CPSC 440: Advanced Machine Learning

Density Estimation

Mark Schmidt

University of British Columbia

Winter 2021

Admin

- Canvas:
  - The 440/540 Canvas page has all the links you need.
- Assignment 1 due tonight.
  - Gradescope submissions instructions posted on Piazza.
- Today is the last day to drop the course.
  - Consider whether you want to take this course if you found Assignment 1 difficult.
    - Particularly if you haven't taken CPSC 320 or 340, even if you are in CPSC or ECE.
    - I remember taking classes I wasn't ready for, it is not going to get better after week 2!

Digression: "Debugging by Frustration" and "Debugging by TA"

- Here is one way to write a complicated program (e.g., softmax with gradient):
  1. Write the entire function at once.
  2. Try it out to "see if it works".
  3. Spend hours fiddling with commands, trying to find magic working combination.
  4. Send code to the TA, asking "what is wrong?"

- If you are lucky, step 2 works and you are done!

- If you are not lucky, this takes way longer than principled coding methods.
  - This is also a great way to introduce bugs into your code.
  - And you won't be able to do Step 4 when you graduate.

Debugging 101

- What strategies could we use to debug an implementation of these functions?

$$f(w) = \sum_{i=1}^{n} [f_i(w) + g_i(w)] \quad \text{and its gradient.}$$

  - Use "print" statements to see what is happening at each step of the code.
    - Or a debugger
  - Check if $\nabla f_i(w)$ is correct on its own with numerical differencing.
    - Maybe you have the second term right but the not first term.
  - Check if $\nabla g_i(w)$ is correct on its own with numerical differencing.
    - Maybe you have the first term right but not the second term.
  - Try the implementation with only one training example or only one feature.
    - Maybe there is an indexing problem, or things aren't being aggregated properly.
  - Develop one ore more simple "test case", where you worked out the result by hand.
    - Maybe one of the functions you are using does not work the way you think it does.
  - Code up $f(w)$, and then run gradient descent with numerical differencing.
    - Maybe your objective function is wrong so it doesn't matter if the gradietn is correct.
- TAs/instructor are happy to help, but when sending code you need to include:
  - "This is what I've tried to diagnose the problem and the problem seems to be here".

Last Time: Structure Prediction

- "Classic" machine learning: models $p(y^i \mid x^i)$, where y^i was a single variable.
  - In 340 we used simple distributions like the Gaussian and sigmoid.
- Structured prediction: y^i could be a vector, protein, image, dependency tree,. . . .
  - This requires defining more-complicated distributions.
- But before considering $p(y^i \mid x^i)$ for complicated y^i:
  - We'll first consider just modeling $p(y^i)$ or $p(x^i)$ with multiple variables, without worrying about conditioning (this is already a hard problem).
    - If you know how to model $p(x^i)$, then $p(y^i \mid x^i)$ isn't much more complicated.

Density Estimation

- The next topic we'll focus on is density estimation:

$$X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \quad \tilde{X} = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$

  - What is probability of $[1\ 0\ 1\ 1]$?
  - Want to estimate probability of feature vectors x^i.

- For the training data this is easy:
  - Set $p(x^i)$ to "number of times x^i is in the training data" divided by n.

- We're interested in the probability of test data,
  - What is probability of seeing feature vector $\tilde{x}^i$ for a new example i.

Density Estimation Applications

- Density estimation could be called a "master problem" in machine learning.
  - Solving this problem lets you solve a lot of other problems.
- If you have $p(x^i)$ then:
  - Outliers could be cases where $p(x^i)$ is small.
  - Missing data in x^i can be "filled in" based on $p(x^i)$.
  - Vector quantization can be achieved by assigning shorter code to high $p(x^i)$ values.
  - Association rules can be computed from conditionals $p(x^i_j \mid x^i_k)$.
- We can also do density estimation on (x^i, y^i) jointly:
  - Supervised learning can be done by conditioning to give $p(y^i \mid x^i)$.
  - Feature relevance can be analyzed by looking at $p(x^i \mid y^i)$.
- If features are continuous, we are estimating the "probability density function".
  - I'll sloppily just say "probability" though.

Unsupervised Learning

- Density estimation is an unsupervised learning method.
  - We only have x^i values, but no explicit target labels.
  - You want to do "something" with them.

