Machine Learning Cheatsheet

Janet Matsen's Machine Learning (ML) notes from CSE 446, Winter 2016. http://courses.cs.washington.edu/courses/cse446/16wi/
Used LaTeX template from an existing Statistics cheat sheet: https://github.com/wzchen/probability_cheatsheet, by William Chen (http://wzchen.com) and Joe Blitzstein.

Last Updated February 6, 2016

Essential ML ideas

- Never ever ever touch the test set
- You know you are overfitting when there is a big test between train and test results. E.g. metric of percent wrong.
- Need to be comfortable taking a hit on fitting accuracy if you can get a benefit on the result.
- Bias vs variance trade-off. High bias when the model is too simple & doesn't fit the data well. High variance is when small changes to the data set lead to large solution changes.

Math/Stat Review

Random Variable X belongs to set Ω

Conditional Probability *is* Probability $P(A|B)$ is a probability function for any fixed B. Any theorem that holds for probability also holds for conditional probability. $P(A|B) = P(A \cap B)/P(B)$

Bayes' Rule - Bayes' Rule unites marginal, joint, and conditional probabilities. We use this as the definition of conditional probability.

$$P(\mathbf{A}|\mathbf{B}) = \frac{P(\mathbf{A} \cap \mathbf{B})}{P(\mathbf{B})} = \frac{P(\mathbf{B}|\mathbf{A})P(\mathbf{A})}{P(\mathbf{B})}$$

$$P(A = a \mid B) = \frac{P(A = a)P(B \mid A = a)}{\sum_{a'} P(A = a)P(B \mid A = a)}$$

Law of Total Probability : $\sum_x P(X = x) = 1$

Product Rule : $P(A, B) = P(A \mid B) \cdot P(B)$

Sum Rule : $P(A) = \sum_{x \in \Omega} P(A, B = b)$

i.i.d : $D = \{x_i | i = 1 \dots n\}, P(D|\theta) = \prod_i P(x_i \mid \theta)$

Vocab:

- **likelihood function** $L(\theta|O)$ is called as the likelihood function. θ = unknown parameters, O is the observed outcomes. The likelihood function is conditioned on the observed O and that it is a function of the unknown parameters θ. Not a probability density function.
- **"likelihood" vs "probability"**: if discrete, $L(\theta|O) = P(O|\theta)$. If continuous, $P(O|\theta) = 0$ so instead we estimate θ given O by maximizing $L(\theta|O) = f(O|\theta)$ where f is the pdf associated with the outcomes O.
- **hypothesis space**

Law of Total Probability (LOTP)

Let $B_1, B_2, B_3, ...B_n$ be a *partition* of the sample space (i.e., they are disjoint and their union is the entire sample space).

$$P(A) = P(A|B_1)P(B_1) + P(A|B_2)P(B_2) + \cdots + P(A|B_n)P(B_n)$$
$$P(A) = P(A \cap B_1) + P(A \cap B_2) + \cdots + P(A \cap B_n)$$

For **LOTP with extra conditioning**, just add in another event C!

$$P(A|C) = P(A|B_1, C)P(B_1|C) + \cdots + P(A|B_n, C)P(B_n|C)$$
$$P(A|C) = P(A \cap B_1|C) + P(A \cap B_2|C) + \cdots + P(A \cap B_n|C)$$

Special case of LOTP with B and B^c as partition:

$$P(A) = P(A|B)P(B) + P(A|B^c)P(B^c)$$
$$P(A) = P(A \cap B) + P(A \cap B^c)$$

Bayes' Rule

Bayes' Rule, and with extra conditioning (just add in C!)

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

$$P(A|B, C) = \frac{P(B|A, C)P(A|C)}{P(B|C)}$$

We can also write

$$P(A|B, C) = \frac{P(A, B, C)}{P(B, C)} = \frac{P(B, C|A)P(A)}{P(B, C)}$$

Odds Form of Bayes' Rule

$$\frac{P(A|B)}{P(A^c|B)} = \frac{P(B|A)}{P(B|A^c)} \frac{P(A)}{P(A^c)}$$

The *posterior odds* of A are the *likelihood ratio* times the *prior odds*.

Practice: What is $P(disease \mid +test)$ if P(disease) = 0.01, P(+ | disease) = 0.99, P(+ | no disease) = 0.01?

Expectation

f(X) probability distribution function of X

X ~ P : X is distributed according to P.

Expected value of f under P : $E_P[f(x)] = \sum_x p(x)f(x)$

E.g. unbiased coin. x = 1, 2, 3, 4, 5, 6. p(X=x) = 1/6 for all x.
$E(X) = \sum_x p(x) \cdot x = (1/6) \cdot [1 + 2 + 3 + 4 + 5 + 6] = 3.5$

Entropy

Always greater than or equal to 0. Zero when outcome is certain. 1 for uniform distribution.
Entropy is based on a pdf, not a list of labels. E.g.
H[1,1,0] → H[2/2, 1/3].
$X \sim P$, $x \in \Omega$

First define **Surprise**: $S(x) = -\log_2 p(x)$
$S(X = \text{heads}) = -\log_2(1/2) = 1$.

Axiom 1 : S(1) = 0. (If an event with probability 1 occurs, it is not surprising at all.)

Axiom 2 : S(q) > S(p) if q < p. (When more unlikely outcomes occur, it is more surprising.)

