{y^{(i)}=0\} * \sum_{j=1}^{n_i}\mathcal{I}\left\{x_j^{(i)}=k\right\}\right)}{\sum_{i=1}^{m}(\mathcal{I}\{y^{(i)}=0\} * n_i)}$$

$$\phi_{k|y=1} = \frac{\sum_{i=1}^{m}\left(\mathcal{I}\{y^{(i)}=1\} * \sum_{j=1}^{n_i}\mathcal{I}\left\{x_j^{(i)}=k\right\}\right)}{\sum_{i=1}^{m}(\mathcal{I}\{y^{(i)}=1\} * n_i)}$$

These MLE read as the number of instances of k divided by the total number of words.

With Laplace smoothing, if there are 10000 possible values for x_j (10000 words in the dictionary), we have

$$\phi_{k|y=0} = \frac{1+\sum_{i=1}^{m}\left(\mathcal{I}\{y^{(i)}=0\} * \sum_{j=1}^{n_i}\mathcal{I}\left\{x_j^{(i)}=k\right\}\right)}{10000+\sum_{i=1}^{m}(\mathcal{I}\{y^{(i)}=0\} * n_i)}$$

$$\phi_{k|y=1} = \frac{1 + \sum_{i=1}^{m} \left(\mathcal{I}\{y^{(i)} = 1\} * \sum_{j=1}^{n_i} \mathcal{I}\left\{x_j^{(i)} = k\right\} \right)}{10000 + \sum_{i=1}^{m} (\mathcal{I}\{y^{(i)} = 1\} * n_i)}$$

5.3 GDA/naïve Bayes tradeoffs

GDA/naïve Bayes aren't the best classifiers. Logreg, SVMs, neural networks are all better.

However, GDA/naïve Bayes are simple to implement (no iteration). Very good to start with.

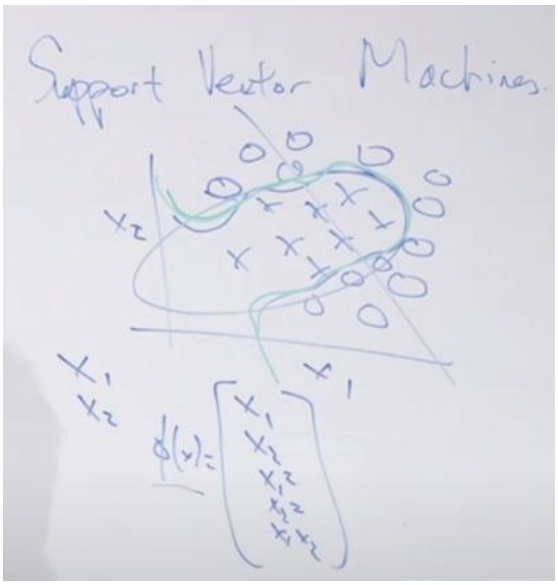
Good for initial prototyping. Usually prefer to start with simple algorithms vs. complicated ones (analogous to writing 10000 lines of code and then compiling for the first time). When you're starting off on a new application problem, it's hard to tell which part of the problem will be the most difficult.

Example: let's say you're designing a spam filter. Spammers have learned to disguise spam by deliberately misspelling words. E.g. instead of mortgage, m0rtg/\ge. Spammers may use a spoofed header. When you're first designing, you'll probably have little idea which problems are the most critical or prevalent, so you'll want to design a simple algorithm so you can figure out which problems you need to solve.

6 Lectures 6/7 – support vector machines

SVMs are classifiers for nonlinear decision boundaries. Logistic regression can also be used to generate nonlinear decision boundaries by increasing the order of the input vector, e.g. instead of just $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, try $\begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \\ x_1 x_2 \end{bmatrix}$ for example. However, feature selection is very difficult.

SVMs is a relatively "turnkey" algorithm, which means it doesn't have many parameters to fiddle with. For example (and in contrast), logreg requires tuning of the learning rate. There are many mature software packages for SVMs that you can download and use very easily without requiring much tuning (but they aren't as good as neural networks).



The optimal margin classifier is the basic building block of SVM.

We'll start with datasets that are linearly separable (can separate with linear decision boundary) to build an algorithm that is similar to logreg.

Then we'll introduce kernels, which map the original input features to a higher order. In our example, map from

$$x \in \mathbb{R}^2 \rightarrow \phi(x) \in \mathbb{R}^5$$

$\phi(x)$ can be $\mathbb{R}^{100000}$ or even $\mathbb{R}^{\infty}$, thus saving us from having to manually choose the nonlinear input features.

Then we'll go over the inseparable case.

6.1 Optimal margin classifier (linearly separable case; baby SVM)

Functional margin definition: how confidently and accurately you classify an example.

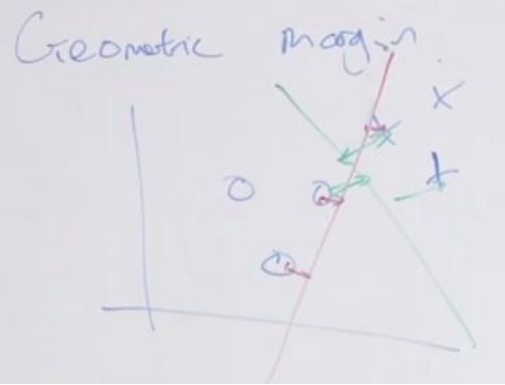
E.g. for logreg, $h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$. Predict 1 if $\theta^T x \geq 0$ (equivalent to $h_\theta(x) = P(y = 1|x) \geq 0.5$) and 0 otherwise.

If these two statements are true, then the classifier has a large functional margin:

1. If $y^{(i)} = 1$, we desire that $\theta^T x^{(i)} \gg 0$ so that $h_\theta(x) \approx 1$, and we have a very correct and confident prediction.
2. If $y^{(i)} = 0$, we desire that $\theta^T x^{(i)} \ll 0$.

Geometric margin quantifies the physical separation of the decision boundary from the training examples. For example, in the linearly separable dataset below, both the red and

green decision boundaries separate the classes. However, the green line has higher geometric margin.



The rudimentary SVM – SVM of low-dimensional space, aka optimal margin classifier – maximizes the geometric margin.

6.1.1 Notation

Labels $y \in \{-1, +1\}$

h outputs values in $\{-1, +1\}$

$$g(z) = \begin{cases} 1 \text{ if } z \geq 0 \\ -1 \text{ otherwise} \end{cases}$$

Previously in logreg, $h_\theta(x) = g(\theta^T x), x \in \mathbb{R}^{n+1}, x_0 = 1$. In SVM, we have

$$h_{w,b}(x) = g(w^T x + b)$$

$$x \in \mathbb{R}^n, b \in \mathbb{R}$$

We've dropped the $x_0 = 1$ convention and instead replaced $\theta_0 x_0$ with b.

$$w^T x + b = \left(\sum_{j=1}^{n} w_j x_j \right) + b$$

The parameters (w, b) define a linear classifier (a line, plane, or hyperplane in higher dimensions).

6.1.2 Functional margin definition

The functional margin of the hyperplane defined by (w, b) w.r.t. one training example $\left(x^{(i)}, y^{(i)}\right)$ is

$$\hat{\gamma}^{(i)} = y^{(i)}\left(w^T x^{(i)} + b\right)$$

If $y^{(i)} = 1$, we want $w^T x^{(i)} + b \gg 0$. If $y^{(i)} = -1$, we want $w^T x^{(i)} + b \ll 0$.

This means that in all cases, we want $\hat{\gamma}^{(i)} \gg 0$.

If $\hat{\gamma}^{(i)} > 0$, then $h\left(x^{(i)}\right) = y^{(i)}$ (correct prediction).

The function margin w.r.t. the entire training set is

$$\hat{\gamma} = \min_i \hat{\gamma}^{(i)}$$

$$i = 1, \ldots, m$$

That is, the functional margin is the worst case. This is pretty brittle, and for now we're assuming that the dataset is linearly separable.

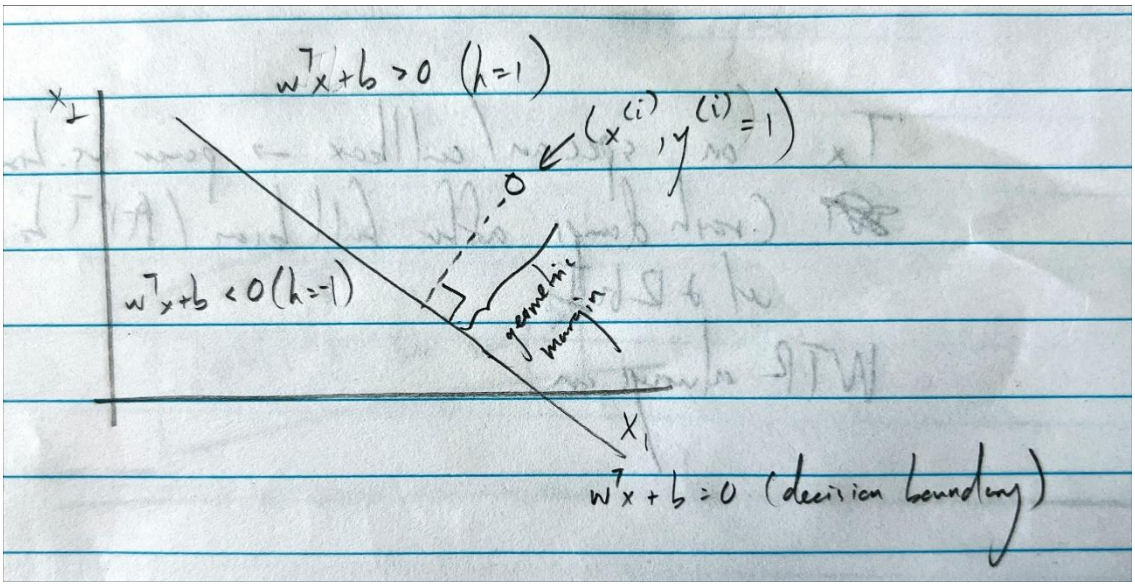
You can easily "cheat" to increase your functional margin. For example, you could multiply all your parameters by 2, i.e. $w \to 2w$ and $b \to 2b$, which doubles your functional margin. However, this doesn't actually change your decision boundary.

To standardize the functional margin, you can scale your parameters by $\|w\|$:

$$(w, b) \to \left(\frac{w}{\|w\|}, \frac{b}{\|w\|}\right)$$

6.1.3 Geometric margin definition

Let's define the geometric margin w.r.t. a single example:



The classifier is classifying the example correctly since $y^{(i)} = h_{w,b}\left(x^{(i)}\right) = 1$. The geometric margin is the Euclidean distance of the example from the decision boundary.

The geometric margin of the hyperplane defined by (w, b) w.r.t. $\left(x^{(i)}, y^{(i)}\right)$ is

$$\gamma^{(i)} = \frac{y^{(i)}\left(w^T x^{(i)} + b\right)}{\|w\|}$$

This is proved in the lecture notes.

Then the functional and geometric margins are related by

$$\gamma^{(i)} = \frac{\hat{\gamma}^{(i)}}{\|w\|}$$

That is, the geometric margin is the functional margin divided by the length of w.

Similarly, the geometric margin of the training set is the worst-case training example:

$$\gamma = \min_i \gamma^{(i)}$$

The optimal margin classifier chooses (w, b) to maximize γ.

One way to pose this problem is

$$\max_{\gamma,w,b} \gamma \text{ s.t. } \frac{y^{(i)}\left(w^T x^{(i)} + b\right)}{\|w\|} \geq \gamma, \forall\, i$$

That is, maximize the worst-case $\gamma^{(i)}$ over all the examples. This is not a convex optimization problem, so it's difficult to solve by gradient descent.

Recall that you can scale (w, b) freely without changing the decision boundary. Let's choose to scale w s.t. $\|w\| = \frac{1}{\gamma}$. Then the problem above becomes

$$\max_{w,b} \frac{1}{\|w\|} \text{ s.t. } y^{(i)}\left(w^T x^{(i)} + b\right)\gamma \geq \gamma, \forall\, i$$

Which is equivalent to

$$\min_{w,b} \frac{1}{2}\|w\|^2 \text{ s.t. } y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1, \forall\, i$$

This is a convex optimization problem. There are very good numerical optimization packages to solve this problem.

6.2 The represented theorem

To derive SVM, we will assume that w can be represented as a linear combination of the training examples ($y^{(i)}$ is ± 1):

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$$

This will allow us to derive algorithms that work even for extremely high-dimensional feature sets (trillions or infinite-dimensional).

The represented theorem (very complicated, full derivation in the lecture notes) shows that you can make this assumption without losing any performance – that is, this isn't an assumption at all but gives you the optimal w.

Why?

Intuition #1: logistic regression. In logreg, we set

$$\theta^{(0)} = \begin{bmatrix} \theta_1^{(0)} \\ \vdots \\ \theta_n^{(0)} \end{bmatrix} = 0$$

Using gradient descent to find the optimal θ, we have

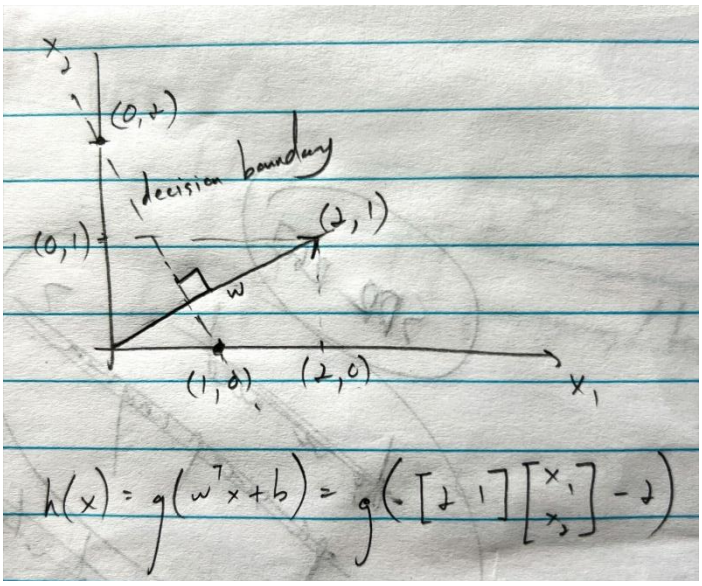
$$\theta_j := \theta_j - \alpha\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

Or

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m}\left[\left(h\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}\right]$$

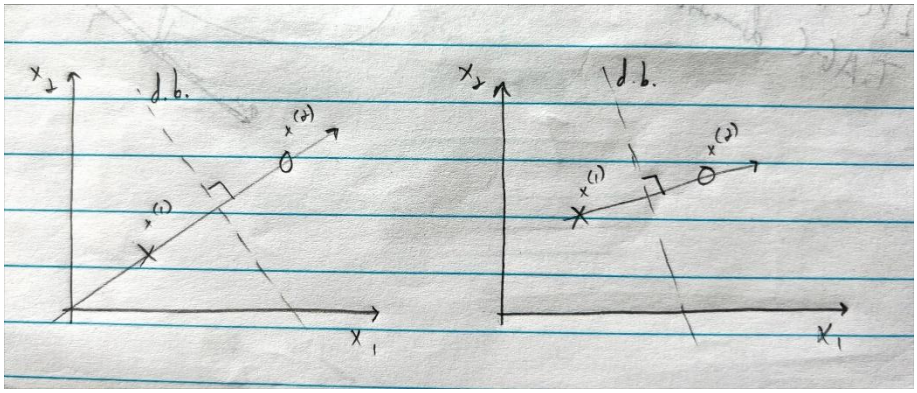
In each iteration of gradient descent, we are adding a little bit of one training example to θ, so using proof by induction, θ (and by extension, (w, b)), must be a linear combination of the training examples at every iteration.

Intuition #2: linear algebra/geometry. From our perceptron discussion, we know that w is orthogonal to the decision boundary. For example, let $w = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$ and $b = -2$:



In the case where the decision boundary passes through the origin, i.e. $b = 0$, we can see that it falls along the vector $\begin{bmatrix} -1 \\ 2 \end{bmatrix}$. Then $w^T \begin{bmatrix} -1 \\ 2 \end{bmatrix} = [2 \quad 1] \begin{bmatrix} -1 \\ 2 \end{bmatrix} = 0$. b simply offsets the decision boundary; w changes the direction.

The lecture cut out here, so I'll do my best to fill in. Ng starts off by saying suppose you have two training examples, one positive and one negative ($x^{(1)}$ and $x^{(2)}$).

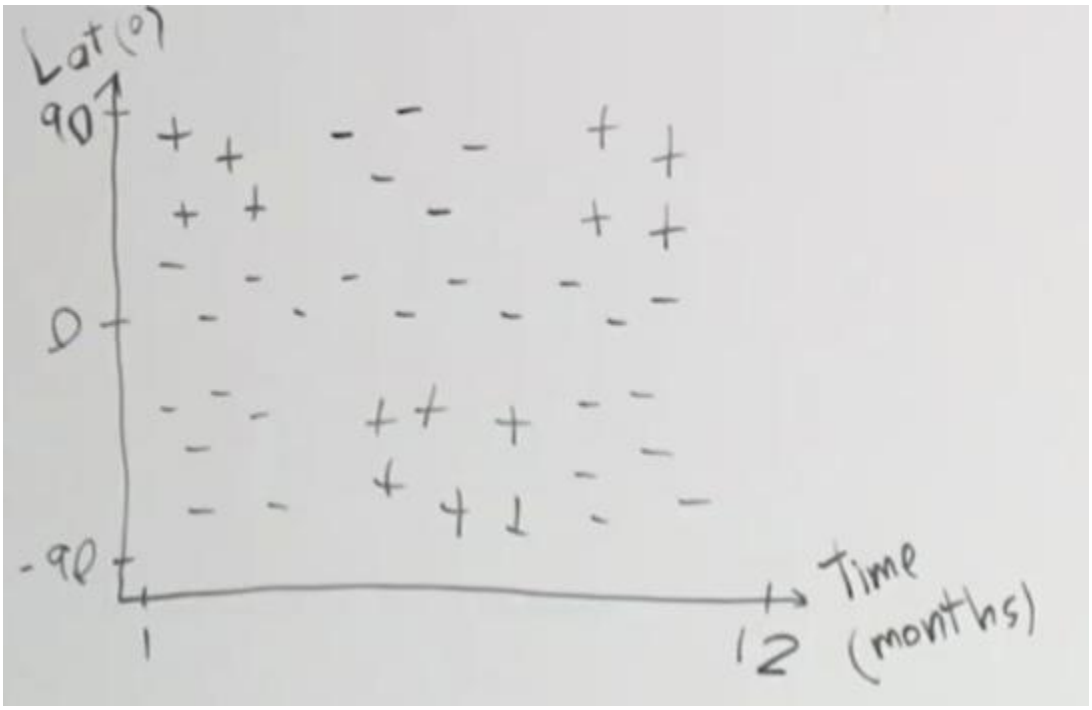


I can think of two scenarios: the examples are either linearly dependent (left pic) or linearly independent (right pic). In both cases, the optimal decision boundary bisects the distance between the examples at a right angle. w falls on the line between the examples. In the linearly dependent case, $x^{(1)}$ and $x^{(2)}$ only span a line, but w falls on that line so w can be written as a scaled version of $x^{(1)}$ or $x^{(2)}$. In the linearly independent case, $x^{(1)}$ and $x^{(2)}$ span $\mathbb{R}^2$ so w is a linear combination of $x^{(1)}$ and $x^{(2)}$.