- Some unsupervised learning tasks from CPSC 340 (depending on semester):
  - Clustering: what types of x^i are there?
  - Association rules: which x_j and x_k occur together?
  - Outlier detection: is this a "normal" x^i?
  - Latent-factors: what "parts" are x^i made from?
  - Data visualization: what do the high-dimensional x^i look like?
  - Ranking: which are the most important x^i?

- You can probably address all these if you can do density estimation.

Bernoulli Distribution on Binary Variables

- Let's start with the simplest case: $x^i \in \{0, 1\}$ (e.g., coin flips),

$$X = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

- For IID data the only choice is the Bernoulli distribution:

$$p(x^i = 1 \mid \theta) = \theta, \quad p(x^i = 0 \mid \theta) = 1 - \theta.$$

- We can write both cases as

$$p(x^i \mid \theta) = \theta^{\mathcal{I}[x^i=1]}(1-\theta)^{\mathcal{I}[x^i=0]}, \text{ where } \mathcal{I}[y] = \begin{cases} 1 & \text{if } y \text{ is true} \\ 0 & \text{if } y \text{ is false} \end{cases}.$$

Maximum Likelihood with Bernoulli Distribution

- MLE for Bernoulli likelihood with IID data is

$$\begin{aligned}
\underset{0 \leq \theta \leq 1}{\text{argmax}}\, p(X \mid \theta) &= \underset{0 \leq \theta \leq 1}{\text{argmax}} \prod_{i=1}^{n} p(x^i \mid \theta) \\
&= \underset{0 \leq \theta \leq 1}{\text{argmax}} \prod_{i=1}^{n} \theta^{\mathcal{I}[x^i=1]}(1-\theta)^{\mathcal{I}[x^i=0]} \\
&= \underset{0 \leq \theta \leq 1}{\text{argmax}} \underbrace{\theta^1\theta^1 \cdots \theta^1}_{\text{number of } x_i = 1} \underbrace{(1-\theta)(1-\theta)\cdots(1-\theta)}_{\text{number of } x_i = 0} \\
&= \underset{0 \leq \theta \leq 1}{\text{argmax}}\, \theta^{n_1}(1-\theta)^{n_0},
\end{aligned}$$

  where n_1 is count of number of 1 values and n_0 is the number of 0 values.
- If you equate the derivative of the log-likelihood with zero, you get $\theta = \frac{n_1}{n_1+n_0}$.
- So if you toss a coin 50 times and it lands heads 24 times, your MLE is $24/50$.

Multinomial Distribution on Categorical Variables

- Consider the multi-category case: $x^i \in \{1, 2, 3, \ldots, k\}$ (e.g., rolling di),

$$X = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 3 \\ 1 \\ 2 \end{bmatrix}.$$

- The categorical distribution is

$$p(x^i = c \mid \theta_1, \theta_2, \ldots, \theta_k) = \theta_c,$$

  where each $\theta_c \geq 0$ and $\sum_{c=1}^{k} \theta_c = 1$.
- We can write this for a generic x as

$$p(x^i \mid \theta_1, \theta_2, \ldots, \theta_k) = \prod_{c=1}^{k} \theta_c^{\mathcal{I}[x^i = c]}.$$

Multinomial Distribution on Categorical Variables

- Using Lagrange multipliers (bonus) to handle constraints, the MLE is

$$\theta_c = \frac{n_c}{n} = \frac{n_c}{\sum_{c'} n_{c'}}. \quad \text{(“fraction of times you rolled a 4”)}$$

- If we never see category 4 in the data, should we assume $\theta_4 = 0$?
  - If we assume $\theta_4 = 0$ and we have a 4 in test set, our test set likelihood is 0.
- To leave room for this possibility we often use “Laplace smoothing”,

$$\theta_c = \frac{n_c + 1}{\sum_{c'}(n_{c'} + 1)}.$$

- This is like adding a “fake” example to the training set for each class.

MAP Estimation with Bernoulli Distributions

- In the binary case, a generalization of Laplace smoothing is

$$\theta = \frac{n_1 + \alpha - 1}{(n_1 + \alpha - 1) + (n_0 + \beta - 1)},$$

- We get the MLE when $\alpha = \beta = 1$, and Laplace smoothing with $\alpha = \beta = 2$.
- This is a MAP estimate under a beta prior,

$$p(\theta \mid \alpha, \beta) = \frac{1}{B(\alpha, \beta)} \theta^{\alpha-1}(1-\theta)^{\beta-1},$$

  where the beta function B makes the probability integrate to one.

$$\text{We want } \int_\theta p(\theta \mid \alpha, \beta) d\theta = 1, \quad \text{so define} \quad B(\alpha, \beta) = \int_\theta \theta^{\alpha-1}(1-\theta)^{\beta-1} d\theta.$$

- Note that $B(\alpha, \beta)$ is constant in terms of θ, it doesn't affect MAP estimate.
  - Above formula assumes $n_1 + \alpha > 1$ and $n_0 + \beta > 1$ (other cases in bonus).