Axiom 3 : S(p) is a continuous function of p. (If an outcomes probability changes by a tiny amount, the corresponding surprise should not change by a big amount.)

Axiom 4 : S(pq) = S(p) + S(q). (Surprise is additive for independent outcomes.)

Surprise of 7 = pretty surprised. Probability of $1/2^7$ of happening (Shannon) **Entropy**:

$$\begin{aligned} H[X] &= -\sum_x p(x) \cdot \log_2 p(x) \\ &= -\sum_x p(x) S(x) \\ &= E[S(x)] \end{aligned}$$

The entropy is the expectation of the surprise. Throw out x for $p(x) = 0$ because log(0) is ∞.

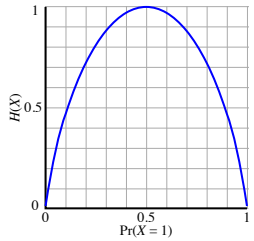
Binary Entropy Function: $p(X = 1) = \theta$ and $p(X = 0) = 1 - \theta$

$$\begin{aligned} H(X) &= -[p(X = 1) \log_2 p(X = 1) + p(X = 0) \log_2 p(X = 0)] \\ &= -[\theta \log_2 \theta + (1 - \theta) \log_2 (1 - \theta)] \end{aligned}$$

Entropy of an unbiased coin flip:
X is a coin flip. $P(X = \text{heads}) = 1/2$, $P(X = \text{tails}) = 1/2$
Note: $\log_2(1/2) = -1$, $-\log_2(1/2) = \log_2(2) = 1$
$H[X] = -[1/2 \log_2(1/2) + 1/2 \log_2(1/2)] = 1$

Entropy of a coin that always flips to heads:
$P(X = \text{heads}) = 1$, $P(X = \text{tails}) = 0$
Note: $\log_x(0) = 0$
$H[X] = -[1 \log_2(1) + 0] = 0$
No surprise: you are sure what you are going to get.

Binary entropy plot.



Canonical example:

X	Y
0	1
1	0
1	1

If you want to estimate entropy of X, you can use P(X=0).

$$\begin{aligned} H[X] &= -[\frac{1}{3} \log_2 \frac{1}{3} + \frac{2}{3} \log_2 \frac{2}{3}] \\ &= \frac{1}{3} \log_2 3 + \frac{2}{3} \log_2 3 - \frac{2}{3} \log_2 2 \\ &= \log_2 3 - \frac{2}{3} \approx 0.91 \end{aligned}$$

This time H[X] = H[Y] because of symmetry.
The discrete distribution with maximum entropy is the uniform distribution. For K values of X, $H(X) = \log_2 K$
Conversely, the distribution with minimum entropy (which is zero) is any delta-function that puts all its mass on one state. Such a distribution has no uncertainty.

Conditional Entropy

If you don't know x: (this is kind of an average).
$H[Y \mid X = x] = -\sum_y P(Y = y \mid X = x) \cdot \log_2 P(y \mid X = x)$
$H[Y \mid X = x] = E[S(Y \mid X = x)]$
Note that we are summing over y because we are specifying x.

For a particular value of X:
$H[Y \mid X] = \sum_x p(x) H[Y \mid X = x]$

Back to table above:

$$H[Y \mid X = 0] = ?$$
look only at X=0 in table.
$$= -[0 + 1 \log_2]$$

Now that you know X=0, entropy goes to 0.

$H[Y \mid X = 1] = 1$: You know *less* if you know X=1.

Now use $H[Y \mid X] = \frac{1}{3}(0) + \frac{2}{3}(1) = 2/3$
Given X, you know more. Average our the more certain case and the less certain case.

Note: $H[Y \mid X] \leq H[Y]$: knowing something can't make you know less.

Entropy and Information Gain

Information Gain - $IG(X) = H(Y) - H(Y \mid X)$
Y is the node on top. X are the nodes below. He might have used lower case.
Class Example: If X_1 is a node for a split, and you want to know the information gain for that node, you:

- calculate entropy of the split. Find Entropy of each branch of the split, and the fraction of points that were channeled to each split. E.g.

$$\{T, T, T, T, T, F\} \rightarrow \{T, T, T, T\} \text{ (for } X_1 = T), \{T, F\} \text{ (for } X_1 = F)$$
$$\rightarrow P(X_1 = T) = 4/6, P(X_1 = F) = 2/6$$
$$\rightarrow H(X_1 = T) = (1 * \log_2 1 + 0 * \log_2 0) = 0$$
$$\rightarrow H(X_1 = F) = \frac{2}{6}(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2})$$
$$= 1 \text{ (uniform distribution)}$$
$$H(Y|X_1) = -\frac{4}{6}(1 * \log_2 1 + 0 * \log_2 0) - \frac{2}{6}(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2})$$
$$= 2/6$$

- find the entropy of the unsplit data:
  $\{T, T, T, T, T, F\} \rightarrow -(5/6)\log_2(5/6) - (1/6)\log_2(1/6) = 0.65$
- subtract the weighted average of the split entropies from the original: $IG(X_1) = H(Y) - H(Y|X_1) = 0.65 - 0.33$

- Low uncertainty $\leftrightarrow$ Low entropy.
- Lowering entropy $\leftrightarrow$ More information gain.

Bits

If you use log base 2 for entropy, the resulting units are called bits (short for binary digits).
How many things can you encode in 15 bits? 2^{25}.