7 Lecture 10 – decision trees and ensemble methods

7.1 Decision trees for classification

First example of nonlinear model. Let's say you have features time and location, and you have a binary classifier that tells you if you can ski.

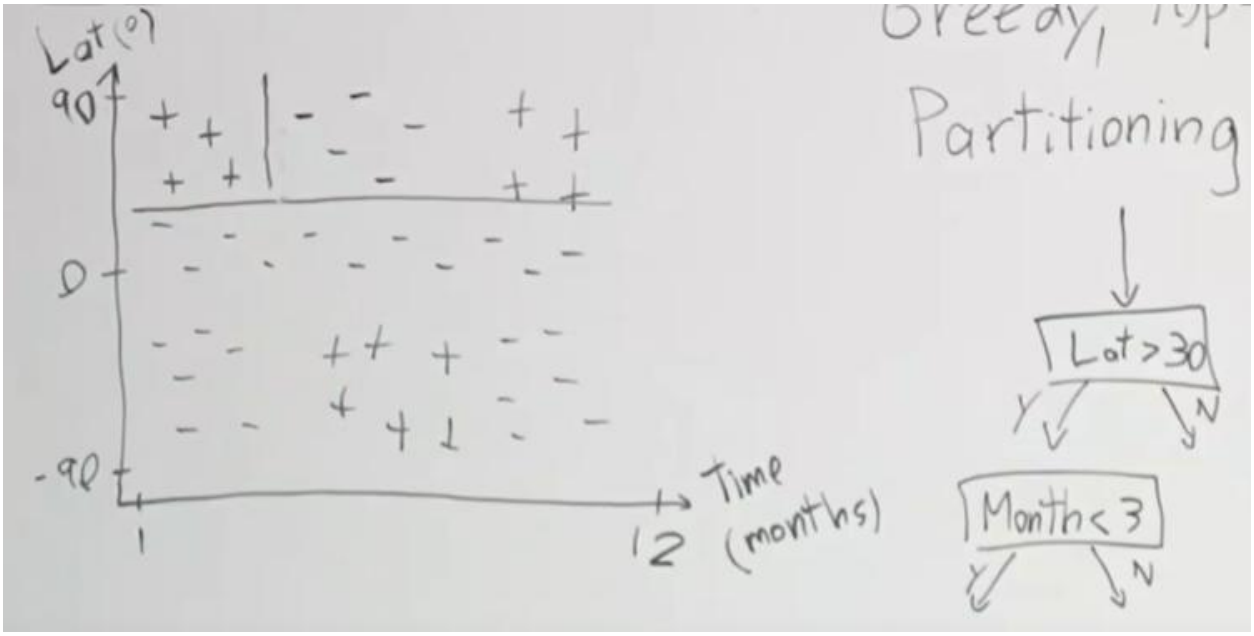


(Latitude indicates north/south location, with 0 being the equator). + means you can ski, − means you can't ski.

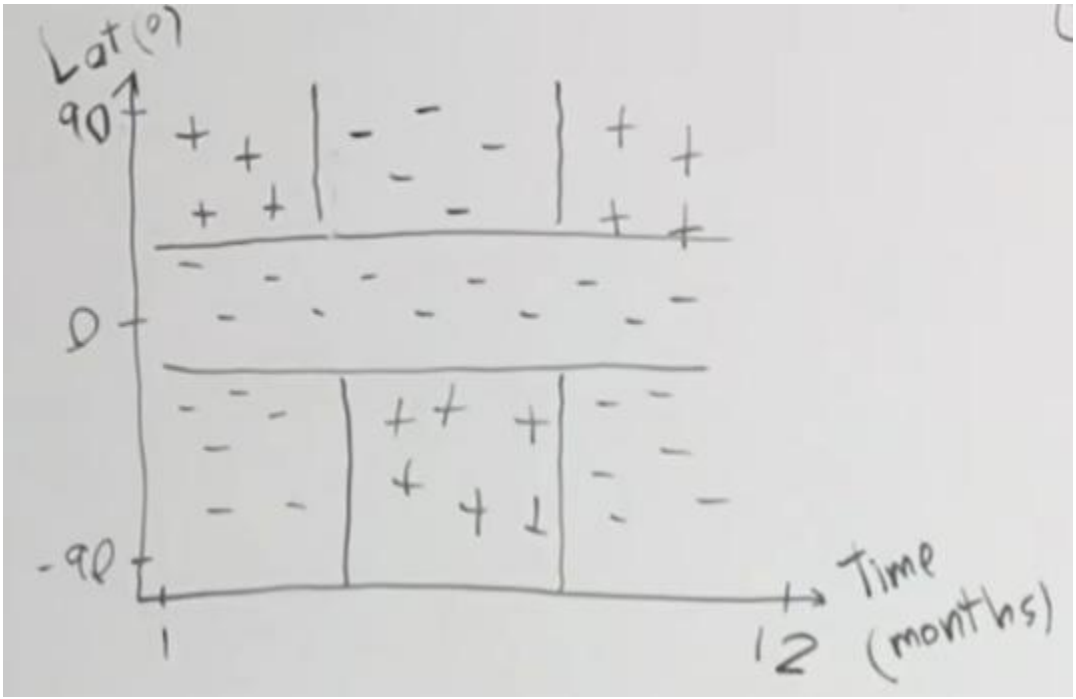
This dataset is impossible to separate linearly. Perhaps with SVM, you can do it, but decision trees are a natural way to classify the data.

We want to partition region, isolate the positive examples from the negative examples. With decision trees, it's greedy (at each step, we pick the best partition), top-down (start from whole region and partition), recursive partitioning.

For example, if we pass the data through these two questions, we form 3 partitions:



This is the goal:



Formally.

Define the region R_p for parent region. We're looking for a split function s_p s.t.

$$s_p(j,t) = \left(\{x|x_j < t, x \in R_p\}, \{x|x_j \geq t, x \in R_p\}\right) = (R_1, R_2)$$

Where j is feature number and t is threshold.

R_1 and R_2 are the children region. R_1 contains the samples in R_P s.t. $x_j < t$ and R_2 contains the samples in R_P s.t. $x_j \geq t$.

The split is defined by (or a function of) the feature of interest and the threshold of that feature.

7.1.1 Misclassification loss

How do we choose the splits?

Define the loss of a region as $L(R)$. We can define loss as the misclassification loss – the number of samples in R that you get wrong.

Given c classes, define $\hat{p}_c$ to be the proportion of samples in R that are of class c.

Then define the loss of any region as

$$L_{\text{misclass}} = 1 - \max_c \hat{p}_c$$

For your region, you want to predict the most common class there, which is $\max_c \hat{p}_c$. Then the proportion of mis-predictions is just $1 - \max_c \hat{p}_c$.

We want to pick splits that result in the greatest decrease in loss, that is, we want to maximize (parent loss) – (children loss):

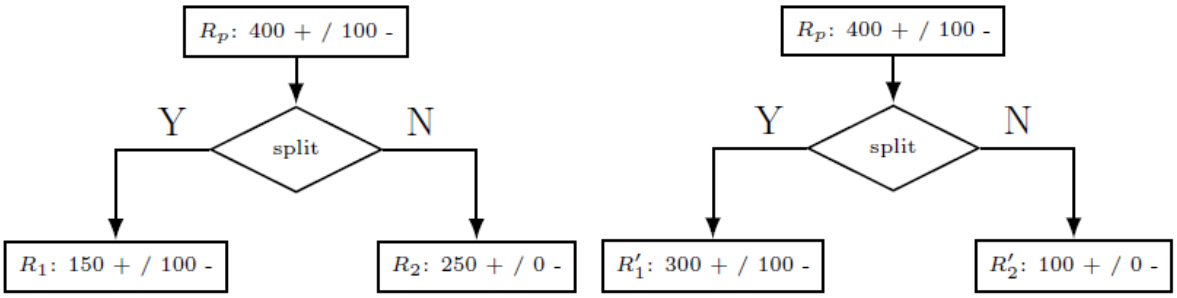
$$\max_{j,t} \left\{ L(R_p) - \frac{|R_1|L(R_1) + |R_2|L(R_2)}{|R_1| + |R_2|} \right\}$$

The children loss is the weighted loss of the two children.

Note that this equation applies regardless of the loss function we choose.

Misclassification loss has issues.

Example:



$$L(R_p) = \frac{100}{500}$$

$$|R_1| = 250, L(R_1) = \frac{100}{250}$$

$$|R_2| = 250, L(R_2) = 0$$

$$|R_1'| = 400, L(R_1') = \frac{100}{400}$$

$$|R_2'| = 100, L(R_2') = 0$$

Calculate the children loss:

$$\frac{|R_1|L(R_1) + |R_2|L(R_2)}{|R_1| + |R_2|} = \frac{250 * \frac{100}{250} + 0}{250 + 250} = \frac{100}{500}$$

$$\frac{|R_1'|L(R_1') + |R_2'|L(R_2')}{|R_1'| + |R_2'|} = \frac{400 * \frac{100}{400} + 0}{400 + 100} = \frac{100}{500}$$

These are identical to the parent loss of $L(R_p)$! This predicts that loss is not reduced in the children in either case, which is not true. This also predicts that loss is the same for the two cases, which is also not true. Misclassification loss is not sensitive to changes in class probabilities. (Another way of looking at it is that misclassification loss is the number of samples you get wrong, which is 100 for R_p and for both children cases).

7.1.2 Cross-entropy loss

Instead, define cross-entropy loss (which is closely related to Gini loss):

$$L_{\text{cross}} = -\sum_c \hat{p}_c \log_2 \hat{p}_c$$

When $\hat{p}_c = 0$, we define

$$\hat{p}_c \log_2 \hat{p}_c \equiv 0$$

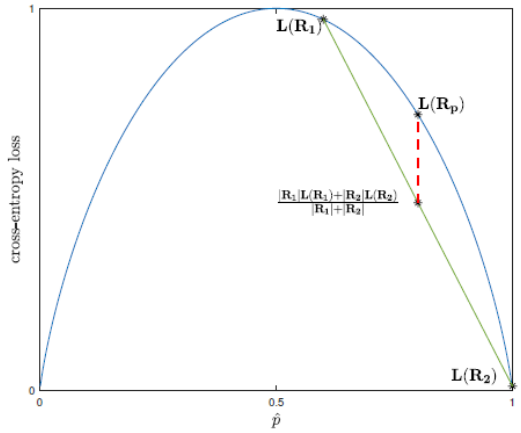
This is from information theory and measures the number of bits needed to specify the outcome or class given the distribution is known. The reduction in loss from parent to child is called information gain.

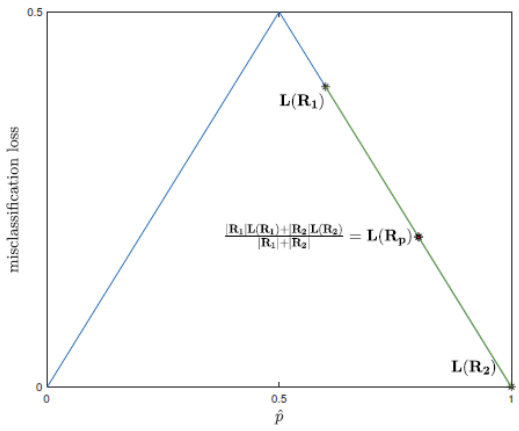
Continuing with the binary classification example, let $\hat{p}$ be the proportion of positive examples in R. The losses are

$$L_{\text{misclass}}(R) = L_{\text{misclass}}(\hat{p}) = 1 - \max(\hat{p}, 1 - \hat{p})$$

$$L_{\text{cross}}(R) = L_{\text{cross}}(\hat{p}) = -\hat{p}\log_2\hat{p} - (1-\hat{p})\log_2(1-\hat{p})$$

Let's plot the losses vs. $\hat{p}$:





The blue curve is the loss function vs. $\hat{p}$. $L(R_1), L(R_2), L(R_p)$ lie on the blue curve.

The green curve is a line through the children loss. The weighted children loss lies on the green curve.

Because the cross-entropy loss is strictly concave, the weighted children loss will always be less than the parent loss. Misclassification loss is not strictly concave and thus does not have this property.

Proof that cross-entropy loss is concave.

$$L_{\text{cross}}(\hat{p}) = -\hat{p}\log_2\hat{p} - (1-\hat{p})\log_2(1-\hat{p})$$

$$\frac{d}{d\hat{p}}L_{\text{cross}}(\hat{p}) = -\log_2\hat{p} - \frac{1}{\ln 2} + \log_2(1-\hat{p}) + \frac{1}{\ln 2} = -\log_2\hat{p} + \log_2(1-\hat{p})$$

$$\frac{d^2}{d\hat{p}^2}L_{\text{cross}}(\hat{p}) = -\frac{1}{\ln 2}\frac{1}{\hat{p}} - \frac{1}{\ln 2}\frac{1}{1-\hat{p}} = -\frac{1}{\ln 2}\frac{1}{\hat{p}(1-\hat{p})}$$

Since $\frac{1}{\hat{p}(1-\hat{p})}$ is always positive, the double derivative of L_{cross} is always negative, which means L_{cross} is strictly concave.

Proof that weighted average lies on the green curve.

Let's say you have two points on a line (x_1, y_1) and (x_2, y_2) s.t. $y_1 = ax_1 + b$ and $y_2 = ax_2 + b$.

We want to take the weighted average of y_1 and y_2, i.e. $w_1y_1 + w_2y_2$ with $w_1 + w_2 = 1$. Then

$$w_1 y_1 + w_2 y_2 = a w_1 x_1 + w_1 b + a w_2 x_2 + w_2 b = a(w_1 x_1 + w_2 x_2) + b$$

That is, if $y_3 = w_1 y_1 + w_2 y_2$, then $x_3 = w_1 x_1 + w_2 x_2$. Therefore, the weighted average of y_1 and y_2 lie on the green line between them. x_3 must be between x_1 and x_2 since $w_1 + w_2 = 1$.

7.1.3 Gini loss

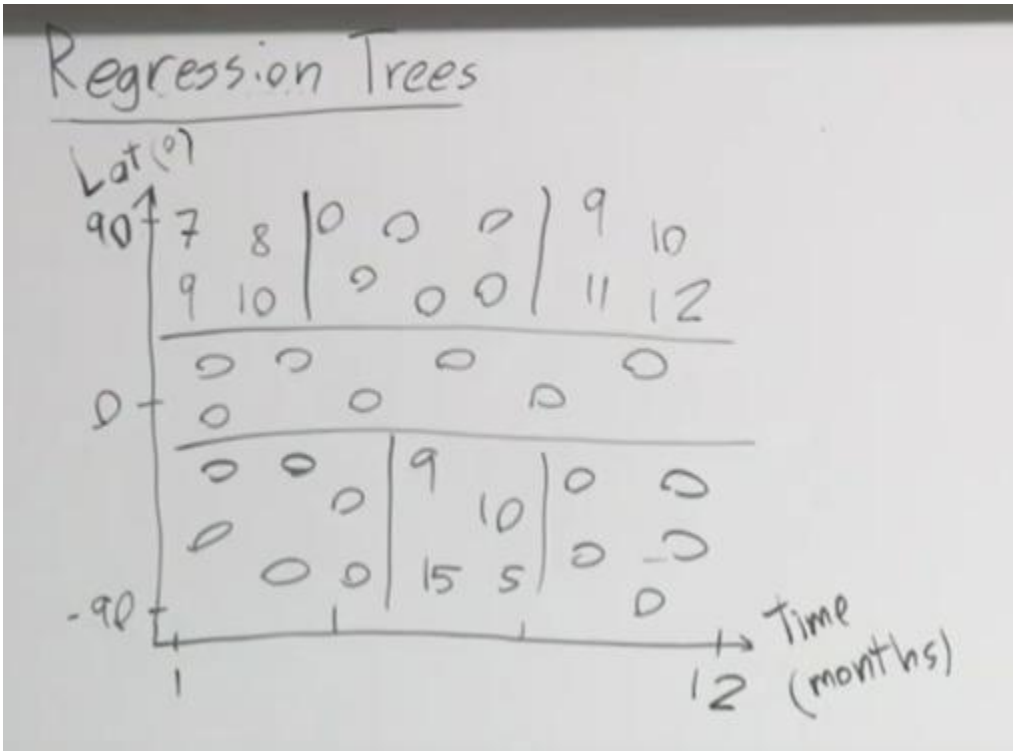
$$L_{\text{Gini}} = \sum_c \hat{p}_c (1 - \hat{p}_c)$$

Again, $\hat{p}_c$ is the proportion of each class c in the region.