MAP Estimation with Categorical Distributions

- In the categorical case, a generalization of Laplace smoothing is

$$\theta_c = \frac{n_c + \alpha_c - 1}{\sum_{c'=1}^{k}(n_{c'} + \alpha_{c'} - 1)},$$

  which is a MAP estimate under a Dirichlet prior,

$$p(\theta_1, \theta_2, \ldots, \theta_k \mid \alpha_1, \alpha_2, \ldots, \alpha_k) = \frac{1}{B(\alpha)} \prod_{c=1}^{k} \theta_c^{\alpha_c - 1},$$

  where $B(\alpha)$ makes the multivariate distribution integrate to 1 over θ,

$$B(\alpha) = \int_{\theta_1} \int_{\theta_2} \cdots \int_{\theta_{k-1}} \int_{\theta_k} \prod_{c=1}^{k} \left[\theta_c^{\alpha_c - 1}\right] d\theta_k d\theta_{k-1} \cdots d\theta_2 d\theta_1.$$

- Because of MAP-regularization connection, Laplace smoothing is regularization.

Outline

1 Discrete Density Estimation ($d = 1$)

2 Discrete Density Estimation ($d > 1$)

General Discrete Distribution

- Now consider the case where $x^i \in \{0, 1\}^d$:
  - Words in e-mails, pixels in binary image, locations of cancers, and so on.

$$X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}.$$

- Now there are 2^d possible values of vector x^i.
  - General discrete distribution would consider θ_{0000}, θ_{0001}, θ_{0010}, θ_{0011}, θ_{0100},...
  - You can compute the MLE of this distribution in $O(nd)$.
    - See at most n unique x^i values, and using a hash data structure.
  - But unless we have a small number of repeated x^i values, we'll hopelessly overfit.
- With finite dataset, we'll need to make assumptions...

Product of Independent Distributions

- A common assumption is that the variables are independent:

$$p(x_1^i, x_2^i, \dots, x_d^i \mid \Theta) = \prod_{j=1}^{d} p(x_j^i \mid \theta_j).$$

- Now we just need to model each column of X as its own dataset:

$$X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \quad \rightarrow \quad X_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad X_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad X_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}, \quad X_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}.$$

- A big assumption, but now you can fit Bernoulli for each variable.
  - We used a similar independence assumption in CPSC 340 for naive Bayes.

Digression: Optimizing "Separable" Functions

- Consider an optimization problem of the form

$$\min_{w_1, w_2} f_1(w_1) + f_2(w_2).$$

- This is called a separable function.
  - The variable w_1 only affects the first term, and w_2 only affects second.
- With separable functions, you can optimize each term separately.
  - Gradient with respect to w_1 is: $\nabla f_1(w_1)$ (not affected by w_2).
  - Gradient with repsect to w_2 is: $\nabla f_2(w_2)$ (not affected by w_1).
- Similarly, if you have $\sum_{j=1}^{d} f_j(w_j)$, you optimize each f_j separately.
  - Use this property to simplify your assignment questions.

Digression: Optimizing "Separable" Functions

- Let's show that "product of independent" model fits each column separately.

$$p(x_1^i, x_2^i, \ldots, x_d^i \mid \Theta) = \prod_{j=1}^{d} p(x_j^i \mid \theta_j).$$

- MLE:

$$\begin{aligned}
&\underset{\Theta}{\text{argmin}} - \log \prod_{i=1}^{n} p(x_1^i, x_2^i, \ldots, x_d^i \mid \Theta) && \text{(NLL for IID data)}\\
\equiv\ &\underset{\Theta}{\text{argmin}} - \sum_{i=1}^{n} \log p(x_1^i, x_2^i, \ldots, x_d^i \mid \Theta) && (\log(\alpha\beta) = \log(\alpha) + \log(\beta))\\
\equiv\ &\underset{\Theta}{\text{argmin}} - \sum_{i=1}^{n} \log \prod_{j=1}^{d} p(x_j^i \mid \theta_j) && \text{(product of independent assumption)}\\
\equiv\ &\underset{\Theta}{\text{argmin}} - \sum_{i=1}^{n} \sum_{j=1}^{d} \log p(x_j^i \mid \theta_j)) && (\log(\alpha\beta) = \log(\alpha) + \log(\beta))\\
\equiv\ &\underset{\Theta}{\text{argmin}} - \sum_{j=1}^{d} \sum_{i=1}^{n} \log p(x_j^i \mid \theta_j)) && \text{(exchanging sums gives separable function: } f_j(\theta_j) = -\sum_{i=1}^{n} \log p(x_j^i \mid \theta_j)).
\end{aligned}$$