Decision Trees

Summary:

- One of the most popular ML tools. Easy to understand, implement, and use. Computationally cheap (to solve heurisRcally).
- Uses informaton gain to select attributes (ID3, C4.5,)
- Presented for classification, but can be used for regression and density estimation too
- Decision trees will overfit!!!
- Must use tricks to find simple trees, e.g., (a) Fixed depth/Early stopping, (b) Pruning, (c) Hypothesis testing
- Tree-based methods partition the feature space into a set of rectangles.
- Interpretability is a key advantage of the recursive binary tree.

Pros:

- easy to explain to people
- more closely mirror human decision-making than do the regression and classification approaches
- can be displayed graphically, and are easily interpreted even by a non-expert
- can easily handle qualitative predictors without the need to create dummy variables

Cons:

- trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches
- can be very non-robust. A small change in the data can cause a large change in the final estimated tree

Vocab:

- **classification tree** - used to predict a qualitative response rather than a quantitative one
- **regression tree** - predicts a quantitative (continuous) variable.
- **depth of tree** -the maximum number of queries that can happen before a leaf is reached and a result obtained
- **split** -
- **node** - synonymous with split. A place where you split the data.
- **node purity** -
- **univariate split** - A split is called univariate if it uses only a single variable, otherwise multivariate.
- **multivariate decision tree** - can split on things like A + B or Petal.Width / Petal.Length ¡ 1. If the multivariate split is a conjunction of univariate splits (e.g. A and B), you probably want to put that in the tree structure instead.
- **univariate decision tree** - a tree with all univariate splits/nodes. E.g. only split on one attribute at a time.
- **binary decision tree**
- **argmax** - the input that leads to the maximum output
- **greedy** - at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.
- **threshold splits** -
- **random forest**: an ensemble of decision trees which will output a prediction value. Each decision tree is constructed by using a random subset of the training data.

Protocol:

1. Start from empty decision tree
2. Split on next best attribute (feature).
   - Use something like information gain to select next attribute. $\arg\max_i IG(X_i) = \arg\max_i H(Y) - H(Y|X_i)$
3. Recurse

When do we stop decision trees?

- Dont split a node if all matching records have the same output value
- Only split if all of your bins will have data in them. His words: "none of the attributes can create multiple nonempty children." He also said "no attributes can distinguish", and showed that for the remaining training data, each category only had data for one label. And third, "If all records have exactly the same set of input attributes then dont recurse"

He noted that you might not want to stop splitting just because all of your information nodes have zero information gain. You would miss out on things like XOR.
Decision trees will overfit. If your labels have no noise, the training set error is always zero. To prevent overfitting, we must introduce some bias towards simpler trees. Methods available:

- Many strategies for picking simpler trees
- Fixed depth
- Fixed number of leaves
- Or something smarter

One definition of overfitting: If your data is generated from a distribution $D(X, Y)$ and you have a hypothesis space H:
Define errors for hypothesis $h \in H$: training error = $error_{train}(h)$, Data (true) error = $error_D(h)$. The hypothesis h overfits the training data if there exists an h' such that $error_{train}(h) < error_{train}(h')$ and $error_D(h) > error_{train}(D)$. In plain english, if there is an alternative hypothesis that gives you more error on the training data but less error in the test data then you have overfit your data.

How to Build Small Trees
Two reasonable approaches:

- Optimize on the held-out (development) set. If growing the tree larger hurts performance, then stop growing. But this requires a larger amount of data
- Use statistical significance testing. Test if the improvement for any split it likely due to noise. If so, dont do the split. Chi Square test w/ MaxPchance = something like 0.05.

Pruning Trees
Start at the bottom, not the top. The top is most likely to have your best splits. In this way, you only cut high branches if all the branches below were cut.
Don't use the validation set for pruning. **Your code should never use the validation set.** The validation set is for **you** to learn from; the code will always learn from the training set.
Classification vs. Regression Trees
In class we mostly discussed nodes with categorical attributes. You can have continuous attributes (see HW1). You can also have either discrete or continuous output. When output is discrete, you can chose your splits based on entropy. If it is continuous, you need to do something more like least squares. For regression trees, see pg 306 from ISL or pg 307 of ESLII.
For discrete data:
"For discrete data, you can't split twice on the same feature. Once you've moved down a branch, you know that all data in that branch has the same value for the splitting feature."
For continuous data:

More computationally expensive than discrete data. Often can try to change continuos data to categorical. Might lose some smoothness for real numbers, but might be worth it
K-fold validation versus using a held-out data set:
If you have enough data to pull out a held-out set, that is preferable to K-fold validation.

Maximum Likelihood & Maximum a Posteriori

Vocab

- **likelihood**: the probability of the data given a parameter. E.g. $P(D|\theta)$ (for discrete like Binomial). Need not a pdf; need not be normalized.
- **log-likelihood**: lower-case: $l(\theta|x) = \log L(\theta|x)$
- **maximum likelihood** (ML):
- **MLE**: Maximum Likelihood Estimation.
- **PAC**: Probability Approximately Correct.
- **Posterior**: the likelihood times the prior, normalized

MLE: Maximum Likelihood Estimation
Choose θ to maximize probability of D.
Set derivative of __ to zero and solve. If function is multivariate, set each partial derivative to zero and solve.
$\hat{\theta} = \arg\max_\theta P(D|\theta) = \arg\max_\theta \ln P(D|\theta)$
Note we are using ln, not $\log_2$ as we did for entropy above. Want it to cancel exponents now.