This is also strictly concave, and most curves that are good for decision splits are strictly concave.

7.2 Decision trees for regression (aka regression trees)

Instead of having classes of (can ski, can't ski), what if you instead had inches of snowfall as the labels?



The tree growth process is the same as for classification, but the final prediction for a region R is the mean of the values in the region:

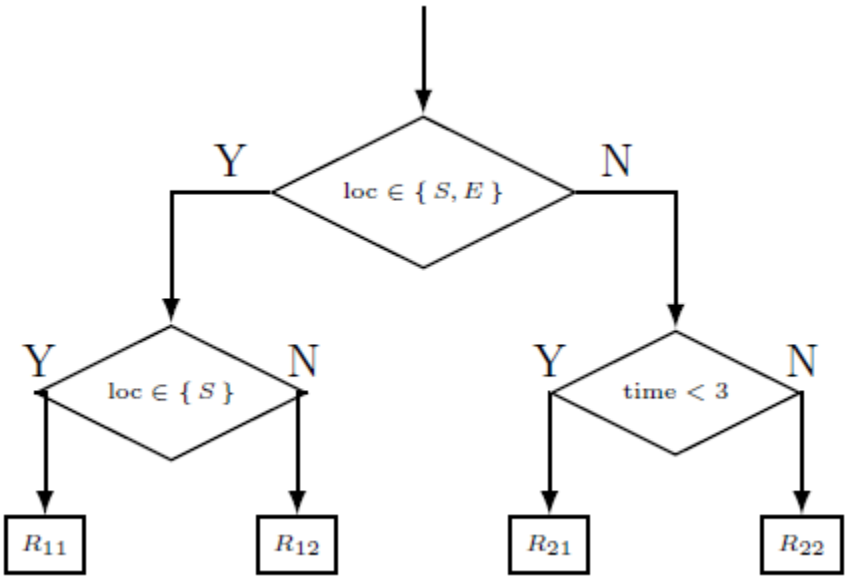
$$\hat{y} = \frac{\sum_{i \in R} y_i}{|R|}$$

And the loss function for a region R is MSE:

$$L_{\text{squared}}(R) = \frac{\sum_{i \in R}(y_i - \hat{y})^2}{|R|}$$

7.3 Categorical variables

Decision trees are good for categorical variables. For example, instead of latitude, we have a variable that takes the values of (northern hemi, equator, southern hemi).



However, for a categorical variable with S categories, the set of possible questions is 2^S, so we must be careful not to have too many categories. As the number of categories increases, the more likely it is that a quantitative variable will be better.

7.4 Regularization

Regularization helps prevent overfitting (model too complex).

1. Set a minimum leaf size – stop splitting once you reach this minimum
2. Set a maximum depth
3. Set a max number of nodes
4. Set a minimum decrease in loss – this is tempting but generally not a good idea because you may miss higher order interactions. For example, you may ask one question that doesn't result in much decrease in loss, but the follow-up question, combined with the first question, results in a high decrease in loss. For example, in the skiing example, you get more decrease in loss by asking both latitude question

and time question, but each question individually may not decrease loss by that much.

5. Pruning – grow entire tree and check misclassification loss on validation sets with certain nodes removed

7.5 Runtime

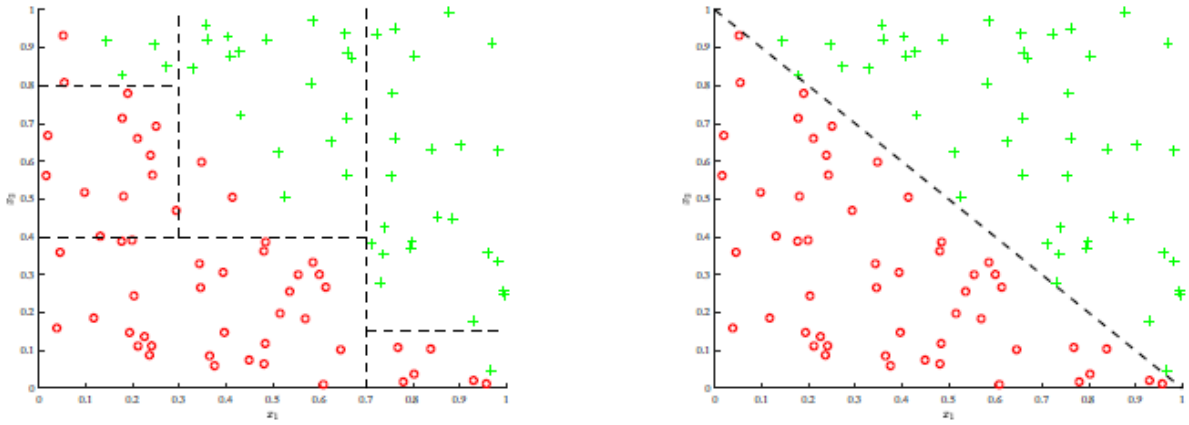
Let's say you've trained on n samples with f features to generate a tree with depth d.

At test time, your decision tree runtime is $O(d)$ where typically $d < \log_2 n$ if your tree is evenly balanced (each split is roughly 50-50).

At train time, each point is part of $O(d)$ nodes. The cost of each point at each node is $O(f)$. Then the total cost is $O(nfd)$. This is pretty fast. Data matrix has size nf, and $d \sim \log_2 n$.

7.6 No additive structure

It's difficult for decision trees to model additive structure, meaning the features add together like in logistic regression (linear model). E.g.



Decision tree on the left, linear model on the right.

7.7 Recap

Good parts:

1. Easy to explain – you're playing 20 questions with your data
2. Interpretable – easy to draw out the trees to show what it's doing
3. Can deal with categorical variables
4. Fast

Bad parts:

1. High variance
2. Bad at additive structure
3. Because of 1 and 2, they have low predictive accuracy

You can make decision trees a lot better through ensembling.

7.8 Ensembling

7.8.1 Concept

Take n random variables (RVs) x_i that are independent identically distributed (iid).

Let $\text{var}(x_i) = \sigma^2$, and let the sample mean be $\bar{x} = \frac{1}{n}\sum_i x_i$. Then

$$E(\bar{x}) = \frac{1}{n}\sum_i \mu_i$$

$$\text{var}(\bar{x}) = \frac{\sigma^2}{n}$$

Proof.

$$E(\bar{x}) = \frac{1}{n}\sum_i E(x_i) = \frac{1}{n}\sum_i \mu_i$$

$$\text{var}(\bar{x}) = E\left(\left(\frac{1}{n}\sum_i x_i\right)^2\right) - \bar{\mu}^2 = \frac{1}{n^2}E\left(\sum_i x_i \sum_j x_j\right) - \bar{\mu}^2 = \frac{1}{n^2}E\left(\sum_{i,j} x_i x_j\right) - \bar{\mu}^2$$
$$= \frac{1}{n^2}\sum_{i,j} E(x_i x_j) - \bar{\mu}^2$$

Since x_i are iid, $E(x_i x_j) = E(x_i)E(x_j) = \mu_i \mu_j$ when $i \neq j$. Then

$$\text{var}(\bar{x}) = \frac{1}{n^2}\left(\sum_i E(x_i^2) + \sum_{i\neq j} \mu_i \mu_j\right) - \frac{1}{n^2}\left(\sum_i \mu_i\right)^2$$
$$= \frac{1}{n^2}\sum_i E(x_i^2) + \frac{1}{n^2}\sum_{i\neq j} \mu_i \mu_j - \frac{1}{n^2}\left(\sum_i \mu_i^2 + \sum_{i\neq j} \mu_i \mu_j\right)$$
$$= \frac{1}{n^2}\sum_i E(x_i^2) - \frac{1}{n^2}\sum_i \mu_i^2 = \frac{1}{n^2}\sum_i \sigma_i^2 = \frac{\sigma^2}{n}$$

If the mean of the RVs is our estimate for the real mean, then more RVs = less variance.

Often, the independence assumption is not true, so your RVs are just identically distributed (id). If we know the correlation between RVs, ρ, then

$$\text{var}(\bar{x}) = \rho\sigma^2 + \frac{1-\rho}{n}\sigma^2$$

If the RVs are fully correlated, that is $\rho = 1$, then $\text{var}(\bar{x}) = \sigma^2$.

If the RVs are uncorrelated, that is $\rho = 0$, then $\text{var}(\bar{x}) = \frac{\sigma^2}{n}$.

To get the best estimate using ensembling, we want as many models as possible, and we want them to be as de-correlated as possible.

7.8.2 Ways to ensemble

1. Different algorithms, e.g. random forest + SVM + neural networks. People on Kaggle often do this, and it's generally effective but requires a lot of work (you need to implement multiple algorithms).
2. Different training sets – but this is limited. It takes a lot of effort to collect high quality data; you can't always simply get more.
3. Bagging (approximates having different training sets). For decision trees, this is known as random forests.
4. Boosting. For decision trees, this is known as adaboost or xgboost.

7.8.3 Bagging

It's a contraction of bootstrap aggregation.

Bootstrapping in statistics is a way to measure the uncertainty of your estimate.

Let's say you have a true population P and you sample your training set S from P. Ideally you would draw multiple training sets from P, i.e. $S_1, S_2, S_3, \ldots$ and train on each to find the error of the estimate from each training set, but you generally don't have time to do that, so you assume that $P = S$, that your population is your sample.

Then you draw your training samples sets Z_i from S, with replacement. Z_i are known as bootstrapped samples. In statistics, this allows you to measure the uncertainty (variance) of your estimates using Z_i.

In machine learning, we aggregate the estimates over Z_i to get an ensemble estimate that reduces variance.

Formally,

1. You have bootstrap training sets $Z_1, \ldots, Z_M$

2. Train model G_m on Z_m
3. Ensemble estimate is $G(m) = \frac{\sum_{m=1}^{M} G_m(x)}{M}$

Bias-variance analysis:

$$\text{var}(\bar{x}) = \rho\sigma^2 + \frac{1-\rho}{M}\sigma^2$$

By sampling with replacement, we (kind of) make the training sets independent of each other, which reduces ρ. However, usually Z_m are still fairly highly correlated since each Z will be 2/3 of S (where does this come from?). But this does result in lower correlation than if we always trained on S.

Higher M results in less variance by driving down the second term. Higher M does not result in overfitting. Generally, we'll keep sampling Z_m until the error stops decreasing.

The bias of each individual prediction slightly increases because you're training on fewer samples, but the reduction in variance outweighs it.

7.8.3.1 Bagging in decision trees (random forests)

DTs are high variance, low bias, which makes them a good fit for bagging.

For random forests, the only additional requirement is at each split, consider only a fraction of your total features, e.g. for one split you only look at latitude and then for another you only look at time. This way we reduce correlation across bootstrapped samples, at the acceptable cost of increased bias due to the restricted feature space.

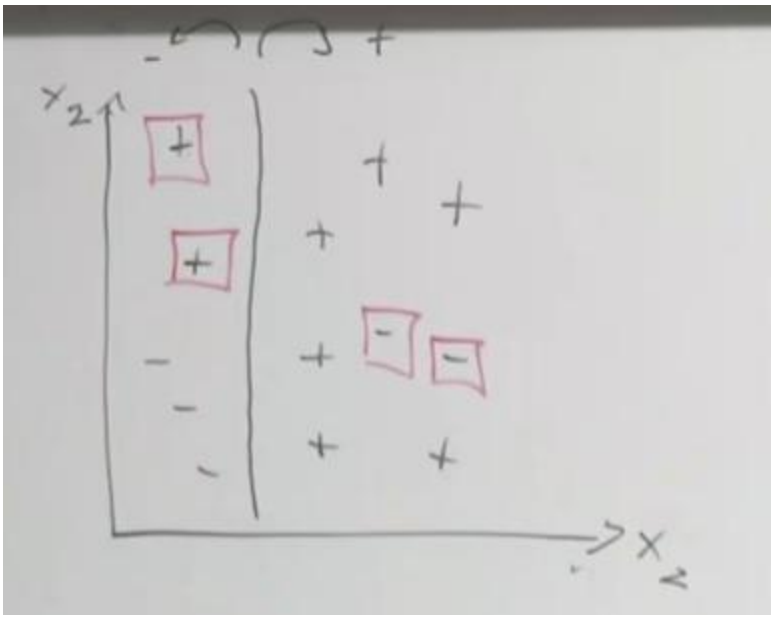
7.8.4 Boosting

Boosting is for decreasing bias and is additive.

In boosting, we train "decision stumps" – decision trees of depth 1, i.e. only one question. Each stump has high bias, which makes boosting a good fit.

Train on the first decision stump, and find the misclassified samples. On the next decision stump, increase the weights of the misclassified samples from the previous decision stump. Do this for each subsequent, sequential decision stump.

E.g. first stump, with red squares around the misclassified samples:



For each classifier (decision stump), calculate its weight on the final prediction. E.g. in AdaBoost, the weight is proportional to $\alpha = \log\left(\frac{1-\text{err}}{\text{err}}\right)$. If G_m is the prediction of each classifier, then the final prediction is

$$G(x) = \sum_m \alpha_m G_m$$

8 Lectures 11/12 – neural networks

Deep learning is a set of techniques, a subset of machine learning, that is used for computer vision, NLP, and speech recognition.

Computationally expensive; need to parallelize the code using GPUs.

Large datasets now available that deep learning can take advantage of.

8.1 Classification

8.1.1 Goal 1: find cats in images

Ex. Find cats in images (binary classification). 1 = presence of a cat, 0 = absence of a cat. Assume just one cat per image.

Let's say we have an image that's 64x64x3 (64 pixels by 64 pixels by 3 channels for a color image, RGB).

Let's flatten this 3D matrix into a 1D vector. Our logistic regression model is

$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \rightarrow wx + b \rightarrow \hat{y} = g(\theta^T x) = g(wx + b) = \frac{1}{1 + e^{-(wx+b)}}$$

$$x \in \mathbb{R}^{12288 \times 1}$$

$$w \in \mathbb{R}^{1 \times 12288}$$

$$b \in \mathbb{R}$$

To train the model,

1. Initialize w, weights, and b, bias
2. Find the optimal w and b (minimize cost/loss function)
3. Use $\hat{y}$ to predict the true labels

For logistic regression, the loss function is

$$\mathcal{L} = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

We use gradient descent to minimize J:

$$w \coloneqq w - \alpha \frac{\partial J}{\partial w}$$

$$b \coloneqq b - \alpha \frac{\partial J}{\partial b}$$

The number of parameters of our model, 12289, depends on the size of the input. This is undesirable and will be changed later.

A neuron is an operation with two parts: linear + activation. In logistic regression, the linear part is $wx + b$ and the activation part is the sigmoid function. That is, the neuron is

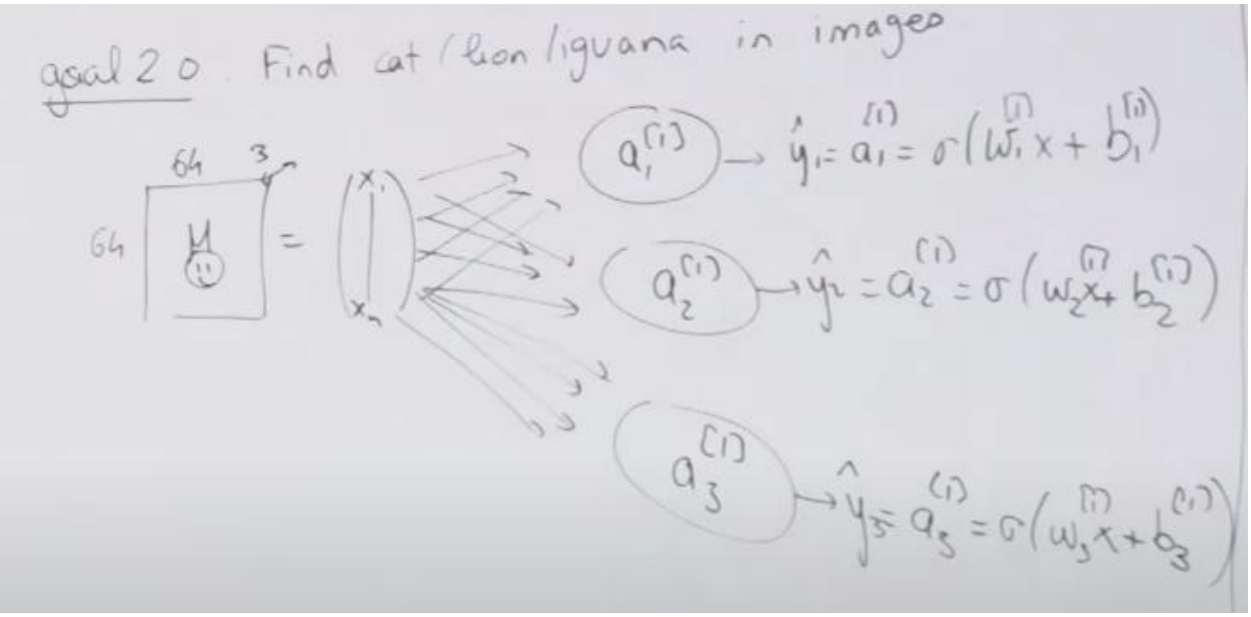
$$wx + b \rightarrow \hat{y} = g(\theta^T x) = g(wx + b)$$

The model is equal to architecture + parameters. In logistic regression, the architecture is one neuron and the parameters are w and b.

8.1.2 Goal 2: find cats/lions/iguanas in images

Ex. Find cat/lion/iguana in images.

How would you modify the model from before? We can add two additional neurons for a total of 3:



The superscript brackets $[1]$ indicate the index of the layer. In this network, there is one layer of neurons. The subscript numbers identify the neuron inside the layer. We have 3 neurons in layer 1.

This model has 12289*3 parameters.

The dataset in the first exercise is images that are classified as either has cat or doesn't have cat.

In this exercise, the images can be labeled using a binary vector with 3 elements. A value of 1 flags that the animal is present. For example, $\begin{bmatrix}1\\1\\1\end{bmatrix}$ means all 3 animals are present, while $\begin{bmatrix}0\\0\\0\end{bmatrix}$ means none are present.