- Since the NLL is separable in the Θ_j, you can minimize each f_j separately.

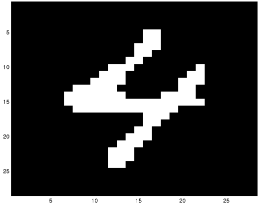
Big Picture: Training and Inference

- Density estimation training phase:
  - Input is a matrix X.
  - Output is a model.

- Density estimation prediction phase:
  - Input is a model, and possibly test data $\tilde{X}$
  - Many possible prediction tasks:
    - Measure probability of test examples $\tilde{x}^i$.
    - Generate new samples x according to the distribution.
    - Find configuration x maximizing $p(x)$.
    - Compute marginal probability like $p(x_j = c)$ for somr variable j and value c.
    - Compute conditional queries like $p(x_j = c \mid x_{j'} = c')$.
- We call these inference tasks.
  - More complicated than superised learning.
    - In supervised learning, inference was "find $\hat{y}^i$" or "compute $p(y = c \mid w, x)$".

Example: Independent vs. General Discrete on Digits

- Consider handwritten images of digits:

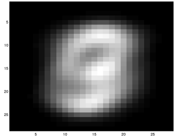
$x^i = \text{vec}\Bigg($  $\Bigg)$,

so each row of X contains all pixels from one image of a 0, 1, 2, . . . , or a 9.

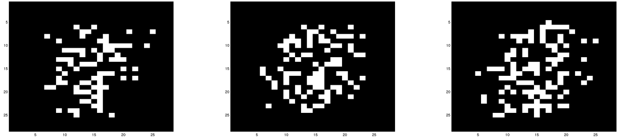
- Previously we had labels and wanted to recognize that this is a 4.
- In density estimation we want probability distribution over images of digits.
- Inference tasks:
  - Given an image, what is the probability that it's a digit?
  - Sampling from the density, which should generate images of new digits.

Example: Independent vs. General Discrete on Digits

- Fitting independent Bernoullis to this data gives a parameter θ_j for each pixel j.
  - MLE is "fraction of times we have a 1 at pixel j":



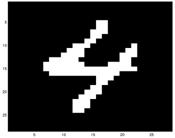
- Samples generated from independent Bernoulli model:



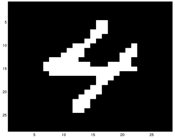
  - Flip a coin that lands hands with probability θ_j for each pixel j.
- This is clearly a terrible model: misses dependencies between pixels.

Example: Independent vs. General Discrete on Digits

- Here is a sample from the MLE with the general discrete distribution:



- Here is an image with a probability of 0:



- This model memorized training images and doesn't generalize.
  - MLE puts probability at least $1/n$ on training images, and 0 on non-training images.

Density Estimation and Fundamental Trade-off

- "Product of independent" distributions (with d parameters):
  - Easily estimate each θ_c but can't model many distributions.
- General discrete distribution (with 2^d parameters):
  - Hard to estimate 2^d parameters but can model any distribution.

- An unsupervised version of the fundamental trade-off:
  - Simple models often don't fit the data well but don't overfit much.
  - Complex models fit the data well but often overfit.

- We'll consider models that lie between these extremes:
  1. Mixture models.
  2. Markov models.
  3. Graphical models.
  4. Boltzmann machines.
  5. Fully-convolutional and recurrent neural networks.
  6. Variational autoencoders.
  7. Generative adversarial networks.

Summary

- Density estimation: unsupervised modelling of probability of feature vectors.
- Bernoulli distribution for modeling binary data.
- Categorical distribution for modeling discrete data.
- MAP estimation with beta and Dirichlet priors ("Laplace smoothing").
- Product of independent distributions is simple/crude density estimation method.
- Next time: we start talking about density estimation with continuous data.