Binomial Distribution
Assumes i.i.d: $D = \{x_i | i = 1 \dots n\}, P(D|\theta) = \prod_i P(x_i \mid \theta)$.
Likelihood function: $P(D|\theta) = \theta^{\alpha_H}(1-\theta)^{\alpha_T}$
P(heads) $= \theta$, P(tails) $= 1 - \theta$

$$\hat{\theta} = \arg\max_\theta \ln P(D|\theta)$$
$$= \arg\max_\theta \ln \theta * \alpha H (1-\theta)^{\alpha T}$$

Find optimal theta by setting the derivative to zero:

$$\frac{d}{d\theta} \ln P(D|\theta) = \frac{d}{d\theta} \ln \theta * \alpha H (1-\theta)^{\alpha T}$$
$$= \arg\max_\theta \ln \theta * \alpha H (1-\theta)^{\alpha T}$$
$$= \dots = \frac{\alpha_H}{\alpha_H + \alpha_T}$$

For Binomial, there is exponential decay in uncertainty with # of observations. You can also find the probability that you are approximately correct (see notes).
$P(|\widehat{\theta} = \theta * | \geq \epsilon) \leq 2e^{-2N\epsilon^2}$. Can calculate N (# of flips) to have error less than ϵ with probability of being incorrect δ. Your sensitivity depends on your problem; error on stock market data might cost billions.
What if you had prior beliefs? Use MAP instead of MLE.

Bayesian Learning

Inferring the probability of the parameters themselves, not the probability of the data. Whenever you see $P(\theta|D)$ you know that is some posterior distribution. That is a tidy way of representing your knowledge about θ and your uncertainty about that knowledge. (The uncertainty is held in the PDF; narrow = certain and flat = uncertain).

Rather than estimating a single θ, we obtain a distribution over possible values of θ.

For small sample size, prior is important!

Use Bayes' Rule: $P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$

- **Posterior**: $P(\theta|D)$. Note $P(\theta|D) \propto P(D|\theta)P(\theta)$
- **Data Likelihood**: $P(D|\theta)$
- **Prior**: $P(\theta)$
- **normalization**: $P(D)$. Just a constant so it doesn't matter. Hard to calculate anyway.

Or equivalently, $P(\theta|D) \propto P(D|\theta)P(\theta)$. **Always use this form**, not the one with $P(D)$ in the denominator.

Note: you are multiplying two PDFs here. When you plug in particular data, your two terms become numbers.

As you get more and more data, $P(\theta|D)$ grows more and more narrow. Like with more cannon ball holes, you are more certain about your angle θ.
Murphy 2012 pg 70: In general, when we have enough data, the posterior $p(h|D)$ becomes peaked on a single concept, namely the MAP estimate, i.e., $p(h|D) \to \delta_{\hat{h}MAP}(h)$ where $\delta_{\hat{h}MAP} = \arg\max_h p(h|D)$ is the posterior mode, and wehre δ is the Dirac measure defined by $\delta_x(A) = 1$ if $x \in A$ and $= 0$ if $x \notin A$

About the $P(D)$. It is the "marginal probability", which is basically the probability of D when you integrate out θ.

For uniform priors, MAP reduces to MLE objective. $P(\theta) \propto 1$ leads to $P(\theta|D) \propto P(D|theta)$

If you have a uniform prior, you just do MLE.
$P(\theta) \propto 1 \to P(\theta|D) \propto P(D|\theta)$

Note: if you have D first it is Likelihood, and if you have θ first it is the Posterior. $(P(D|\theta)\ P(\theta|D))$.
Vocab

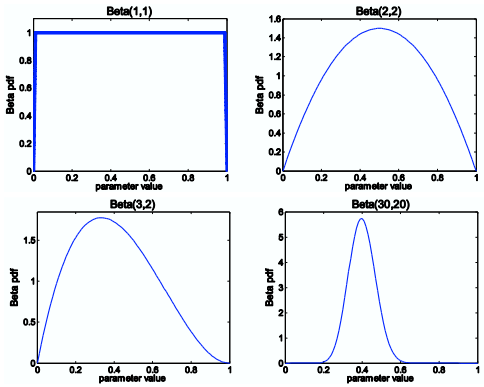
- **prior**:
- **prior distribution**: (same as "prior")
- **posterior**:
- **posterior distribution**: (same as "posterior")
- **Maximum likelihood**: Find the parameter that makes the probability highest. E.g. θ for coin toss. (A famous "point estimator")
- **MAP**: Maximum a posteriori (estimation). Maximize the posterior instead of the likelihood. Take the value that causes the highest point in the posterior distribution.
  Just take the peak of your posterior. Forget about the uncertainty. Pretty much like MLE, but you also have some influence of a prior.
- **conjugate**: If our posterior is a distribution that is of the same family as our prior, then we have conjugacy. We say that the prior is conjugate to the likelihood.
- **conjugate model**: great because we know the exact distribution of the posterior so we can easily simulate or derive quantities of interest analytically. In practice, we rarely have conjugacy.