The process of training the model is the same, but now we have 3 neurons. a_1 is responsible for detecting a cat, a_2 is responsible for detecting a lion, and a_3 is responsible for detecting an iguana. The three neurons do not affect each other and are trained independently.

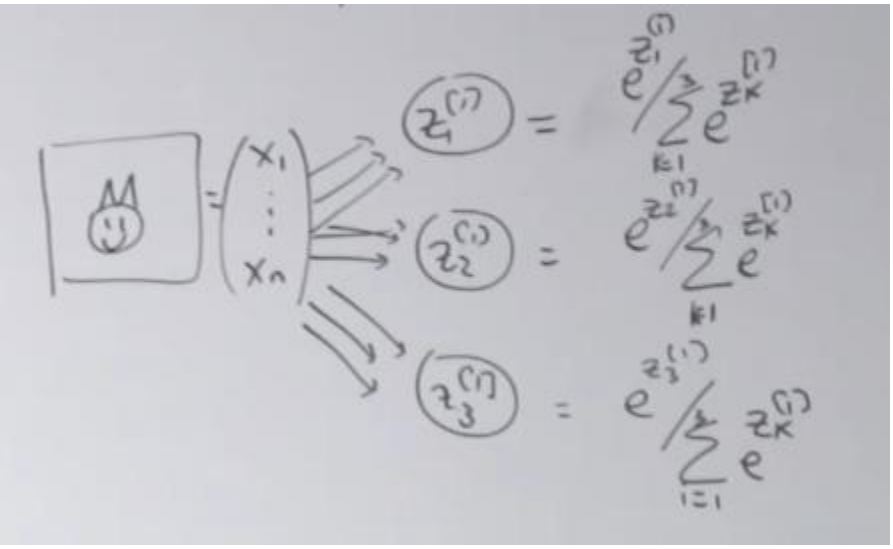
The loss function becomes

$$\mathcal{L} = -\sum_c (y_c \log \hat{y}_c + (1 - y_c) \log(1 - \hat{y}_c))$$

Since $\hat{y}_c$ depends only on w_c, b_c, $\frac{\partial \mathcal{J}}{\partial w_c}$ and $\frac{\partial \mathcal{J}}{\partial b_c}$ are exactly the same as in the single neuron case (since only the term with class c remains after taking the derivative; the others are zero).

8.1.3 Goal 3: add a constraint – each image has at most one animal

In this case, we want to identify the animal in the image. There are still 3 classes, but only 1 class per picture. We use softmax regression, which looks like this:



Where we've introduced the notation that

$$z_c = w_c x + b_c$$

The output of each neuron is the probability that the image contains class c:

$$\hat{p}(c) = \frac{e^{z_c^{[1]}}}{\sum_{c=1}^{3} e^{z_c^{[1]}}}$$

The outputs form a probability distribution, which, unlike the previous example, means that the outputs of the neurons need to add up to 1:

$$\sum_{c=1}^{3} \hat{p}(c) = 1$$

The final predicted label can only take on the values $\begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix}$.

Like the last example, this model has 12289*3 parameters.

We cannot use the same loss function as in the previous case. The predictions, $\hat{y}_c$ or $\hat{p}(c)$ depend on the parameters for all classes, not just class c. Unlike the previous case, terms from the other classes do not become zero when taking derivative w.r.t. class c.

The loss we'll use for softmax is called cross-entropy loss and is common in deep learning:

$$\mathcal{L}_{ce} = -\sum_c y_c \log \hat{y}_c$$

8.2 Regression

Let's say instead of identifying whether or not a cat is present, we want to estimate the age of the cat in the image. How can we change our neural network?

Change the activation function. For linear regression, we use $\hat{y} = wx + b$, that is, the prediction is exactly equal to the linear part. Graphically, the activation function is given by $y = x$.

Another activation function that's common in deep learning is the ReLU (rectified linear unit) function, which is

$$y = \begin{cases} x, x \geq 0 \\ 0, x < 0 \end{cases}$$

The ReLU function makes sense to fit data that is non-negative, like age.

The loss function also needs to be updated to

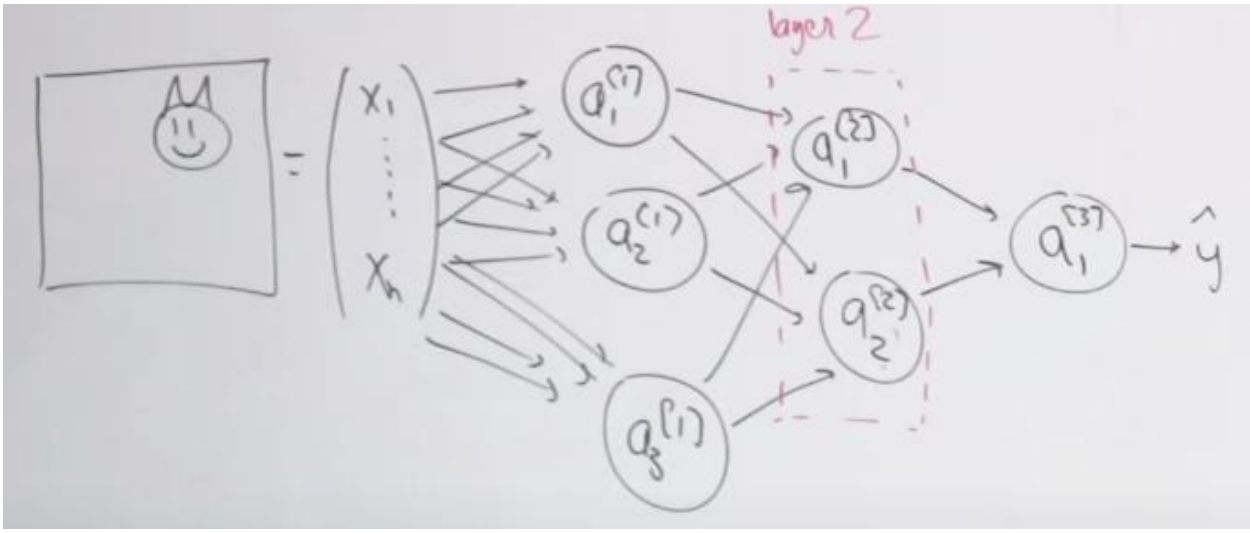
$$\|\hat{y} - y\|^2$$

Or

$$|\hat{y} - y|$$

8.3 Neural networks

Goal: given an image, is there a cat?



In a neural network, the number of neurons in the final layer needs to be equal to the number of classes you have (1 for regression).

The number of parameters in this network is (n+1)*3 + (2+1)*3 + (2+1)*1. Remember +1 is for the bias in each neuron.

A layer is a cluster of neurons that are not connected to each other.

In the figure, layer 1 is the input layer because it sees the inputs, layer 3 is the output layer because it sees the output, and layer 2 is a hidden layer. It's hidden because it doesn't see the inputs or the output.

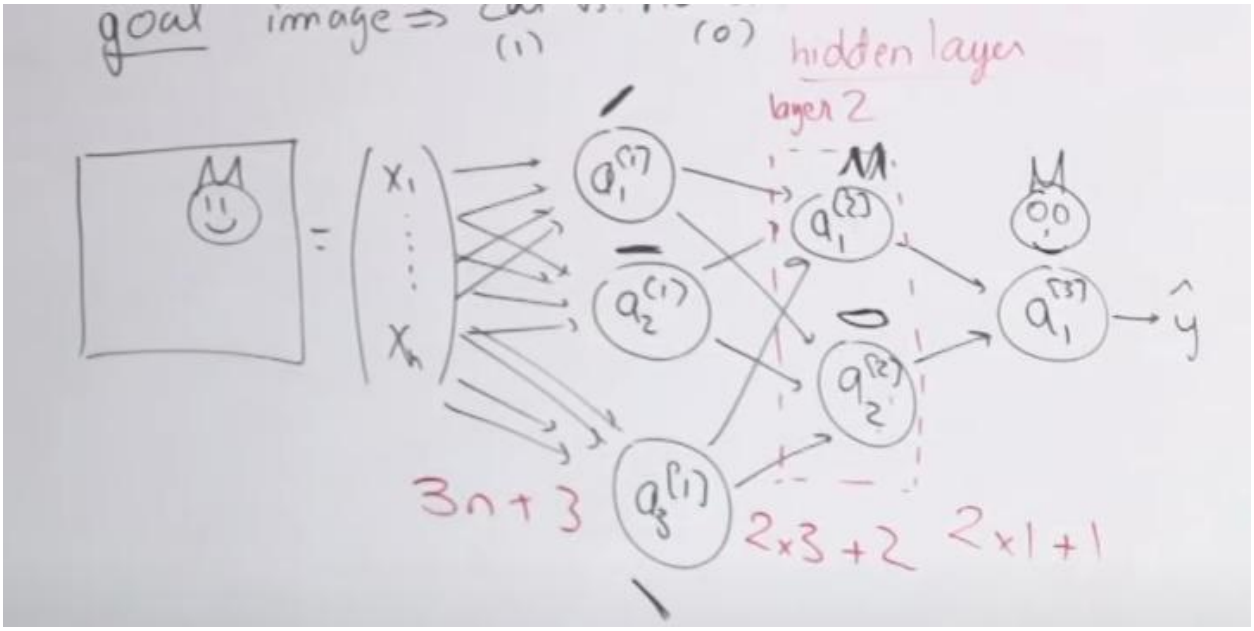
For example, the input layer may detect edges, which the hidden layer then translates to ears and a mouth, and finally the output layer constructs the image of a cat.

Intuitively, the input layer receives pixel information, so it can't do much more than detect edges. The hidden layer receives edge information, which it can then process to detect more complex structures. And so on.

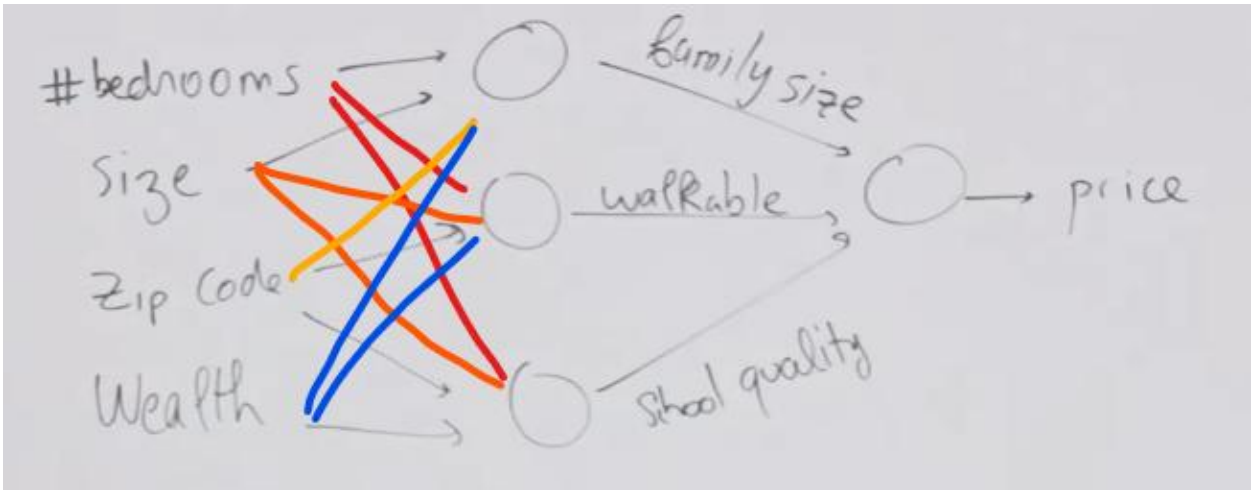
However, another reason we call it the hidden layer is because we don't know exactly what it will detect. As you traverse the layers, the neurons in the deeper layers are able to detect more complex information. You need to make sure your neural network is complex enough for your application.

Nobody knows how many layers and how many neurons per layer you need. We pick a bunch of different architectures and train/validate/test them and see which one is the best.

Previous experience is valuable. You can try and gauge how complex an application is based on applications you've built before (more or less complex, equally complex).



Another example – house price prediction. As a human, you might construct a neural network like this (black lines), where you connect features/neurons to each other based on human reasoning. In practice, we make all possible connections in the neural network and let the network figure it out (the colored lines).



This is why neural networks are referred to as black box models – we don't understand the connections inside the network; the things that the network is figuring out internally may not have any physical meaning.

This is also known as end-to-end learning. We don't constrain the network internally; we're training based only on the input and output.

8.3.1 Forward propagation equations

The equations that control the forward propagation of information through the neural network.

Layer 1:

$$z^{[1]} = w^{[1]}x + b^{[1]}$$

$$a^{[1]} = g\left(z^{[1]}\right)$$

Dimensionally, we have

$$b^{[1]}, z^{[1]}, a^{[1]} \in \mathbb{R}^{k1}$$

$$x \in \mathbb{R}^{n}$$

$$w^{[1]} \in \mathbb{R}^{k1 \times n}$$

Where $k1$ is the number of neurons in layer 1.

Layer 2:

$$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = g\left(z^{[2]}\right)$$

$$b^{[2]}, z^{[2]}, a^{[2]} \in \mathbb{R}^{k2}$$

$$w^{[2]} \in \mathbb{R}^{k2 \times k1}$$

Where $k2$ is the number of neurons in layer 2.

Layer 3:

$$z^{[3]} = w^{[3]}a^{[2]} + b^{[3]}$$

$$a^{[3]} = g\left(z^{[3]}\right)$$

$$b^{[3]}, z^{[3]}, a^{[3]} \in \mathbb{R}^{k3}$$

$$w^{[3]} \in \mathbb{R}^{k3 \times k2}$$

Where $k3$ is the number of neurons in layer 3.

$$\hat{y} = a^{[3]}$$

The number of parameters for a layer is given by (number of inputs + 1)*(number of neurons).

1. Layer 1: $(n+1) \times k1$
2. Layer 2: $(k1+1) \times k2$
3. Layer 3: $(k2+1) \times k3$

Total number of parameters is

$$nk1 + k1 + k1k2 + k2 + k2k3 + k3 = nk1 + k1k2 + k2k3 + (k1 + k2 + k3)$$

8.3.1.1 Batching

What happens for an input batch of m examples?

$$X = [x^{(1)} \quad x^{(2)} \quad \dots \quad x^{(m)}] \in \mathbb{R}^{n \times m}$$

We want to parallelize our equations as much as possible for computational efficiency when we are given a batch of inputs.

$$Z^{[1]} = w^{[1]}X + \tilde{b}^{[1]} = [z^{[1](1)} \quad \dots \quad z^{[1](m)}] \in \mathbb{R}^{k1 \times m}$$

$$A^{[1]} = g\left(Z^{[1]}\right) \in \mathbb{R}^{k1 \times m}$$

The shape of $w^{[1]}$ doesn't change, but $b^{[1]}$ needs to be broadcasted s.t.

$$\tilde{b}^{[1]} = [b^{[1]} \quad \dots \quad b^{[1]}] \in \mathbb{R}^{k1 \times m}$$

Numpy automatically does the broadcasting.

$$Z^{[2]} = w^{[2]}A^{[1]} + \tilde{b}^{[2]} \in \mathbb{R}^{k2 \times m}$$

$$A^{[2]} = g\left(Z^{[2]}\right) \in \mathbb{R}^{k2 \times m}$$

$$Z^{[3]} = w^{[3]}A^{[2]} + \tilde{b}^{[3]} \in \mathbb{R}^{k3 \times m}$$

$$A^{[3]} = g\left(Z^{[3]}\right) \in \mathbb{R}^{k3 \times m}$$

8.3.2 Optimizing the parameters

Optimizing $w^{[1]}, w^{[2]}, w^{[3]}, b^{[1]}, b^{[2]}, b^{[3]}$. We need to define a loss/cost function. Conventionally, loss is for one example and cost is for multiple examples. If $\mathcal{L}$ is loss and $\mathcal{J}$ is cost,

$$\mathcal{J}(\hat{y}, y) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}^{(i)}$$

When m is the number of examples, then we're going for batch gradient descent. When $m = 1$, it's stochastic gradient descent.

Loss is just the logistic loss (in the example of binary classification):

$$\mathcal{L}^{(i)} = -\left[y^{(i)} \log \hat{y}^{(i)} + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)\right]$$

8.3.3 Backward propagation

Backward propagation is just iteratively updating (optimizing) the parameters of the model, starting from the last layer.

$$\forall\, l,$$

$$w^{[l]} := w^{[l]} - \alpha \frac{\partial \mathcal{J}}{\partial w^{[l]}}$$

$$b^{[l]} := b^{[l]} - \alpha \frac{\partial \mathcal{J}}{\partial b^{[l]}}$$

We start with the last layer to take advantage of the chain rule of calculus.

In our 3-layer example, we start with $\frac{\partial \mathcal{J}}{\partial w^{[3]}}$:

$$\frac{\partial \mathcal{J}}{\partial w^{[3]}} = \frac{\partial \mathcal{J}}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}} \cdot \frac{\partial z^{[3]}}{\partial w^{[3]}}$$

Then $\frac{\partial \mathcal{J}}{\partial w^{[2]}}$ becomes easy:

$$\frac{\partial \mathcal{J}}{\partial w^{[2]}} = \frac{\partial \mathcal{J}}{\partial z^{[3]}} \cdot \frac{\partial z^{[3]}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial w^{[2]}}$$

Note that we've already computed $\frac{\partial \mathcal{J}}{\partial z^{[3]}} = \frac{\partial \mathcal{J}}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}}$ in the previous step.