Debugging Checklist (From Cinda Heeran)

LMNOP List

Before placing yourself on the Queue or posting to Piazza, please make sure to:

1. Format your code so it can be read.
2. Comment your code so you (and we) can understand what it's doing.
3. Isolate your error as specifically as you can. "This line of code doesn't do what I expect" or "this function returns the wrong value for these parameters."
4. Sketch your algorithm on paper, with pencil. Use small or simplified instances of the problem.
5. Re-read the specification to make sure you're solving the right problem.
6. Read error messages carefully and try to understand what they're telling you.
7. Google your error message. Stackoverflow.com is a gift from god, populated by angels.
8. Break your functions into smaller chunks, or even helper functions, and make sure each smaller piece behaves as you expect (i.e. test them!).
9. Challenge the assumptions you make
   a. Does the order of what I'm doing matter?
   b. Do I need to check the input values?
   c. Does this do what I expect? (will it overflow?, will it be a shallow copy?)

Help-seeking behaviors that are doomed to fail:

1. Saying only, "It's not working!"
2. Asking if your code looks correct without having finished it or tested it.
3. Ignoring the suggestions of the TA.
4. Looking at Github for answers.

ProTip: Check your code into a repository every time you finish a function or take a break, and give yourself good commit messages. This helps enormously if you have to roll back changes.

Lagrangian Function for Optimization with Equality Constraints

- Consider minimizing a differentiable f with linear equality constraints,

$$\underset{Aw=b}{\operatorname{argmin}}\, f(w).$$

- The Lagrangian of this problem is defined by

$$L(w, v) = f(w) + v^T(Aw - b),$$

  for a vector $v \in \mathbb{R}^m$ (with A being m by d).

- At a solution of the problem we must have

$$\begin{aligned} \nabla_w L(w, v) &= \nabla f(w) + A^T v = 0 && \text{(gradient is orthogonal to constraints)} \\ \nabla_v L(w, v) &= Aw - b = 0 && \text{(constraints are satisfied)} \end{aligned}$$

- So solution is stationary point of Lagrangian.

Lagrangian Function for Optimization with Equality Constraints

- Scans from Bertsekas discussing Lagrange multipliers (also see CPSC 406).

3.1 NECESSARY CONDITIONS FOR EQUALITY CONSTRAINTS

In this section we consider problems with equality constraints of the form

$$\begin{aligned}&\text{minimize} \quad f(x)\\&\text{subject to} \quad h_i(x) = 0, \qquad i = 1,\dots,m.\end{aligned} \qquad \text{(ECP)}$$

We assume that $f : \Re^n \mapsto \Re$, $h_i : \Re^n \mapsto \Re$, $i = 1,\dots,m$, are continuously differentiable functions. *All the necessary and the sufficient conditions of this chapter relating to a local minimum can also be shown to hold if f and h_i are defined and are continuously differentiable within just an open set containing the local minimum.* The proofs are essentially identical to those given here.

For notational convenience, we introduce the constraint function $h : \Re^n \mapsto \Re^m$, where

$$h = (h_1,\dots,h_m).$$

We can then write the constraints in the more compact form

$$h(x) = 0. \qquad (3.1)$$

Our basic Lagrange multiplier theorem states that for a given local minimum x^*, there exist scalars $\lambda_1,\dots,\lambda_m$, called *Lagrange multipliers*, such that

$$\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i \nabla h_i(x^*) = 0. \qquad (3.2)$$

There are two ways to interpret this equation:

(a) The cost gradient $\nabla f(x^*)$ belongs to the subspace spanned by the constraint gradients at x^*. The example of Fig. 3.1.1 illustrates this interpretation.

(b) The cost gradient $\nabla f(x^*)$ is orthogonal to the subspace of *first order feasible variations*

$$V(x^*) = \{\Delta x \mid \nabla h_i(x^*)'\Delta x = 0,\ i = 1,\dots,m\}.$$

This is the subspace of variations Δx for which the vector $x = x^* + \Delta x$ satisfies the constraint $h(x) = 0$ up to first order. Thus, according to the Lagrange multiplier condition of Eq. (3.2), at the local minimum x^*, the first order cost variation $\nabla f(x^*)'\Delta x$ is zero for all variations

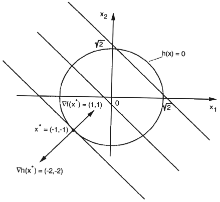


Figure 3.1.1. Illustration of the Lagrange multiplier condition (3.1) for the problem

$$\begin{aligned}&\text{minimize} \quad x_1 + x_2\\&\text{subject to} \quad x_1^2 + x_2^2 = 2.\end{aligned}$$

At the local minimum $x^* = (-1,-1)$, the cost gradient $\nabla f(x^*)$ is normal to the constraint surface and is therefore, collinear with the constraint gradient $\nabla h(x^*) = (-2,-2)$. The Lagrange multiplier is $\lambda = 1/2$.