Likelihood	Prior	Posterior
Binomial	Beta	Beta
Negative Binomial	Beta	Beta
Poisson	Gamma	Gamma
Geometric	Beta	Beta
Exponential	Gamma	Gamma
Normal (mean unknown)	Normal	Normal
Normal (variance unknown)	Inverse Gamma	Inverse Gamma
Normal (mean and variance unknown)	Normal/Gamma	Normal/Gamma
Multinomial	Dirichlet	Dirichlet

Thumbtack Problem, Bayesian style (MAP)
Start as usual with Bayes' without $P(D)$: $P(\theta|D) \propto P(D|\theta)P(\theta)$.
Define parameters: θ is the probability of one side up. α_H and α_T are the number of heads and tails tossed. β_H and β_T are the parameters of the prior. These "beta prior parameters" can be thought of as "fake counts" in the case of the beta distribution.

- use Binomial as the likelihood: $P(D|\theta) = \theta^{\alpha_H}(1-\theta)^{\alpha_T}$.
  Note: we are estimating θ after seeing α_H heads and α_T tails.
- the prior is $P(\theta) = \frac{\theta^{\beta_H}(1-\theta)^{\beta_T - 1}}{B(\beta_H, \beta_T)} \sim \text{Beta}(\beta_H, \beta_T)$.
  Now β_H and β_T are the number of heads and tails you expected to see. The B in the denominator is for the beta function (not same as beta distribution).
- To get a simple posterior form, use a conjugate prior.
  Conjugate prior of Binomial is the Beta Distribution. See slides for math: $P(\theta|D) \sim \text{Beta}(\beta_H + \alpha_H, \beta_T, \alpha_T)$
- note that there are similar terms in the prior and likelihood functions. Some will cancel out when you multiply them.
- $P(\theta|D) = \frac{\theta^{\beta_H + \alpha_H - 1} \cdot (1-\theta)^{\beta_T + \alpha_T - 1}}{B(\beta_H + \alpha_H, \beta_T + \alpha_T)} \sim$
  $\text{Beta}(\beta_H + \alpha_H, \beta_T + \alpha_T)$.
  So your probability is shaped by the number of heads/tails you expected to see (β_H, β_T) and the number of heads/tails you actually saw (α_H, α_T).
- Apply MAP: $\widehat{\theta} = \arg\max_\theta P(\theta|D) = \frac{\alpha_H + \beta_H - 1}{\alpha_H + \beta_H + \alpha_T + \beta_T - 2}$.
  Effectively, our prior is just adding $\beta_H - 1$ heads flips and $\beta_T - 1$ tail flips to the dataset.
- The Beta prior is equivalent to extra thumbtack flips. As $N \to \infty$, the prior is forgotten. But for small sample size, prior is important.

Beta Distribution



MAP (point) estimation:

1. Chose a distribution to fit the data to. Your choice determines the form of the likelihood ($P(\theta|D)$).
2. Chose a prior (distribution). Can use a table that shows conjugate priors for various distributions. Prior is over the parameters you are guessing.
3. Now you have a posterior (multiply prior by likelihood).
4. Plug in your particular data values under many values of θ to get the likelihood ($P(D|\theta)$). Recall the likelihood need not be a PDF (need not be normalized).
5. Pick the value that causes the highest point on the peak.

MAP estimation
Closely related to Fisher's method of maximum likelihood (ML), but employs an augmented optimization objective which incorporates a prior distribution over the quantity one wants to estimate. You get to pick the distribution to represent the prior. MAP estimation can therefore be seen as a regularization of ML estimation. (Another famous "point estimator")

Chosing between MLE and MAP:
Chose ML if you don't know enough about the domain to impose a new prior.

If you are measuring a continuous variable, Gaussians are your friend.

Gaussians

Properties of Gaussians:

- Affine transformation (multiplying by a scalar and adding a constant) are Gaussian. If X $\sim$ N(μ,σ^2) and Y = aX + b, then Y $\sim$ N($a\mu + b, a^2\sigma^2$)
- Sum of Gaussians is Gaussian. If X $\sim$ N(μ_X, σ_X^2), Y $\sim$ N(μ_Y, σ_Y^2), and Z = X+Y, then Z $\sim$ N($\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2$)
- Easy to differentiate.

Learn a Gaussian: $P(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$.
MLE for Gaussian: Prob of i.i.d. samples D = $\{x_1, \ldots, x_N\}$:
$P(D|\mu,\sigma) = (\frac{1}{\sigma\sqrt{2\pi}})^N \prod_{i=1}^{N} e^{\frac{-(x_i-\mu)^2}{2\sigma^2}}$.
Note: it is not $P(\mu,\sigma|D)$, like I thought in class.
Find μ_{MLE}, $\sigma_{MLE} = \arg\max_{\mu,\sigma} P(D|\mu,\sigma)$.
Log-likelihood:

$$\ln P(D|\mu,\sigma) = \ln[\text{thing above}] = -N \ln \sigma\sqrt{2\pi} - \sum_{i=1}^{N} \frac{(x_i-\mu)^2}{2\sigma^2}.$$

Differentiate w.r.t. μ and set = 0. End up with $\widehat{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i$.