Now for the first layer:

$$\frac{\partial \mathcal{J}}{\partial w^{[1]}} = \frac{\partial \mathcal{J}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}} \cdot \frac{\partial z^{[1]}}{\partial w^{[1]}}$$

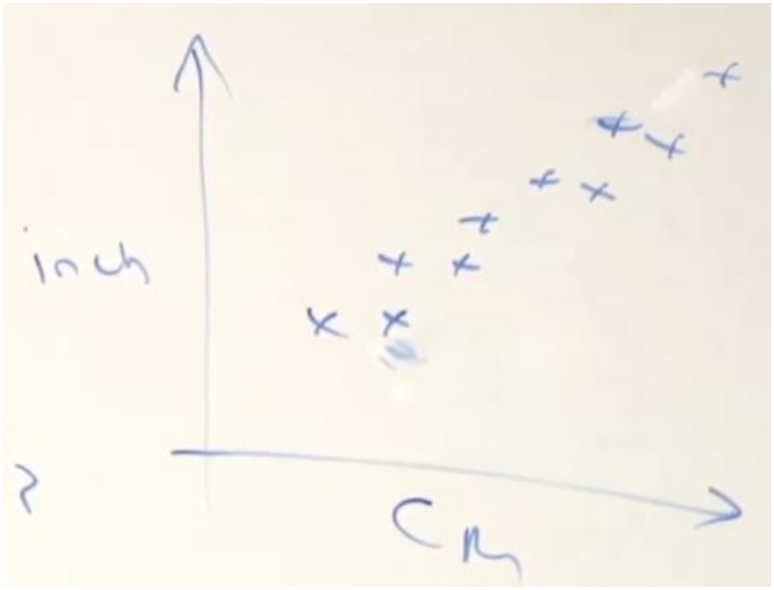
As before, we've already computed $\frac{\partial \mathcal{J}}{\partial z^{[2]}}$ in the previous step.

9 Lecture 15.5 – PCA

Principal component analysis is a technique to figure out if your high-dimensional data lives on a lower-dimensional subspace, thus allowing you to reduce the dimension of your data.

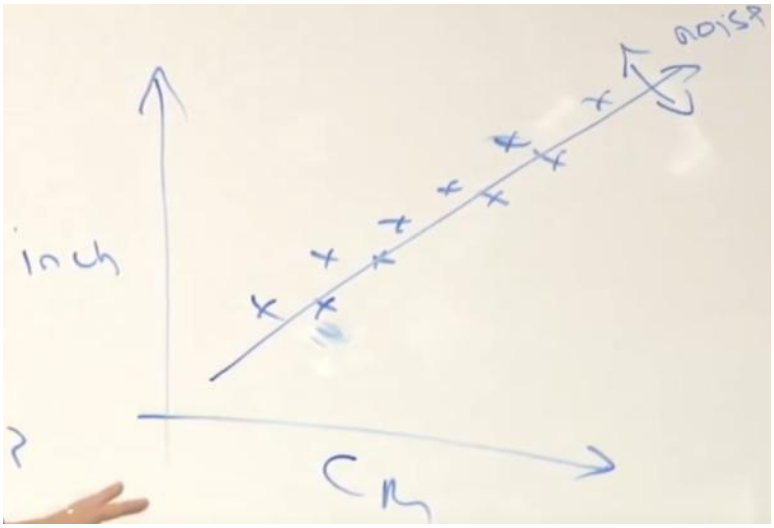
Let's say we have an unlabeled dataset $\{x^{(1)}, \ldots, x^{(m)}\} \in \mathbb{R}^n$. We would like to reduce the dimension from n to k, where (hopefully) $k \ll n$.

For example, let's say we measure the height of students in a school in both inches and cm. Because of roundoff to the nearest inch or cm, the data is highly correlated but not perfectly correlated.



So we have a 2D dataset, but it's really 1D living in 2D space.

PCA will figure out the principal axis of the variation of the data and project the data onto the principal axis so your 2D data becomes cleaned-up, less noisy 1D data. Noise/variation is orthogonal to the principal axis.



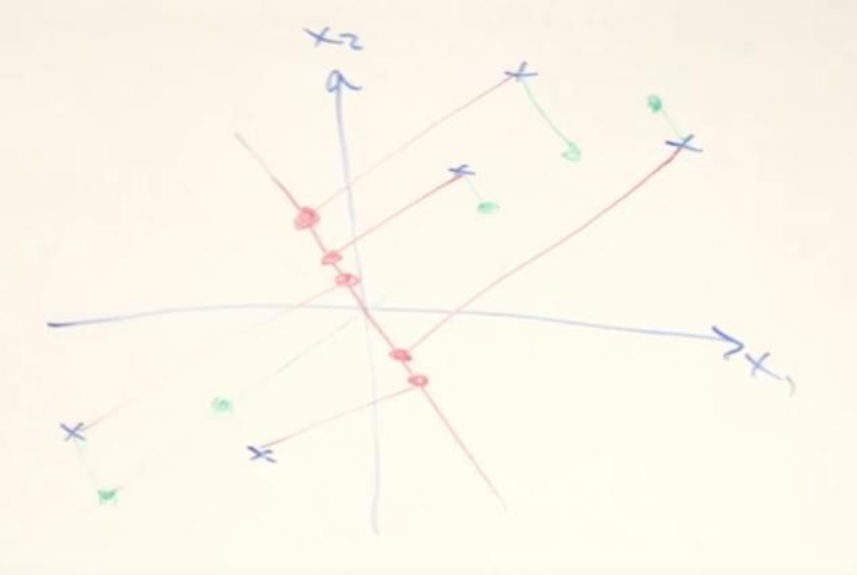
In practice, PCA is used for high dimensions, like 10000 to 100.

Before running PCA, we need pre-processing.

1. First, zero out the mean
   a. $\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$
   b. $x^{(i)} := x^{(i)} - \mu$
2. Second, standardize the variance to 1
   a. $\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m} {x_j^{(i)}}^2$
   b. $x_j^{(i)} := \frac{x_j^{(i)}}{\sigma_j}$

Now, all of your features have zero mean and variance 1. This pre-processing is often used in all kinds of learning algorithms, supervised learning included.

Now, let's say you have the following 5 data points after pre-processing.



We've drawn two candidates for the principal axis of variation, green and red. Green looks much better, but why? The orthogonal projection of the data onto the green line results in a much smaller sum of squared distances. PCA wants to minimize the sum of squares of the projection vectors.

The green dots are spread out, but the red dots are close together. A mathematically equivalent definition of PCA: if you want to preserve the variability of the data, then you want to find the principal axis on which the data is spread as much as possible.

We want to find the unit vector $\|u\|$ that points in the direction of the principal axis. If $\|u\| = 1$, then the length of the projection of $x^{(i)}$ onto u is $u^T x^{(i)}$.

(Quick proof): let p be the projection of $x^{(i)}$ onto u and e be the error b/w $x^{(i)}$ and p, i.e. $e = x^{(i)} - p$. Since p falls along u, we can write $p = \alpha u$. Since p and e are orthogonal,

$$p \cdot e = 0 = p \cdot \left(x^{(i)} - p\right) = \alpha u \cdot \left(x^{(i)} - \alpha u\right) \rightarrow \alpha u \cdot x^{(i)} = \alpha^2 u \cdot u \rightarrow$$

$$\alpha = \frac{u \cdot x^{(i)}}{u \cdot u}$$

Since $u \cdot u = \|u\|^2 = 1$, $\alpha = u \cdot x^{(i)} = u^T x^{(i)}$. $\|p\| = \alpha\|u\| = \alpha$.

To find the principal axis, we want

$$\max_{u:\|u\|=1} \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)^T} u\right)^2$$

$$\|u\| = 1$$

We want to maximize the lengths of $x^{(i)}$ along the principal axis to capture the most variation.

Since $x^{(i)^T} u = u^T x^{(i)^T}$, we have

$$\frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)^T} u\right)^2 = \frac{1}{m}\sum_{i=1}^{m} u^T x^{(i)} x^{(i)^T} u = u^T \left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)} x^{(i)^T}\right) u = u^T \Sigma_{xx} u$$

Where Σ_{xx} is the covariance matrix of x.

$$\Sigma_{xx} = E(xx^T) - E(x)E(x)^T$$

Since we subtracted the sample mean $\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$ from all samples $x^{(i)}$, $E(x) = 0$. Then Σ_{xx} is the sample mean of xx^T, or

$$\Sigma_{xx} = E(xx^T) = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} x^{(i)^T}$$

Then we want

$$\max_{u:\|u\|=1} u^T \Sigma_{xx} u$$

u is the principal eigenvector of Σ_{xx}.

Quick aside: If $Au = \lambda u$, then u is an eigenvector of A with eigenvalue λ. If your goal is

$$\max_{u:\|u\|=1} u^T \Sigma_{xx} u$$

Then you form a Lagrangian

$$\mathcal{L}(u, \lambda) = u^T \Sigma_{xx} u - \lambda(u^T u - 1)$$

Take gradient and set to 0:

$$\nabla_u(\mathcal{L}) = \Sigma_{xx} u - \lambda u \stackrel{\text{set}}{=} 0 \rightarrow \Sigma_{xx} u = \lambda u$$

So u is an eigenvector of Σ_{xx}.

If you want to choose a 1D subspace to approx. your data x, choose the dimensions of that subspace to be given by the principal eigenvector of the covariance matrix of the data Σ_{xx}.

In the general case, if you wish to project the data to k dimensions, then set $u_1, u_2, \dots, u_k$ to be the top k eigenvectors of Σ_{xx} (corresponding to eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_k$). Top k ranked according to magnitude of λ.

Ex. Have $x^{(i)} \in \mathbb{R}^n, n = 1000$. Want to reduce the number of dimensions to $k = 10$. Compute $u_1, \dots, u_k$. Now transform each $x^{(i)}$ by projecting it onto each of your new basis vectors u:

$$x^{(i)} \rightarrow \left(u_1^T x^{(i)}, u_2^T x^{(i)}, \dots, u_{10}^T x^{(i)}\right) = y^{(i)} \in \mathbb{R}^k$$

Your new representation is k-dimensional.

Because Σ_{xx} is symmetric, its eigenvectors are orthogonal, and which means $u_1, \dots, u_k$ form an orthonormal basis for $\mathbb{R}^k$.

In matrix form,

$$U = [u_1 \quad \dots \quad u_k] \in \mathbb{R}^{n \times k}$$

$$y^{(i)} = U^T x^{(i)}$$

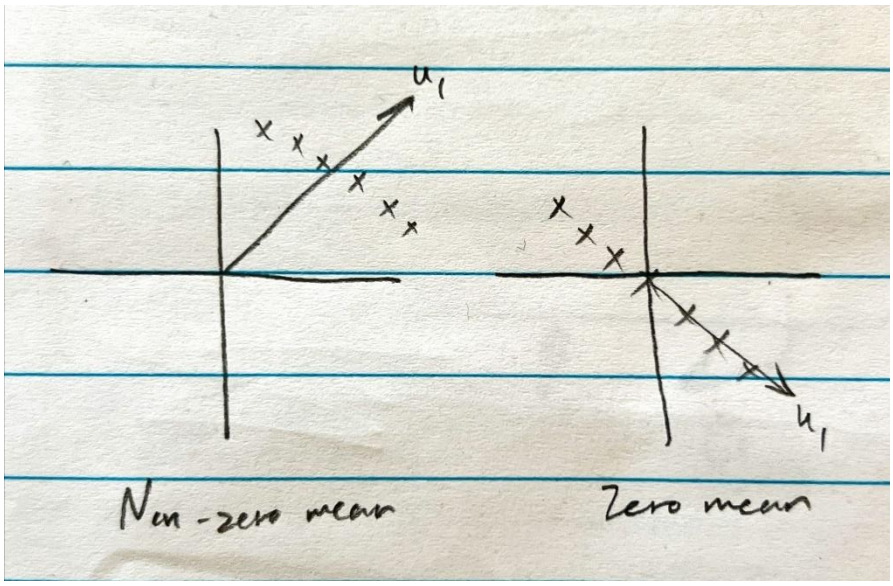
To reverse,

$$x = (UU^T)^{-1} U y$$

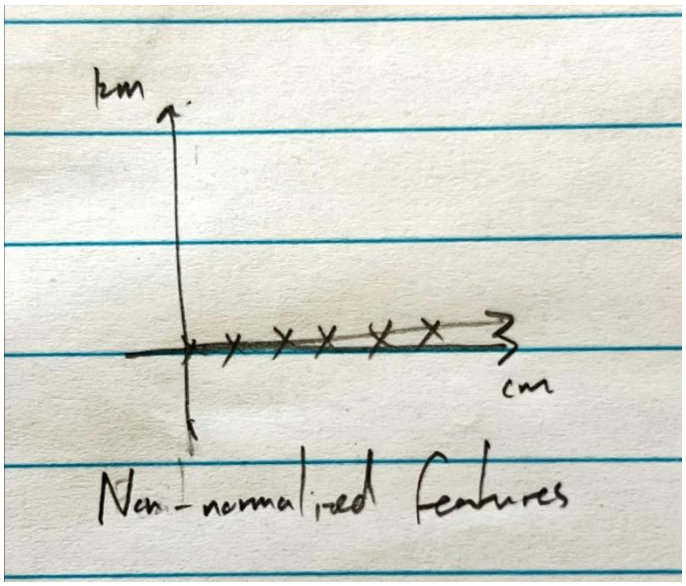
Ng says that $x \approx Uy$. Not sure how he arrives here, although it's only approximate cuz you lose information going from higher to lower dimensions.

Why do we remove the mean and standardize the variance?

If we don't remove the mean, then we can get a very poor principal axis:



If we don't standardize variance, then we can get skewed results when the feature magnitudes are very different (e.g. centimeters vs. kilometers):



9.1 When should you use PCA?

Good applications:

1. Visualization, e.g. projecting from nD to 2D or 3D
2. Compression for ML efficiency – most high-dimensional data lies on a low-dimensional subspace. You can reduce the computational load of your algorithm by reducing the input feature dimension.

Questionable applications:

1. Reduce overfitting (fails as often as it works). To reduce overfitting, it's better to use regularization.
2. Outlier detection and matching

Before using PCA, consider using the original data $x^{(i)}$.

10 Problem set 0 – linear algebra and multivariable calculus

10.1 Problem 1 – gradients and Hessians

Let A be an $n \times n$ symmetric matrix and x be a column vector of length n.

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

Let f be a function that maps an n-dimensional vector to a scalar, i.e. $f: \mathbb{R}^n \to \mathbb{R}$.

The gradient of f w.r.t. x is given by

$$\nabla_x(f(x)) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{bmatrix} \in \mathbb{R}^n$$

The Hessian of f w.r.t. x is the symmetric matrix of twice partial derivatives,

$$\nabla_x^2(f(x)) = \begin{bmatrix} \frac{\partial^2}{\partial^2 x_1} f(x) & \frac{\partial^2}{\partial x_1 \partial x_2} f(x) & \dots & \frac{\partial^2}{\partial x_1 \partial x_n} f(x) \\ \frac{\partial^2}{\partial x_2 \partial x_1} f(x) & \frac{\partial^2}{\partial^2 x_2} f(x) & \dots & \frac{\partial^2}{\partial x_2 \partial x_n} f(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1} f(x) & \frac{\partial^2}{\partial x_n \partial x_2} f(x) & \dots & \frac{\partial^2}{\partial^2 x_n} f(x) \end{bmatrix} \in \mathbb{R}^{n \times n}$$

In other words, the value of $\nabla_x^2(f(x))$ at row i and column j is $\frac{\partial^2}{\partial x_i \partial x_j} f(x)$.

The Hessian can be rewritten as

$$\nabla_x^2(f(x)) = \begin{bmatrix} \frac{\partial}{\partial x_1}\left(\frac{\partial}{\partial x_1}f(x)\right) & \frac{\partial}{\partial x_2}\left(\frac{\partial}{\partial x_1}f(x)\right) & \dots & \frac{\partial}{\partial x_n}\left(\frac{\partial}{\partial x_1}f(x)\right) \\ \frac{\partial}{\partial x_1}\left(\frac{\partial}{\partial x_2}f(x)\right) & \frac{\partial}{\partial x_2}\left(\frac{\partial}{\partial x_2}f(x)\right) & \dots & \frac{\partial}{\partial x_n}\left(\frac{\partial}{\partial x_2}f(x)\right) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_1}\left(\frac{\partial}{\partial x_n}f(x)\right) & \frac{\partial}{\partial x_2}\left(\frac{\partial}{\partial x_n}f(x)\right) & \dots & \frac{\partial}{\partial x_n}\left(\frac{\partial}{\partial x_n}f(x)\right) \end{bmatrix}$$

$$\nabla_x^2(f(x)) = \begin{bmatrix} \frac{\partial \nabla_x(f(x))}{\partial x_1} & \frac{\partial \nabla_x(f(x))}{\partial x_2} & \dots & \frac{\partial \nabla_x(f(x))}{\partial x_n} \end{bmatrix}$$

10.1.1 1a – gradient for matrices is like derivative for scalars

Let A be an $n \times n$ symmetric matrix and b, x be n-dimensional column vectors.

What does $\nabla_x\left(\frac{1}{2}x^T Ax + b^T x\right)$ evaluate to?

$$\nabla_x\left(\frac{1}{2}x^T Ax + b^T x\right) = Ax + b$$

This is like a matrix derivative. For scalars, this is equivalent to $\frac{d}{dx}\left(\frac{1}{2}ax^2 + bx\right) = ax + b$.

Let's find $\nabla_x(x^T Ax)$.

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$x^T A = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{n} x_k a_{k1} & \dots & \sum_{k=1}^{n} x_k a_{kn} \end{bmatrix}$$

$$x^T Ax = \begin{bmatrix} \sum_{k=1}^{n} x_k a_{k1} & \dots & \sum_{k=1}^{n} x_k a_{kn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \left(x_1 \sum_{k=1}^{n} x_k a_{k1}\right) + \dots + \left(x_n \sum_{k=1}^{n} x_k a_{kn}\right)$$

$$= \sum_{l=1}^{n}\left(x_l \sum_{k=1}^{n} x_k a_{kl}\right)$$

Recall that $\nabla_x(f(x)) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$.