Δx in this subspace. This statement is analogous to the "zero gradient condition" $\nabla f(x^*) = 0$ of unconstrained optimization.

Here is a formal statement of the main Lagrange multiplier theorem:

Proposition 3.1.1: (Lagrange Multiplier Theorem – Necessary Conditions) Let x^* be a local minimum of f subject to $h(x) = 0$, and assume that the constraint gradients $\nabla h_1(x^*),\dots,\nabla h_m(x^*)$ are linearly independent. Then there exists a unique vector $\lambda^* = (\lambda_1^*,\dots,\lambda_m^*)$, called a *Lagrange multiplier vector*, such that

$$\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla h_i(x^*) = 0. \qquad (3.3)$$

If in addition f and h are twice continuously differentiable, we have

Lagrangian Function for Optimization with Equality Constraints

- We can use these optimality conditions,

$$\nabla_w L(w, v) = \nabla f(w) + A^T v = 0 \qquad \text{(gradient is orthogonal to constraints)}$$
$$\nabla_v L(w, v) = Aw - b = 0 \qquad \text{(constraints are satisfied)}$$

  to solve some constrained optimization problems.

- A typical approach might be:
  1. Solve for w in the equation $\nabla_w L(w, v) = 0$ to get $w = g(v)$ for some function g.
  2. Plug this $w = g(v)$ into the the equation $\nabla_v L(w, v) = 0$ to solve for v.
  3. Use this v in $g(v)$ to get the optimal w.

- But note that these are necessary conditions (may need to check it's a min).

Lagrangian Function for Optimization with Equality Constraints

- Example: minimize $\frac{1}{2}(w_1 + 1)^2 + \frac{1}{2}(w_2 + 2)^2$ subject to $w_1 + w_2 = 1$.
  - So Lagrangian is $L(w, v) = \frac{1}{2}(w_1 + 1)^2 + \frac{1}{2}(w_2 + 2)^2 + v(w_1 + w_2 - 1)$.
- Solving this problem using the Lagrangian:
  1. Solve for w in the equation $\nabla_w L(w, v) = 0$ to get $w = g(v)$ for some function g.

$$\nabla_{w_1} L(w, v) = (w_1 + 1) + v, \quad \text{so with } \nabla_{w_1} L(w, v) = 0 \text{ we have } w_1 = -v - 1,$$
$$\nabla_{w_2} L(w, v) = (w_2 + 2) + v, \quad \text{so with } \nabla_{w_2} L(w, v) = 0 \text{ we have } w_2 = -v - 2.$$

  2. Plug this $w = g(v)$ into the the equation $\nabla_v L(w, v) = 0$ to solve for v.

$$\nabla_v L(w, v) = w_1 + w_2 - 1, \quad \text{so with } \nabla_v L(w, v) = 0 \text{ we have using the above } w_1 \text{ and } w_2\text{:}$$
$$0 = (-v - 1) + (-v - 2) - 1, \quad \text{or v = -2}.$$

  3. Use this v in $g(v)$ to get the optimal w.

$$w_1 = -(-2) - 1,$$
$$w_2 = -(-2) - 2,$$

giving $w_1 = 1$ and $w_2 = 0$ as the minimum subject to the constraint.

Beta Distribution with $\alpha < 1$ and $\beta < 1$

- Wikipedia has a rather extensive article on the beta distribution:
`https://en.wikipedia.org/wiki/Beta_distribution`

- In their picture of the beta distribution with $\alpha = \beta = 0.5$, you see that it's "U"-shaped, with modes at the extreme values of 0 or 1. I think of this as regularizing towards the coin being biased, but you're not sure whether the coin is biased towards heads or tails.

- Also, the MAP formula given in class only works if $n_1 + \alpha$ and $n_0 + \alpha$ are both greater than 1. This trivial holds for Laplace smoothing and the MLE case, but doesn't hold if you haven't seen both heads and tails when α and β are less than 1. In that case, the MAP will be either 0 or 1 or both depending on the precise values.