Differentiate w.r.t. σ and set = 0. End up with
$\widehat{\sigma}^2_{MLE} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \widehat{\mu})^2$.
But actually, that leads to a biased estimate, so people actually use
$\widehat{\sigma}^2_{unbiased} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \widehat{\mu})^2$
The conjugate priors: mean: use Gaussian prior:
$P(\mu|\nu,\lambda) = \frac{1}{\lambda\sqrt{2\pi}} e^{\frac{-(\mu-\nu)^2}{2\sigma^2}}$. (Instead of σ, use λ and replace the $(x-\mu)^2$ with $(\mu-\nu)^2$).
For variance: use Wishard Distribution:

Linear Regression

Ordinary Least Squares
Notation:

- x_i: an input data point. __ rows by __ columns.
- y_i: a predicted output
- $\widehat{y_i}$: a predicted output
- $\widehat{y}$:
- w_k: weight k
- $\boldsymbol{w}^*$: the vector of weights found in regression.
- $f_k(x_i)$
- t: what we want to regress against
- t_j: the output variable that you either have data for or are predicting.
- $t(\boldsymbol{x})$: Data. "Mapping from x to t(x)"
- H: $H = \{h_1, \ldots, h_K\}$. Basis functions. In the simplest case, they can just be the value of an input variable/feature or a constant (for bias).
- L_2: The L_2. Can appear as a loss function to describe the deviation from data or as a penalty.
- $||\widehat{w}||_1$: "L_1" penalty. The "Manhattan distance". Like traveling a, b in a pythagorean triangle. $\sum |x_i|$
- $||\widehat{w}||_2$. "L_2" penalty. Euclidean length of a vector. Like c in a pythagorean triangle. $\sqrt{\sum |x_i|^2}$

Vocab:

- **bias-variance tradeoff** - the problem of simultaneously minimizing two sources of error that prevent supervised learning algorithms from generalizing beyond their training set.
  - The bias is error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting).
  - The variance is error from sensitivity to small fluctuations in the training set. High variance can cause overfitting: modeling the random noise in the training data, rather than the intended outputs.
- **basis function**
- **bias** (parameter): like the intercept in a linear equation. The part that doesn't depend on the features.
- **bias** (learning bias): (?? "inductive bias" ??) -
- **hyperplane** - a plane, usually with more than 2 dimensions.
- **input variable** - a.k.a. feature. E.g. a column like CEO salary for rows of data corresponding to different companies.
- **response variable** - synonyms: "dependent variable", "regressand", "predicted variable", "measured variable", "explained variable", "experimental variable", "responding variable", "outcome variable", and "output variable". E.g. a predicted stock price.
- **regularization** - introducing additional information in order to solve an ill-posed problem or to prevent overfitting. E.g. applying a penalty for large parameters in the model.
- **ridge regression** -
- **vector norm**: put in a vector and get out a number like length or size. Real valued function of some sort of vector or matrix quantity.
- **hyperparameters**:
  λ, not w. Parameters that control your actual parameters.

  In Bayesian analysis, the parameters that don't touch the data. Like the parameters for the prior on the prior. Called the ridge regression λ a hyperparameter, though this is a stretch in the terminology.
  Class version:
- **feature selection**: explicitly select features that can go into your model instead of throwing all features in.
- **loss function**: A function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. An optimization problem seeks to minimize a loss function.
  $\sum_j (t(x_j) - \sum w_i h(x_i))^2$. (least-squares error ($L_2$)) (??? = training error??)
- **training set error**: *doesn't include the regularization penalty!*. A.k.a. "training error". Sum of squares error divided by the number of points. See formula later.

Ordinary Least Squares

$$\text{total error} = \sum_i (y_i - \hat{y_i})^2 = \sum_i (y_i - \sum_k w_k f_k(x_i))^2$$

i is for each data point, k is for each of the k basis functions.

Under the additional assumption that the errors be normally distributed, OLS is the maximum likelihood estimator.
?? Use words to describe what subset of regression in general this is. What is ordinary? What are we limiting?

The regression problem:
Given basis functions $\{h_1, \ldots, h_K\}$ with $h_i(\mathbf{x}) \in \mathbb{R}$,
find coefficients $\boldsymbol{w} = \{w_1, \ldots, w_k\}$.
$t(\boldsymbol{x}) \approx \widehat{f}(\boldsymbol{x}) = \sum_i w_i h_i(\boldsymbol{x})$

This is called linear regression b/c it is linear in the parameters. We can still fit to nonlinear functions by using nonlinear basis functions; $f_k \to h_i$ Minimize the **residual squared error**:
$\boldsymbol{w}* = \arg\min_{\boldsymbol{w}} \sum_j (t(\boldsymbol{x}_j) - \sum_i w_i h_i(\boldsymbol{x}_j))^2$
j for each data point, i for the number of weights, which is the number of basis functions.

For fitting a line in 2D space, your basis functions are
$\{h_1(x) = x, h_2(x) = 1\}$. $h_2(x) = 1$ is the (constant) bias basis function. The size of w_2 controls the effect in prediction.

To fit a parabola, your basis functions could be
$\{h_1(x) = x^2, h_2(x) = x, h_3(x) = 1\}$.
Want a 2D parabola? Use
$\{h_1(x) = x_1^2, h_2(x) = x_2^2, h_3(x) = x_1 x_2, \ldots\}$.
Can define any basis functions $h_i(\boldsymbol{x})$ for n-dimensional input
$\boldsymbol{x} =< x_1, \ldots, x_n >$

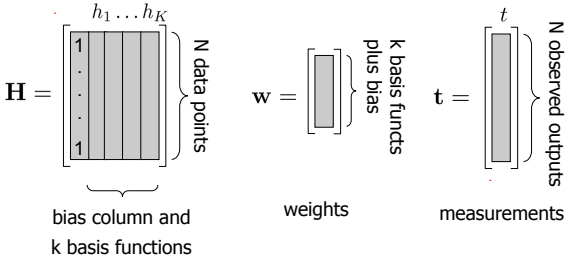
Linear in feature space vs linear in the parameter space:
When we want to fit a parabola, we are still using a function that is linear in parameter space. The weight matrix is still all constants. When we use nonlinear basis functions, we are nonlinear in feature space. The basis functions may be transformations of the input parameters.