When we take partial derivative $\frac{\partial}{\partial x_j}(x^T Ax) = \frac{\partial}{\partial x_j}(\sum_{l=1}^{n}(x_l \sum_{k=1}^{n} x_k a_{kl}))$, we only care about three scenarios:

1. $l = j, k \neq j$
2. $l \neq j, k = j$
3. $l = j, k = j$

Plug these in:

$$\sum_{l=1}^{n}\left(x_l \sum_{k=1}^{n} x_k a_{kl}\right) = x_j \sum_{k\neq j} x_k a_{kj} + x_j \sum_{l\neq j} x_l\, a_{jl} + a_{jj}x_j^2$$

Since A is symmetric, the first two terms are identical, and we have

$$2x_j \sum_{k\neq j} x_k a_{kj} + a_{jj}x_j^2$$

Then

$$\frac{\partial}{\partial x_j}\left(2x_j \sum_{k\neq j} x_k a_{kj} + a_{jj}x_j^2\right) = 2\sum_{k\neq j} x_k a_{kj} + 2a_{jj}x_j = 2\sum_{k=1}^{n} a_{kj}x_k = 2\sum_{k=1}^{n} a_{jk}x_k$$

$$\nabla_x(x^T Ax) = \begin{bmatrix} 2\sum_{k=1}^{n} a_{1k}x_k \\ \vdots \\ 2\sum_{k=1}^{n} a_{nk}x_k \end{bmatrix} = 2Ax$$

Now let's find $\nabla_x(b^T x)$.

$$b^T x = \sum_{k=1}^{n} b_k x_k$$

$$\frac{\partial}{\partial x_j}(b^T x) = b_j$$

$$\nabla_x(b^T x) = \begin{bmatrix} \frac{\partial}{\partial x_1}(b^T x) \\ \vdots \\ \frac{\partial}{\partial x_n}(b^T x) \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = b$$

10.1.2 1b – chain rule for matrices

Let g be a differentiable function $g: \mathbb{R} \to \mathbb{R}$ and h be a differentiable function $h: \mathbb{R}^n \to \mathbb{R}$. Let $f(x) = g\big(h(x)\big)$. Let x be an n-dimensional column vector. What does $\nabla_x\big(f(x)\big)$ evaluate to?

$$\nabla_x\big(f(x)\big) = \nabla_x\big(h(x)\big)g'\big(h(x)\big)$$

This is like the chain rule for matrices. For scalars, this is equivalent to $\frac{d}{dx}\Big(g\big(h(x)\big)\Big) = g'\big(h(x)\big)h'(x) = \frac{dg}{dh}\frac{dh}{dx}$.

Derivation.

$$\nabla_x\big(f(x)\big) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

$$\frac{\partial}{\partial x_j} f(x) = \frac{\partial}{\partial x_j} g\big(h(x)\big) = g'\big(h(x)\big)\frac{\partial h}{\partial x_j} = \frac{dg}{dh}\frac{\partial h}{\partial x_j}$$

$$\nabla_x\big(f(x)\big) = \begin{bmatrix} \frac{\partial g}{\partial h}\frac{\partial h}{\partial x_1} \\ \vdots \\ \frac{\partial g}{\partial h}\frac{\partial h}{\partial x_n} \end{bmatrix} = \nabla_x\big(h(x)\big)\frac{dg}{dh} = \nabla_x\big(h(x)\big)g'\big(h(x)\big)$$

10.1.3 1c – Hessian for matrices is like second derivative for scalars

Let $f(x) = \frac{1}{2}x^T Ax + b^T x$, where A is a symmetric $n \times n$ matrix and b, x are n-dimensional column vectors. $f: \mathbb{R}^n \to \mathbb{R}$.

What does $\nabla_x^2\big(f(x)\big)$ evaluate to?

$$\nabla_x^2\big(f(x)\big) = A$$

This is like second derivative for matrices. For scalars, this is equivalent to $\frac{d^2}{dx^2}\left(\frac{1}{2}ax^2 + bx\right) = a$.

We know that

$$\nabla_x^2(f(x)) = \begin{bmatrix} \frac{\partial \nabla_x(f(x))}{\partial x_1} & \frac{\partial \nabla_x(f(x))}{\partial x_2} & \dots & \frac{\partial \nabla_x(f(x))}{\partial x_n} \end{bmatrix}$$

And from 1A, we know that

$$\nabla_x\left(\frac{1}{2}x^T Ax + b^T x\right) = Ax + b \in \mathbb{R}^n$$

Then

$$\nabla_x^2(f(x)) = \begin{bmatrix} \frac{\partial (Ax+b)}{\partial x_1} & \frac{\partial (Ax+b)}{\partial x_2} & \dots & \frac{\partial (Ax+b)}{\partial x_n} \end{bmatrix}$$

$$Ax + b = \begin{bmatrix} \sum_{k=1}^{n} a_{1k}x_k + b_1 \\ \vdots \\ \sum_{k=1}^{n} a_{nk}x_k + b_n \end{bmatrix}$$

$$\frac{\partial (Ax+b)}{\partial x_j} = \begin{bmatrix} a_{1j} \\ \vdots \\ a_{nj} \end{bmatrix}$$

$$\nabla_x^2(f(x)) = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \vdots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} = A$$

10.1.4 1d – chain rule for matrices times two

Let $f(x) = g(a^T x)$ where g: $\mathbb{R} \to \mathbb{R}$ is continuously differentiable and a, x are n-dimensional column vectors. What do $\nabla_x(f(x))$ and $\nabla_x^2(f(x))$ evaluate to?

$$\nabla_x(f(x)) = g'(a^T x) * a$$

$$\nabla_x^2(f(x)) = g''(a^T x)aa^T$$

This is like using the chain rule twice for matrices. For scalars, this is equivalent to $\frac{d^2}{dx^2}(g(ax)) = a^2 g''(ax)$.

From 1b, if $g: \mathbb{R} \to \mathbb{R}$ and $h: \mathbb{R}^n \to \mathbb{R}$ are continuously differentiable functions, then $\nabla_x \big(g\big(h(x)\big)\big) = \nabla_x\big(h(x)\big)g'\big(h(x)\big)$.

In this case, $h(x) = a^T x$, so

$$\nabla_x\big(f(x)\big) = \nabla_x\big(g(a^T x)\big) = \nabla_x(a^T x)g'(a^T x)$$

From 1a, we know $\nabla_x(a^T x) = a$. Then

$$\nabla_x\big(f(x)\big) = a * g'(a^T x)$$

We know

$$\nabla_x^2\big(f(x)\big) = \left[\frac{\partial \nabla_x\big(f(x)\big)}{\partial x_1} \quad \frac{\partial \nabla_x\big(f(x)\big)}{\partial x_2} \quad \dots \quad \frac{\partial \nabla_x\big(f(x)\big)}{\partial x_n}\right]$$

In this case,

$$\begin{aligned}
\nabla_x^2\big(g(a^T x)\big) &= \left[\frac{\partial \nabla_x\big(g(a^T x)\big)}{\partial x_1} \quad \frac{\partial \nabla_x\big(g(a^T x)\big)}{\partial x_2} \quad \dots \quad \frac{\partial \nabla_x\big(g(a^T x)\big)}{\partial x_n}\right] \\
&= \left[\frac{\partial (g'(a^T x)a)}{\partial x_1} \quad \frac{\partial (g'(a^T x)a)}{\partial x_2} \quad \dots \quad \frac{\partial (g'(a^T x)a)}{\partial x_n}\right] \\
&= a\left[\frac{\partial \big(g'(a^T x)\big)}{\partial x_1} \quad \frac{\partial \big(g'(a^T x)\big)}{\partial x_2} \quad \dots \quad \frac{\partial \big(g'(a^T x)\big)}{\partial x_n}\right] \\
&= a\left[g''(a^T x)\frac{\partial h}{\partial x_1} \quad g''(a^T x)\frac{\partial h}{\partial x_2} \quad \dots \quad g''(a^T x)\frac{\partial h}{\partial x_n}\right] \\
&= g''(a^T x)a\left[\frac{\partial h}{\partial x_1} \quad \frac{\partial h}{\partial x_2} \quad \dots \quad \frac{\partial h}{\partial x_n}\right] = g''(a^T x)a(\nabla_x\big(h(x)\big)^T = g''(a^T x)aa^T
\end{aligned}$$

Another derivation:

We know that the Hessian matrix consists of twice partial derivatives of $f(x)$. We can evaluate each element in the Hessian as

$$\frac{\partial^2}{\partial x_i \partial x_j}\big(f(x)\big) = \frac{\partial^2}{\partial x_i \partial x_j}\Big(g\big(h(x)\big)\Big) = \frac{\partial^2 g}{(\partial h)^2}\frac{\partial h}{\partial x_i}\frac{\partial h}{\partial x_j}$$

In this case, $h = a^T x$, so

$$\frac{\partial^2 g}{(\partial h)^2}\frac{\partial h}{\partial x_i}\frac{\partial h}{\partial x_j} = g''(a^T x)a_i a_j$$

Then

$$\nabla_x^2\big(f(x)\big) = g''(a^T x)\begin{bmatrix} a_1^2 & a_1a_2 & \dots & a_1a_n \\ a_2a_1 & a_2^2 & \dots & a_2a_n \\ \vdots & \vdots & \ddots & \vdots \\ a_na_1 & a_na_2 & \dots & a_n^2 \end{bmatrix} = g''(a^T x)aa^T$$

10.2 Problem 2 – positive definite matrices

An $n \times n$ symmetric matrix A is positive semidefinite, denoted $A \geq 0$, if $x^T Ax \geq 0$ for all n-dimensional vectors x.

An $n \times n$ symmetric matrix A is positive definite, denoted $A > 0$, if $x^T Ax > 0$ for all non-zero n-dimensional vectors x.

Symmetric means $A = A^T$.

The simplest example of a positive definite matrix is the identity I:

$$x^T Ix = x^T x = \|x\|_2^2 = \sum_{i=1}^{n} x_i^2$$

$\|x\|_2$ is called the L2 norm (L^2 norm) of x.

10.2.1 2a

If z is an n-dimensional column vector, show that $A = zz^T$ is positive semidefinite.

$$A^T = (zz^T)^T = zz^T = A$$

$$x^T Ax = x^T zz^T x = (z^T x)^T z^T x = y^T y = \|y\|^2 \geq 0$$

10.2.2 2b

Let z be a non-zero n-dimensional column vector, and $A = zz^T$. What is the null space of A? What is the rank of A?

$$N(A) = \{x \in \mathbb{R}^n : x^T z = 0\}$$

$$R(A) = 1$$

$N(A)$ is the subspace of $\mathbb{R}^n$ that satisfies the equation $Ax = \mathbf{0}$.

$$zz^T x = \mathbf{0}$$

If z is non-zero, then the above statement is true only when $z^T x = 0$, i.e.

$$N(A) = \{x \in \mathbb{R}^n : x^T z = 0\}$$

In words, the nullspace of A is all n-dimensional vectors x that are orthogonal to z. What is the basis for this nullspace?

$$x^T z = \sum_k x_k z_k = x_1 z_1 + x_2 z_2 + \cdots + x_n z_n$$

Leave x_1 alone. Let $x_2 = 1$ and all others be 0. Then

$$x_1 z_1 + z_2 = 0 \rightarrow x_1 = -\frac{z_2}{z_1}$$

So one basis vector is

$$\begin{bmatrix} -\frac{z_2}{z_1} \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Repeat for all x_j to get $n - 1$ basis vectors, e.g. for x_3 and x_n, we have

$$\begin{bmatrix} -\frac{z_3}{z_1} \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} -\frac{z_n}{z_1} \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

The rank of A is the number of linearly independent columns of A, which is equivalent to the dimension of the column space of A. The dimensions of the column space and nullspace of A add up to n. Since the dimension of the nullspace is $n - 1$, we have $R(A) = 1$.

10.2.3 2c

Let $A \in \mathbb{R}^{n \times n}$ be positive semidefinite and $B \in \mathbb{R}^{m \times n}$ be an arbitrary $m \times n$ matrix. Is BAB^T positive semidefinite? Yes.

Since A is positive semidefinite, we know that $x^T Ax \geq 0$.

$$x^T BAB^T x = (B^T x)^T A(B^T x) = y^T Ay \geq 0$$

10.3 Problem 3 – eigenvectors, eigenvalues, and the spectral theorem

The eigenvalues, λ, for an $n \times n$ matrix A are the roots of the characteristic polynomial $p_A(\lambda) = \det(\lambda I - A) = |\lambda I - A|$. In general, eigenvalues may be complex.

An equivalent definition for eigenvalues: they are the values $\lambda \in \mathbb{C}$ for which there exists a non-zero vector $x \in \mathbb{C}^n$ s.t. $Ax = \lambda x$. (x, λ) denotes an eigenvector/eigenvalue pair.

$$Ax = \lambda x \rightarrow Ax - \lambda x = 0 \rightarrow (A - I\lambda)x = 0$$

Since x is non-zero, $A - I\lambda$ has a nontrivial nullspace and is singular. This means $|A - I\lambda| = 0$.

Let $\text{diag}(\lambda_1, \dots, \lambda_n)$ be a diagonal matrix with elements $\lambda_1, \dots, \lambda_n$:

$$\text{diag}(\lambda_1, \dots, \lambda_n) = \begin{bmatrix} \lambda_1 & 0 & 0 & \dots & 0 \\ 0 & \lambda_2 & 0 & \dots & 0 \\ 0 & 0 & \lambda_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \lambda_n \end{bmatrix}$$

10.3.1 3a – diagonalization of a matrix

Suppose $A \in \mathbb{R}^{n \times n}$ is diagonalizable, i.e. $A = T\Lambda T^{-1}$ where $T \in \mathbb{R}^{n \times n}$ is invertible and $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ is diagonal. Let $t^{(i)} \in \mathbb{R}^n$ denote the columns of T so that $T = [t^{(1)} \quad \dots \quad t^{(n)}]$.

Show that $At^{(i)} = \lambda_i t^{(i)}$ so that the eigenvector/eigenvalue pairs of A are $\left(t^{(i)}, \lambda_i\right)$.

$$A = T\Lambda T^{-1}$$

$$AT = T\Lambda$$

$$A[t^{(1)} \quad \dots \quad t^{(n)}] = [t^{(1)} \quad \dots \quad t^{(n)}]\Lambda$$

$$[At^{(1)} \quad \dots \quad At^{(n)}] = [\lambda_1 t^{(1)} \quad \dots \quad \lambda_n t^{(n)}]$$

$$At^{(i)} = \lambda_i t^{(i)}$$

10.3.2 3b – diagonalization of a symmetric matrix

A matrix $U \in \mathbb{R}^{n \times n}$ is orthogonal if $U^T U = I$. The spectral theorem, one of the most important theorems in linear algebra, states that if $A \in \mathbb{R}^{n \times n}$ is symmetric, then A is diagonalizable by a real orthogonal matrix. That is, there is a diagonal matrix $\Lambda \in \mathbb{R}^{n \times n}$ and an orthogonal matrix $U \in \mathbb{R}^{n \times n}$ s.t.

$$U^T A U = \Lambda$$

$$A = U\Lambda U^T$$

Let A be symmetric. Let $u^{(i)} \in \mathbb{R}^n$ denote the column vectors of U s.t. $U = [u^{(1)} \quad \dots \quad u^{(n)}]$. Show that the eigenvector/eigenvalue pairs of A are $\left(u^{(i)}, \lambda_i\right)$.

$$A = U\Lambda U^T$$

$$AU = U\Lambda$$

$$A[u^{(1)} \quad \dots \quad u^{(n)}] = [u^{(1)} \quad \dots \quad u^{(n)}]\Lambda$$

$$[Au^{(1)} \quad \dots \quad Au^{(n)}] = [\lambda_1 u^{(1)} \quad \dots \quad \lambda_n u^{(n)}]$$

$$Au^{(i)} = \lambda_i u^{(i)}$$

10.3.3 3c – eigenvalues of a positive semidefinite matrix are ≥ 0

Let A be symmetric. Show that if A is positive semidefinite, then $\lambda_i(A) \geq 0$ for all i, where $\lambda_i(A)$ denotes the eigenvalues of A.

From 3b, we have

$$Au^{(i)} = \lambda_i u^{(i)}$$

$${u^{(i)}}^T Au^{(i)} = {u^{(i)}}^T \lambda_i u^{(i)}$$

Note that ${u^{(i)}}^T u^{(i)} = 1$ (proof below). Then

$${u^{(i)}}^T Au^{(i)} = \lambda_i$$

By the definition of positive semidefinite, ${u^{(i)}}^T Au^{(i)} \geq 0$, which means $\lambda_i \geq 0$.

$$U^T U = I$$

$$\begin{bmatrix} {u^{(1)}}^T \\ \vdots \\ {u^{(n)}}^T \end{bmatrix} [u^{(1)} \quad \dots \quad u^{(n)}] = \begin{bmatrix} {u^{(1)}}^T u^{(1)} & \dots & {u^{(1)}}^T u^{(n)} \\ \vdots & \ddots & \vdots \\ {u^{(n)}}^T u^{(1)} & \dots & {u^{(n)}}^T u^{(n)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore,

$${u^{(r)}}^T u^{(c)} = \begin{cases} 1, r = c \\ 0, r \neq c \end{cases}$$

$$U^T AU = \Lambda$$

$$\begin{bmatrix} u^{(1)^T} \\ \vdots \\ u^{(n)^T} \end{bmatrix} [a^{(1)} \quad \dots \quad a^{(n)}][u^{(1)} \quad \dots \quad u^{(n)}] = \begin{bmatrix} u^{(1)^T} A \\ \vdots \\ u^{(n)^T} A \end{bmatrix} [u^{(1)} \quad \dots \quad u^{(n)}]$$

$$= \begin{bmatrix} u^{(1)^T} A u^{(1)} & \dots & u^{(1)^T} A u^{(n)} \\ \vdots & \ddots & \vdots \\ u^{(n)^T} A u^{(1)} & \dots & u^{(n)^T} A u^{(n)} \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \lambda_n \end{bmatrix}$$

Again, you have $\lambda_i = u^{(i)^T} A u^{(i)} \geq 0$.

Can you prove that $u^{(r)^T} A u^{(c)} = 0$ for $r \neq c$?

11 Problem set 1 – supervised learning

11.1 Problem 1 – linear classifiers (logistic regression and GDA)

Two probabilistic linear classifiers –

1. A discriminative linear classifier: logistic regression
2. A generative linear classifier: Gaussian discriminant analysis (GDA)

Both find a linear decision boundary that separates the data into two classes, but they make different assumptions.

11.1.1 1a – loss function for logistic regression is convex (has only one minimum)

The empirical loss for logistic regression is

$$J(\theta) = -\frac{1}{m}\ell(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log h_\theta\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right]$$

Where $y^{(i)} \in \{0,1\}, h_\theta(x) = g(\theta^T x)$, and $g(z) = \frac{1}{1+e^{-z}}$.

Find the Hessian H of $J(\theta)$ and show that for any vector z,

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

This means that H is positive semidefinite, i.e. $H \geq 0$, which implies that J is convex and has no local minima other than the global one. This is analogous to a twice-differentiable scalar convex function, which is convex if and only if its second derivative is nonnegative on its entire domain.

For example, let $y(x) = \frac{1}{2}ax^2 + bx + c$, where $a > 0$. We know this is a quadratic function that faces up, a classic example of a strictly convex function. Then

$$\frac{dy}{dx} = ax + b$$

$$\frac{d^2y}{dx^2} = a > 0$$

Proof. From the notes on logistic regression, we have

$$\frac{\partial \ell(\theta)}{\partial \theta_j} = \sum_{i=1}^{m} \left(y^{(i)} - g\left(\theta^T x^{(i)}\right)\right) x_j^{(i)}$$

$$g'(z) = g(z)\left(1 - g(z)\right)$$

Take another partial derivative of $\ell(\theta)$:

$$\frac{\partial}{\partial \theta_k}\left(\frac{\partial \ell(\theta)}{\partial \theta_j}\right) = \frac{\partial}{\partial \theta_k}\left(\sum_{i=1}^{m} \left(y^{(i)} - g\left(\theta^T x^{(i)}\right)\right) x_j^{(i)}\right) = \sum_{i=1}^{m} \frac{\partial}{\partial \theta_k}\left(\left(y^{(i)} - g\left(\theta^T x^{(i)}\right)\right) x_j^{(i)}\right)$$

$$= -\sum_{i=1}^{m} x_j^{(i)} \frac{\partial}{\partial \theta_k}\left(g\left(\theta^T x^{(i)}\right)\right) = -\sum_{i=1}^{m} x_j^{(i)} g\left(\theta^T x^{(i)}\right)\left(1 - g\left(\theta^T x^{(i)}\right)\right) x_k^{(i)}$$

$$\frac{\partial^2}{\partial \theta_j \partial \theta_k} \ell(\theta) = -\sum_{i=1}^{m} x_j^{(i)} x_k^{(i)} g\left(\theta^T x^{(i)}\right)\left(1 - g\left(\theta^T x^{(i)}\right)\right)$$

Then each element in the Hessian is given by

$$H_{jk} = \frac{\partial^2}{\partial \theta_j \partial \theta_k} J(\theta) = \frac{\partial^2}{\partial \theta_j \partial \theta_k}\left(-\frac{1}{m}\ell(\theta)\right) = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)} x_k^{(i)} g\left(\theta^T x^{(i)}\right)\left(1 - g\left(\theta^T x^{(i)}\right)\right)$$

Evaluate $z^T H z$:

$$[z_1 \quad \dots \quad z_n]H\begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix}$$

$$= [(z_1H_{11} + z_2H_{21} + \cdots + z_nH_{n1}) \quad (z_1H_{12} + z_2H_{22} + \cdots + z_nH_{n2}) \quad \dots \quad (z_1H_{1n} + z_2H_{2n} + \cdots + z_nH_{nn})]\begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix}$$

$$= \left[\sum_j z_jH_{j1} \quad \sum_j z_jH_{j2} \quad \dots \quad \sum_j z_jH_{jn}\right]\begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} = z_1\sum_j z_jH_{j1} + z_2\sum_j z_jH_{j2} + \cdots + z_n\sum_j z_jH_{j2}$$

$$= \sum_k \left(z_k \sum_j z_jH_{jk}\right) = \sum_k\sum_j z_jz_kH_{jk} = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{n}\sum_{j=1}^{n} g\left(\theta^T x^{(i)}\right)\left(1 - g\left(\theta^T x^{(i)}\right)\right)x_j^{(i)}z_jx_k^{(i)}z_k$$

$$= \frac{1}{m}\sum_{i=1}^{m} g\left(\theta^T x^{(i)}\right)\left(1 - g\left(\theta^T x^{(i)}\right)\right)\sum_{k=1}^{n} x_k^{(i)}z_k\sum_{j=1}^{n} x_j^{(i)}z_j$$

$$= \frac{1}{m}\sum_{i=1}^{m} g\left(\theta^T x^{(i)}\right)\left(1 - g\left(\theta^T x^{(i)}\right)\right)\left(x^{(i)^T}z\right)^2$$

Since $g\left(\theta^T x^{(i)}\right)\left(1 - g\left(\theta^T x^{(i)}\right)\right) \geq 0$ and $\left(x^{(i)^T}z\right)^2 \geq 0$, we have $z^THz \geq 0$.

11.1.2 1c – GDA logistic function proof

From the GDA assumptions below, prove that GDA results in a linear decision boundary.

Assume that we have already fit the parameters of our model $\phi, \mu_0, \mu_1, \Sigma$.

$$P(x|y=0) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right)$$

$$P(x|y=1) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right)$$

$$P(y) = \phi^y(1-\phi)^{1-y}$$

By Bayes,

$$P(y=1|x) = \frac{P(x|y=1)P(y=1)}{P(x)} = \frac{P(x|y=1)P(y=1)}{P(x|y=1)P(y=1) + P(x|y=0)P(y=0)}$$
$$= \frac{1}{1 + \frac{P(x|y=0)P(y=0)}{P(x|y=1)P(y=1)}}$$

$$\frac{P(x|y=0)P(y=0)}{P(x|y=1)P(y=1)} = \frac{\frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right)}{\frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right)} * \frac{1-\phi}{\phi}$$

$$= \frac{1-\phi}{\phi}\exp\left(\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1) - \frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right)$$

$$= \frac{1-\phi}{\phi}\exp\left(\frac{1}{2}\left(x^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu_1 - \mu_1^T\Sigma^{-1}x + \mu_1^T\Sigma^{-1}\mu_1\right.\right.$$

$$\left.\left. - (x^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu_0 - \mu_0^T\Sigma^{-1}x + \mu_0^T\Sigma^{-1}\mu_0)\right)\right)$$

Since the argument to exp is a real number and Σ^{-1} is symmetric, we have

$$\frac{1-\phi}{\phi}\exp\left(\frac{1}{2}(2\mu_0^T\Sigma^{-1}x - 2\mu_1^T\Sigma^{-1}x + \mu_1^T\Sigma^{-1}\mu_1 - \mu_0^T\Sigma^{-1}\mu_0)\right)$$

$$= \exp\left((\mu_0-\mu_1)^T\Sigma^{-1}x + \frac{1}{2}(\mu_1+\mu_0)^T\Sigma^{-1}(\mu_1-\mu_0) + \log\frac{1-\phi}{\phi}\right)$$

$$= \exp\left(-\left(\left(\Sigma^{-1}(\mu_1-\mu_0)\right)^T x + \frac{1}{2}(\mu_0+\mu_1)^T\Sigma^{-1}(\mu_0-\mu_1) - \log\frac{1-\phi}{\phi}\right)\right)$$

Then

$$P(y=1|x) = \frac{1}{1+\exp\left(-(\theta^T x + \theta_0)\right)}$$

$$\theta = \Sigma^{-1}(\mu_1-\mu_0)$$

$$\theta_0 = \frac{1}{2}(\mu_0+\mu_1)^T\Sigma^{-1}(\mu_0-\mu_1) - \log\frac{1-\phi}{\phi}$$

$P(y=1|x)$ is the logistic function and the decision boundary, $\theta^T x$, is linear.

11.1.3 1d – GDA MLE parameters

For this problem, assume the dimension of x is 1 so that the GDA assumptions are given by

$$\Sigma = |\Sigma| = \sigma^2$$

$$P(x|y=0) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{1}{2}\frac{(x-\mu_0)^2}{\sigma^2}\right)$$

$$P(x|y=1) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{1}{2}\frac{(x-\mu_1)^2}{\sigma^2}\right)$$

$$P(y) = \phi^y (1-\phi)^{1-y}$$

The log-likelihood of the data is

$$\ell(\phi, \mu_0, \mu_1, \Sigma) = \log \prod_{i=1}^{m} P\left(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma\right) = \log \prod_{i=1}^{m} P\left(x^{(i)} \middle| y^{(i)}; \mu_0, \mu_1, \Sigma\right) P\left(y^{(i)}; \phi\right)$$

Find the MLE of the parameters $\phi, \mu_0, \mu_1, \Sigma$.

We can write $P\left(x^{(i)} \middle| y^{(i)}; \mu_0, \mu_1, \Sigma\right)$ as

$$\left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{\left(x^{(i)} - \mu_1\right)^2}{\sigma^2}\right)\right)^{y^{(i)}} \left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{\left(x^{(i)} - \mu_0\right)^2}{\sigma^2}\right)\right)^{1-y^{(i)}}$$

Then ℓ is

$$\sum_{i=1}^{m} y^{(i)} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{\left(x^{(i)} - \mu_1\right)^2}{\sigma^2}\right)\right) + y^{(i)} \log\phi$$
$$+ \left(1 - y^{(i)}\right) \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{\left(x^{(i)} - \mu_0\right)^2}{\sigma^2}\right)\right) + \left(1 - y^{(i)}\right) \log(1-\phi)$$

Now take partial derivatives.

$$\frac{\partial \ell}{\partial \phi} = \sum_{i=1}^{m} \frac{y^{(i)}}{\phi} + \frac{y^{(i)} - 1}{1-\phi} = \sum_{i=1}^{m} y^{(i)}(1-\phi) + \left(y^{(i)} - 1\right)\phi = \sum_{i=1}^{m} \left(y^{(i)} - \phi\right) = 0 \rightarrow$$

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

$$\phi = \frac{1}{m}\sum_{i=1}^{m} \mathcal{I}\left(y^{(i)} = 1\right)$$

$$\frac{\partial \ell}{\partial \mu_0} = \frac{\partial}{\partial \mu_0}\left(\sum_{i=1}^{m}\left(1-y^{(i)}\right)\left(-\frac{1}{2}\frac{\left(x^{(i)}-\mu_0\right)^2}{\sigma^2}\right)\right) = \sum_{i=1}^{m}\frac{\left(1-y^{(i)}\right)}{\sigma^2}\left(x^{(i)}-\mu_0\right)$$

$$= \frac{1}{\sigma^2}\sum_{i=1}^{m}\left(1-y^{(i)}\right)\left(x^{(i)}-\mu_0\right) = \frac{1}{\sigma^2}\sum_{i=1}^{m}\mathcal{I}\left(y^{(i)}=0\right)\left(x^{(i)}-\mu_0\right)$$

$$= \frac{1}{\sigma^2}\sum_{i=1}^{m}\mathcal{I}\left(y^{(i)}=0\right)\left(x^{(i)}-\mu_0\right) = \frac{1}{\sigma^2}\sum_{i=1}^{m}\left(\mathcal{I}\left(y^{(i)}=0\right)x^{(i)}-\mathcal{I}\left(y^{(i)}=0\right)\mu_0\right) = 0$$

$$\rightarrow \sum_{i=1}^{m}\mathcal{I}\left(y^{(i)}=0\right)x^{(i)} = \mu_0\sum_{i=1}^{m}\mathcal{I}\left(y^{(i)}=0\right) \rightarrow \mu_0 = \frac{\sum_{i=1}^{m}\mathcal{I}\left(y^{(i)}=0\right)x^{(i)}}{\sum_{i=1}^{m}\mathcal{I}\left(y^{(i)}=0\right)}$$

$$\mu_1 = \frac{\sum_{i=1}^{m}\mathcal{I}\left(y^{(i)}=1\right)x^{(i)}}{\sum_{i=1}^{m}\mathcal{I}\left(y^{(i)}=1\right)}$$

$$\frac{\partial \ell}{\partial \Sigma} = \frac{\partial}{\partial \Sigma}\left(\sum_{i=1}^{m} y^{(i)}\log\left(\frac{1}{\sqrt{2\pi\Sigma}}\exp\left(-\frac{1}{2}\frac{\left(x^{(i)}-\mu_1\right)^2}{\Sigma}\right)\right)\right.$$

$$\left. + \left(1-y^{(i)}\right)\log\left(\frac{1}{\sqrt{2\pi\Sigma}}\exp\left(-\frac{1}{2}\frac{\left(x^{(i)}-\mu_0\right)^2}{\Sigma}\right)\right)\right)$$

$$= \frac{\partial}{\partial \Sigma}\left(\sum_{i=1}^{m} y^{(i)}\log\left(\exp\left(-\frac{1}{2}\frac{\left(x^{(i)}-\mu_1\right)^2}{\Sigma}-\frac{1}{2}\log 2\pi\Sigma\right)\right)\right.$$

$$\left. + \left(1-y^{(i)}\right)\log\left(\exp\left(-\frac{1}{2}\frac{\left(x^{(i)}-\mu_0\right)^2}{\Sigma}-\frac{1}{2}\log 2\pi\Sigma\right)\right)\right)$$

$$= \frac{\partial}{\partial \Sigma}\left(\sum_{i=1}^{m} y^{(i)}\left(-\frac{\left(x^{(i)}-\mu_1\right)^2}{2}\Sigma^{-1}-\frac{1}{2}\log 2\pi\Sigma\right)\right.$$

$$\left. + \left(1-y^{(i)}\right)\left(-\frac{\left(x^{(i)}-\mu_0\right)^2}{2}\Sigma^{-1}-\frac{1}{2}\log 2\pi\Sigma\right)\right)$$

$$= \sum_{i=1}^{m} y^{(i)}\left(\frac{\left(x^{(i)}-\mu_1\right)^2}{2}\Sigma^{-2}-\frac{\Sigma^{-1}}{2}\right) + \left(1-y^{(i)}\right)\left(\frac{\left(x^{(i)}-\mu_0\right)^2}{2}\Sigma^{-2}-\frac{\Sigma^{-1}}{2}\right) = 0$$

$$\rightarrow \sum_{i=1}^{m} y^{(i)}\left(\frac{\left(x^{(i)}-\mu_1\right)^2}{2}-\frac{\Sigma}{2}\right) + \left(1-y^{(i)}\right)\left(\frac{\left(x^{(i)}-\mu_0\right)^2}{2}-\frac{\Sigma}{2}\right) = 0$$

$$\rightarrow \sum_{i=1}^{m} y^{(i)}\left(x^{(i)}-\mu_1\right)^2 + \left(1-y^{(i)}\right)\left(x^{(i)}-\mu_0\right)^2 - \Sigma = 0 \rightarrow$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m} y^{(i)}\left(x^{(i)}-\mu_1\right)^2 + \left(1-y^{(i)}\right)\left(x^{(i)}-\mu_0\right)^2 \rightarrow$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \mu_{y^{(i)}}\right)^2 = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \mu_{y^{(i)}}\right)\left(x^{(i)} - \mu_{y^{(i)}}\right)^T$$

11.2 Problem 2 – incomplete, positive-only labels

In this scenario, we have labels only for a subset of the positive examples/data. The rest of the examples are unlabeled. Each example is

$$\left\{\left(x^{(i)}, t^{(i)}, y^{(i)}\right)\right\}_{i=1}^{m}$$

$t^{(i)} \in \{0,1\}$ is the "true" label. A positive example means $t^{(i)} = 1$, and a negative example means $t^{(i)} = 0$.

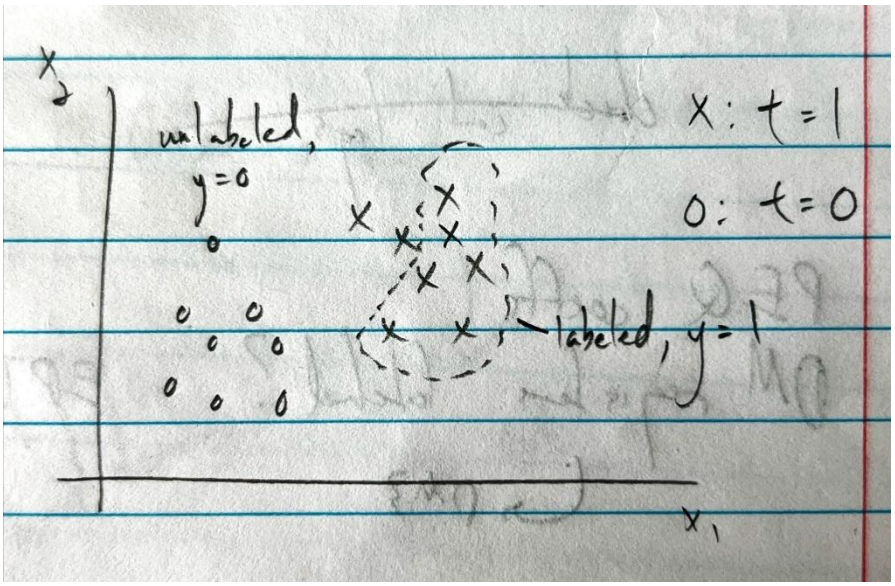
All labeled examples are positive, that is,

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

$$P\left(t^{(i)} = 1 \middle| y^{(i)} = 1\right) = 1$$

For a real world application, see the problem set.