Regression: matrix notation

$$\boldsymbol{w}* = \arg\min_{w} \sum_{j} (t(\boldsymbol{x}_j - \sum_{i} w_i h_i(\boldsymbol{x}_j))^2$$

$$\boldsymbol{w}* = \arg\min_{w} (\boldsymbol{Hw} - \boldsymbol{t})^T (\boldsymbol{Hw} - \boldsymbol{t})$$

$(\boldsymbol{Hw} - \boldsymbol{t})^T (\boldsymbol{Hw} - \boldsymbol{t})$ is the residual error.



Regression: closed form solution

$$\boldsymbol{w}^* = \arg\min_{w} (\boldsymbol{Hw} - \boldsymbol{t})^T (\boldsymbol{Hw} - \boldsymbol{t})$$

$$\boldsymbol{F}(\boldsymbol{w}) = \arg\min_{w} (\boldsymbol{Hw} - \boldsymbol{t})^T (\boldsymbol{Hw} - \boldsymbol{t})$$

$$\nabla_{\boldsymbol{w}} \boldsymbol{F}(\boldsymbol{w}) = 0$$

$$2\boldsymbol{H}^T(\boldsymbol{Hw} - \boldsymbol{t}) = 0$$

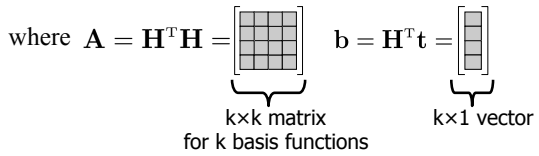
$$(\boldsymbol{H}^T\boldsymbol{Hw}) - \boldsymbol{H}^T\boldsymbol{t} = 0$$

$$\boldsymbol{w}^* = (\boldsymbol{H}^T\boldsymbol{H})^{-1}\boldsymbol{H}^T\boldsymbol{t}$$

Regression solution: simple matrix math

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \underbrace{(\mathbf{Hw} - \mathbf{t})^T(\mathbf{Hw} - \mathbf{t})}_{\text{residual error}}$$

$$\text{solution: } \mathbf{w}^* = \underbrace{(\mathbf{H}^T\mathbf{H})^{-1}}_{\mathbf{A}^{-1}} \underbrace{\mathbf{H}^T\mathbf{t}}_{\mathbf{b}} = \mathbf{A}^{-1}\mathbf{b}$$



Dimensions:

- $\boldsymbol{t}$: N-dimensional (N = # of input data points)
- $\boldsymbol{w}$:
- $\boldsymbol{H}$: k + 1 by N. N is # of rows.

Linear regression prediction is a linear function plus Gaussian noise

We can model as linear combination of basis functions + noise ϵ. It's safe to assume epsilon comes from Gaussian distribution.
$t(\boldsymbol{x}) = \sum_i w_i h_i(\boldsymbol{x}) + \epsilon$
Note: no μ because we set it to zero.

We can learn $\mathbf{w}$ using MLE: $P(t|x, w, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-[t - \sum_i w_i h_i(x)]^2}{2\sigma^2}}$
Take the log and maximize with respect to w: (maximizing log-likelihood with respect to w)

$$\ln P(D|\boldsymbol{w}, \sigma) = \ln(\frac{1}{\sigma\sqrt{2\pi}})^N \prod_{j=1}^{N} e^{\frac{-[t_j - \sum_i w_i h_i(x_j)]^2}{2\sigma^2}}$$

Now find the w that maximizes this:
$\arg\max_w \ln(\frac{1}{\sigma\sqrt{2\pi}})^N + \sum_{j=1}^{N} \frac{-[t_j - \sum_i w_i h_i(x_j)]^2}{2\sigma^2}$
the first term isn't impacted by w so
$= \arg\max_w \sum_{j=1}^{N} \frac{-[t_j - \sum_i w_i h_i(x_j)]^2}{2\sigma^2}$
switch to $\arg\min_w$ when we divide by -1. The numerator is constant.:
$= \arg\min_w [t_j - \sum_i w_i h_i(x_j)]^2$
Least-squares Linear Regression is MLE for Gaussians!!!

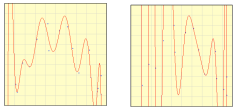
If you have a polynomial you are fitting, how many basis functions are there (??)

OLS Protocol (incomplete)

1. Chose your basis functions h_i. (Requires expertise). Can be nonlinear, e.g. $x_1^2, \sin(x)$, etc.
    - The number of parameters is len(H) + 1 (bias). E.g. for fitting a parabola (formula $y = ax^2 + bx + c$), you have 3 parameters: weights for basis functions x, x^2, and bias. Typically the # of basis functions is < the # of features.
2. Chose a regularization method so your weights don't get too big. (see below)
3. Plug them in to regression to get the weights (w_is). (Form the sum (residual squared error + regularization) that you want to minimize, then minimize.)
4. Make sure your weights aren't too big.

Regularization in Linear Regression

You need to regularize to prevent parameters from growing too large. Both of these were built from the same set of basis functions; the right one is clearly over-fit.