Graphically,



Let's call the labeled examples the database. If the database examples are selected uniformly at random from the positive examples, then equivalently, we can say that $y^{(i)}$ and $x^{(i)}$ are conditionally independent given $t^{(i)}$:

$$P\left(y^{(i)} = 1 \middle| t^{(i)} = 1, x^{(i)}\right) = P\left(y^{(i)} = 1 \middle| t^{(i)} = 1\right)$$

Then

$$P\left(t^{(i)} = 1 \middle| x^{(i)}\right) = \frac{P\left(y^{(i)} = 1 \middle| x^{(i)}\right)}{\alpha}$$

$$\alpha = P\left(y^{(i)} = 1 \middle| t^{(i)} = 1\right)$$

α is a constant $\in \mathbb{R}$. Under the uniform selection assumption, α should be independent of i (it should be the ratio of number of examples in the database to number of positive examples). The probability that an example is in the database is equal to the probability that the example is positive and was uniformly and randomly selected to be in the database.

Let's say you train a classifier $h\left(x^{(i)}\right) = P\left(y^{(i)} = 1 \middle| x^{(i)}\right)$ on the y labels. If you look only at the database examples (where $y^{(i)} = 1$), you can approximately say

$$P\left(t^{(i)} = 1 \middle| x^{(i)}\right) \approx 1 = \frac{P\left(y^{(i)} = 1 \middle| x^{(i)}\right)}{\alpha} = \frac{h\left(x^{(i)}\right)}{\alpha} \rightarrow h\left(x^{(i)}\right) \approx \alpha$$

Therefore, you can estimate α by taking the average of $h\left(x^{(i)}\right)$ across the database examples.

With this estimate for the correction factor α, you can offset $h\left(x^{(i)}\right)$ to estimate $P\left(t^{(i)} = 1 \middle| x^{(i)}\right)$. For logistic regression,

$$\frac{h\left(x^{(i)}\right)}{\alpha} = \frac{1}{\alpha}\frac{1}{1 + e^{-\theta^T x}}$$

The decision boundary is then

$$\alpha\left(1 + e^{-\theta^T x}\right) = 2$$

If θ and x are 3-dimensional, and if we freely choose x_1, then the expression for x_2 is

$$x_2 = -\frac{\left(\ln\left(\frac{2}{\alpha} - 1\right) + \theta_0 + \theta_1 x_1\right)}{\theta_2}$$

Effectively, α offsets θ_0 by $\ln\left(\frac{2}{\alpha} - 1\right)$.

$P(t = 1|x) = \frac{P(y = 1|x)}{\alpha}$. If we only want to rank the examples (sort them) in order from likeliest to least likely, then we don't even need to estimate α. We can rank the examples based purely on $P(y = 1|x)$, our classifier.

11.2.1 2a – correction factor between true labels and database

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

$$P\left(y^{(i)} = 1 \middle| t^{(i)} = 1, x^{(i)}\right) = P\left(y^{(i)} = 1 \middle| t^{(i)} = 1\right)$$

This is equivalent to saying that the labeled examples were selected uniformly at random from the set of positive examples. Remember that $y^{(i)}$ can be 0 even if $t^{(i)} = 1$ – this just means this is an unlabeled and positive example.

Prove that the probability of an example being labeled differs by a constant factor from the probability of an example being positive, i.e.

$$P\left(t^{(i)} = 1 \middle| x^{(i)}\right) = \frac{P\left(y^{(i)} = 1 \middle| x^{(i)}\right)}{\alpha}$$

$$\alpha \in \mathbb{R}$$

In fact, $\alpha = P\left(y^{(i)} = 1 \middle| t^{(i)} = 1\right)$.

Proof.

$$P\left(x^{(i)}, y^{(i)} = 1, t^{(i)} = 1\right) = P\left(x^{(i)}\right) P\left(y^{(i)} = 1 \middle| x^{(i)}\right) P\left(t^{(i)} = 1 \middle| x^{(i)}, y^{(i)} = 1\right)$$

Since $P\left(t^{(i)} = 1 \middle| x^{(i)}, y^{(i)} = 1\right) = 1$,

$$P\left(x^{(i)}, y^{(i)} = 1, t^{(i)} = 1\right) = P\left(x^{(i)}\right) P\left(y^{(i)} = 1 \middle| x^{(i)}\right)$$

We can write a different expression:

$$P\left(x^{(i)}, y^{(i)} = 1, t^{(i)} = 1\right) = P\left(x^{(i)}\right) P\left(t^{(i)} = 1 \middle| x^{(i)}\right) P\left(y^{(i)} = 1 \middle| t^{(i)} = 1, x^{(i)}\right)$$

With the conditional independence assumption, we have

$$P\left(x^{(i)}, y^{(i)} = 1, t^{(i)} = 1\right) = P\left(x^{(i)}\right) P\left(t^{(i)} = 1 \middle| x^{(i)}\right) P\left(y^{(i)} = 1 \middle| t^{(i)} = 1\right)$$

Then setting the two expressions equal,

$$P\left(x^{(i)}\right) P\left(y^{(i)} = 1 \middle| x^{(i)}\right) = P\left(x^{(i)}\right) P\left(t^{(i)} = 1 \middle| x^{(i)}\right) P\left(y^{(i)} = 1 \middle| t^{(i)} = 1\right)$$

$$\frac{P\left(y^{(i)} = 1 \middle| x^{(i)}\right)}{P\left(y^{(i)} = 1 \middle| t^{(i)} = 1\right)} = P\left(t^{(i)} = 1 \middle| x^{(i)}\right)$$

$$\alpha = P\left(y^{(i)} = 1 \middle| t^{(i)} = 1\right)$$

11.2.2 2b – estimate correction factor

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

$$h(x^{(i)}) \approx \alpha \; \forall \; x^{(i)} \in V_+$$

Proof. From 2a,

$$h(x^{(i)}) \approx P(y^{(i)} = 1 | x^{(i)}) = \alpha P(t^{(i)} = 1 | x^{(i)})$$

For examples $x^{(i)} \in V_+$, $P(t^{(i)} = 1 | x^{(i)}) = 1$, so

$$h(x^{(i)}) \approx \alpha$$

11.3 Problem 3 – Poisson regression

11.3.1 3a – Poisson distribution is in the exponential family

Prove that the Poisson distribution is in the exponential family.

$$P(y; \lambda) = \frac{e^{-\lambda} \lambda^y}{y!} = \frac{1}{y!} \exp(\log \lambda^y) \exp(-\lambda) = \frac{1}{y!} \exp\big((\log \lambda) y - \lambda\big)$$

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

$$b(y) = \frac{1}{y!}$$

$$\eta = \log \lambda \rightarrow \lambda = \exp \eta$$

$$T(y) = y$$

$$a(\eta) = \lambda = \exp \eta$$

11.3.2 3b – Poisson canonical response function

The canonical response function is

$$g(\eta) = \frac{\partial}{\partial \eta} a(\eta) = \frac{\partial}{\partial \eta} \exp \eta = \exp \eta$$

Since $\eta = \theta^T x$,

$$h_\theta(x) = g(\theta^T x) = \exp \theta^T x$$

11.3.3 3c – gradient ascent update rule

Derive the gradient ascent update rule for the Poisson distribution.

$$P(y;\eta) = \frac{1}{y!}\exp(\eta y - \exp\eta) \rightarrow \log P(y;\eta) = -\log y! + \eta y - \exp\eta$$
$$= -\log y! + \theta^T x^{(i)} y^{(i)} - \exp\theta^T x^{(i)}$$

$$\frac{\partial}{\partial\theta_j}\log P(y;\eta) = x_j^{(i)} y^{(i)} - x_j^{(i)}\exp\theta^T x^{(i)} = \left(y^{(i)} - \exp\theta^T x^{(i)}\right)x_j^{(i)}$$

Then

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

11.4 Problem 4 – convexity of GLMs

As we know from lecture, GLMs are trained using the negative log-likelihood (NLL) as the loss function, which is equivalent to MLE (maximizing the log-likelihood is equivalent to minimizing the negative log-likelihood). In this problem, we'll show that the loss of a GLM is a convex function w.r.t. the model parameters θ, which is convenient since this means any local minimum is also the global minimum. We'll show that the Hessian of the loss w.r.t. θ is positive semi-definite (PSD).

Assume η is a scalar $\in \mathbb{R}$, $P(y|x;\theta) \sim \text{ExponentialFamily}(\eta)$, and $T(y) = y$ so that

$$P(y;\eta) = b(y)\exp\left(\eta y - a(\eta)\right)$$

11.4.1 4a – mean of exponential family distribution

By the definition of a probability distribution,

$$\int P(y;\eta)dy = 1$$

Take partial derivative of both sides:

$$\frac{\partial}{\partial\eta}\left(\int_y P(y;\eta)dy\right) = \frac{\partial}{\partial\eta}(1) = 0$$

$$\frac{\partial}{\partial\eta}\left(\int_y P(y;\eta)dy\right) = \int_y \frac{\partial}{\partial\eta}P(y;\eta)dy = \int_y \frac{\partial}{\partial\eta}\left(b(y)\exp\left(\eta y - a(\eta)\right)\right)dy$$
$$= \int_y \left(b(y)\exp\left(\eta y - a(\eta)\right)\left(y - \frac{\partial a}{\partial\eta}\right)\right)dy = \int_y \left(P(y;\eta)\left(y - \frac{\partial a}{\partial\eta}\right)\right)dy$$
$$= \int_y y * P(y;\eta)dy - \frac{\partial a}{\partial\eta}\int_y P(y;\eta)dy = \int_y y * P(y;\eta)dy - \frac{\partial a}{\partial\eta} = 0$$

$$\frac{\partial a}{\partial \eta} = \int_y y * P(y;\eta)dy = E(y;\eta) = E(y|x;\theta)$$

11.4.2 4b – variance of exponential family distribution

By the definition of a probability distribution,

$$\int P(y;\eta)dy = 1$$

Take double partial derivative of both sides:

$$\frac{\partial^2}{\partial \eta^2}\left(\int_y P(y;\eta)dy\right) = \frac{\partial^2}{\partial \eta^2}(1) = 0$$

From 4a, we know that

$$\frac{\partial}{\partial \eta}\left(\int_y P(y;\eta)dy\right) = \int_y y * P(y;\eta)dy - \frac{\partial a}{\partial \eta}$$

Then

$$\begin{aligned}
\frac{\partial^2}{\partial \eta^2}\left(\int_y P(y;\eta)dy\right) &= \frac{\partial}{\partial \eta}\left(\int_y y * P(y;\eta)dy - \frac{\partial a}{\partial \eta}\right) = \int_y \frac{\partial}{\partial \eta}\big(y * P(y;\eta)\big)dy - \frac{\partial^2 a}{\partial \eta^2} \\
&= \int_y y * \frac{\partial}{\partial \eta}P(y;\eta)dy - \frac{\partial^2 a}{\partial \eta^2} = \int_y y * P(y;\eta)\left(y - \frac{\partial a}{\partial \eta}\right)dy - \frac{\partial^2 a}{\partial \eta^2} \\
&= \int_y y^2 * P(y;\eta)dy - \frac{\partial a}{\partial \eta}\int_y y * P(y;\eta)dy - \frac{\partial^2 a}{\partial \eta^2} \\
&= \int_y y^2 * P(y;\eta)dy - \left(\int_y y * P(y;\eta)dy\right)^2 - \frac{\partial^2 a}{\partial \eta^2} = 0
\end{aligned}$$

Then

$$\begin{aligned}
\frac{\partial^2 a}{\partial \eta^2} &= \int_y y^2 * P(y;\eta)dy - \left(\int_y y * P(y;\eta)dy\right)^2 = E(y^2;\eta) - E(y;\eta)^2 = \mathrm{Var}(y;\eta) \\
&= \mathrm{Var}(y|x;\theta)
\end{aligned}$$

11.4.3 4c – Hessian of loss function is positive semi-definite

Prove that the Hessian of the loss function is PSD, which means that the loss function is convex.

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

$$\log P(y;\eta) = \log b(y) + \eta y - a(\eta)$$

Over all samples,

$$\ell(\eta) = \sum_i \log b\left(y^{(i)}\right) + \eta y^{(i)} - a(\eta)$$

$$\ell(\theta) = \sum_i \log b\left(y^{(i)}\right) + \left(\theta^T x^{(i)}\right) y^{(i)} - a\left(\theta^T x^{(i)}\right)$$

$$\text{NLL} = -\frac{\ell(\theta)}{m} = -\frac{1}{m}\sum_i \log b\left(y^{(i)}\right) + \left(\theta^T x^{(i)}\right) y^{(i)} - a\left(\theta^T x^{(i)}\right)$$

The value of row j and column k in the Hessian is given by

$$\begin{aligned} H_{jk} &= \frac{\partial^2}{\partial\theta_j \partial\theta_k}(\text{NLL}) = \frac{\partial^2}{\partial\theta_j \partial\theta_k}\left(-\frac{1}{m}\sum_i \log b\left(y^{(i)}\right) + \left(\theta^T x^{(i)}\right) y^{(i)} - a\left(\theta^T x^{(i)}\right)\right) \\ &= \frac{\partial^2}{\partial\theta_j \partial\theta_k}\left(-\frac{1}{m}\sum_i \log b\left(y^{(i)}\right) + \left(\theta^T x^{(i)}\right) y^{(i)} - a\left(\theta^T x^{(i)}\right)\right) \\ &= \frac{1}{m}\sum_i \frac{\partial}{\partial\theta_j} a'\left(\theta^T x^{(i)}\right) x_k^{(i)} = \frac{1}{m}\sum_i a''\left(\theta^T x^{(i)}\right) x_j^{(i)} x_k^{(i)} \end{aligned}$$

Then $z^T H z$ evaluates to (see problem 1a for similar proof)

$$\begin{aligned} \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{n}\sum_{j=1}^{n} a''\left(\theta^T x^{(i)}\right) x_j^{(i)} z_j x_k^{(i)} z_k &= \frac{1}{m}\sum_{i=1}^{m} a''\left(\theta^T x^{(i)}\right) \sum_{k=1}^{n} x_k^{(i)} z_k \sum_{j=1}^{n} x_j^{(i)} z_j \\ &= \frac{1}{m}\sum_{i=1}^{m} a''\left(\theta^T x^{(i)}\right)\left(x^{(i)^T} z\right)^2 \end{aligned}$$

Since $a''\left(\theta^T x^{(i)}\right) = \text{Var}\left(y^{(i)} \middle| x^{(i)}; \theta\right) \geq 0$ and $\left(x^{(i)^T} z\right)^2 \geq 0$, we have $z^T H z \geq 0$.

11.5 Problem 5 – locally weighted linear regression

In locally weighted linear regression, we want to "weight" different training examples differently. Specifically, we want to minimize the cost function

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

11.5.1 5ai – matrix form of cost function

Show that $J(\theta)$ can be written as

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

Where

$$X = \begin{bmatrix} {x^{(1)}}^T \\ {x^{(2)}}^T \\ \vdots \\ {x^{(m)}}^T \end{bmatrix} \in \mathbb{R}^{m\times(n+1)}, \theta \in \mathbb{R}^{(n+1)\times 1}, y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

W is a diagonal matrix with elements $w^{(i)}$.

$$W = \frac{1}{2}\begin{bmatrix} w^{(1)} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & w^{(m)} \end{bmatrix} \in \mathbb{R}^{m\times m}$$

11.5.2 5aii – minimize cost function

$$\begin{aligned} \nabla_\theta\big(J(\theta)\big) = \nabla_\theta\big((X\theta - y)^T W(X\theta - y)\big) &= \nabla_\theta\big((\theta^T X^T - y^T)W(X\theta - y)\big) \\ &= \nabla_\theta(\theta^T X^T WX\theta - \theta^T X^T Wy - y^T WX\theta + y^T Wy) \\ &= \nabla_\theta(\theta^T X^T WX\theta - (y^T WX\theta)^T - y^T WX\theta) \end{aligned}$$

All of the expressions inside the derivative reduce to one real value, meaning

$$(y^T WX\theta)^T = y^T WX\theta \in \mathbb{R}$$

Then

$$\begin{aligned} \nabla_\theta(\theta^T X^T WX\theta - (y^T WX\theta)^T - y^T WX\theta) &= \nabla_\theta(\theta^T X^T WX\theta - 2y^T WX\theta) \\ &= \nabla_\theta(\theta^T X^T WX\theta - 2(X^T Wy)^T\theta) \end{aligned}$$

For the second expression, note that $\nabla_x(b^T x) = b$ (at least, if b and x are column vectors). For the first expression, note that $\nabla_x(x^T Ax) = 2Ax$ (see problem set 0).

Then

$$\nabla_\theta(\theta^T X^T WX\theta - 2(X^T Wy)^T\theta) = 2X^T WX\theta - 2X^T Wy$$

$$2X^T WX\theta - 2X^T Wy \overset{\text{set}}{=} \vec{0}$$

The normal equation is

$$X^T WX\theta = X^T Wy$$

Then the optimum θ is

$$\theta = (X^T WX)^{-1} X^T Wy$$

11.5.3 5aiii – probabilistic interpretation

Suppose we have a dataset $\{x^{(i)}, y^{(i)}\}$ and each sample comes from a Gaussian distribution with a different mean $\theta^T x^{(i)}$ and different variance ${\sigma^{(i)}}^2$, where ${\sigma^{(i)}}^2$ are fixed, known constants:

$$P(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma^{(i)}} \exp\left(-\frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2{\sigma^{(i)}}^2}\right)$$

This is different from classic linear regression which assumes ${\sigma^{(i)}}^2 = \sigma^2 \; \forall \; i$. Show that finding the MLE of θ is equivalent to solving a weighted linear regression problem.

$$L(\theta) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma^{(i)}} \exp\left(-\frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2{\sigma^{(i)}}^2}\right)$$

$$\ell(\theta) = \log L = -\sum_{i=1}^{m} \log \sqrt{2\pi}\sigma^{(i)} + \frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2{\sigma^{(i)}}^2}$$

Finding the MLE is equivalent to minimizing $\sum_{i=1}^{m} \frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2{\sigma^{(i)}}^2}$:

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

From the top, solving a weighted linear regression means minimizing the cost function

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

That is,

$$w^{(i)} = \frac{1}{{\sigma^{(i)}}^2}$$