Ridge Regression

Ridge Regression is the most famous form of linear regression Here is our old "ordinary" least squares objective function:

$$\widehat{w} = \arg\min_{w} \sum_{j=1}^{N} [t(x_j) - (w_0 + \sum_{i=1}^{k} w_i h_i(x_j))]^2$$

It is the same as the previous ones but $i = 0$ is pulled out.
Now for ridge regression, we use that same notation.
And we add a penalty term that isn't applied to the bias feature:

$$\widehat{w}_{ridge} = \arg\min_{w} \sum_{j=1}^{N} [t(x_j) - (w_0 + \sum_{i=1}^{k} w_i h_i(x_j))]^2 + \lambda \sum_{i=1}^{k} w_i^2$$

$$= \arg\min_{w} (\boldsymbol{Hw} - \boldsymbol{t})^T (\boldsymbol{Hw} - \boldsymbol{t}) + \lambda \boldsymbol{w}^T I_{0+k} \boldsymbol{w}$$

$$I_{0+k} = \begin{bmatrix} 0 & 0 & 0 & \cdots \\ 0 & 1 & 0 & \cdots \\ 0 & 0 & 1 & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$ (k+1 × k+1)

k+1 x k+1 identity matrix, but with 0 in upper left

That I_{0+k} matrix is this:
Allows you to multiply the whole weight array without getting the bias term in there.
Note: $W^T W$ is w_i^2 or $||w_i||^2$
A similar derivation leads to a closed form solution:
$w^*_{ridge} = (\boldsymbol{H}^T\boldsymbol{H} + \lambda I_{0+k})^{-1}\boldsymbol{H}^T\boldsymbol{t}$
(Recall that un-regularized regression was $w^* = (\boldsymbol{H}^T\boldsymbol{H})^{-1}\boldsymbol{H}^T\boldsymbol{t}$).

How do you chose how large λ is?
* As $\lambda \to 0$, becomes same as MLE: unregularized. Large magnitudes of coefficients.
* As $\lambda \to \infty$, all weights become 0.

Experiment cycle

1. select a hypothesis f to best match the training set. (??) Is this the same as choosing your basis functions (?)
2. isolate a held-out data set if you have enough data, or do K-fold cross-validation if not enough data.
    - tune hyperparameters (λ) on the held-out set or via cross-validation. (Try many values of λ and chose the best one.)
    - You can use the same held-out data set each time if that set is big.
    - If doing K-fold, divide the data into k subsets. Repeatedly train on k-1 and test on the remaining one. Average the results.
    - find the w that minimizes the error. (Do so by taking the derivative and setting = 0); see ridge regression notes.
3. Select basis functions

Regularization options: Ridge & Lasso

Ridge:

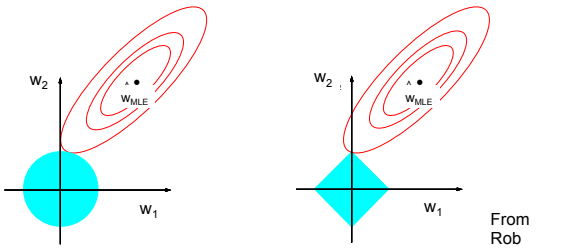
- $\widehat{w}_{ridge} = \arg\min_{w} \sum_{j=1}^{N} [t(x_j) - (w_0 + \sum_{i=1}^{k} w_i h_i(x_j))]^2 + \lambda \sum_{i=1}^{k} w_i^2$
- L_2 penalty. ("L_2 norm of $\boldsymbol{w}$). Large distances get penalized more. $Y = x^2$: don't want errors to cancel each other; differentiable.

Lasso:

- $\widehat{w}_{ridge} = \arg\min_{w} \sum_{j=1}^{N} [t(x_j) - (w_0 + \sum_{i=1}^{k} w_i h_i(x_j))]^2 + \lambda \sum_{i=1}^{k} |w_i|$
- L1 penalty: linear penalty pushes more weights to zero. Allows for a type of feature selection. But it is not differentiable and there is no closed form solution.
- L1 is absolute value: don't need to square it.
- Lasso may be more useful when you have too many features for the amount of data you get. Example: 100k parameters about companies to predict stock prices but only 100 data points. Could tune lambda until you have about 100 nonzero weights.

Your chose of penalty has huge effects on the algorithm!
Sometimes L_1 will do better than L_2 and vice versa, but usually L_2 is more powerful than L_1.
Would be better to use a combination of the penalties than to first reduce the number of features with L_2 before applying L_1.
You can generate a lot of basis functions and use L_1 to chose the good ones.

Geometric Intuition



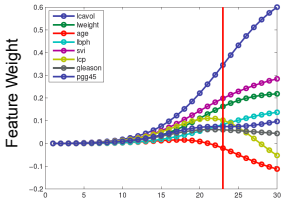
Ridge Regression Lasso

From Rob Tibshirani slides

This figure shows:

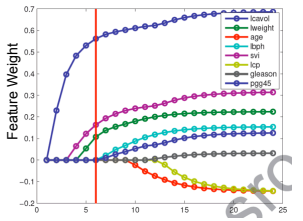
- The contour lines represent the maximum likelihood of the vector of weights. All points on the contour have equal likelihood.
- The two axes represent different parameters for two of the weights. (regression coefficients)
- Circles are characteristic of ridge regression (with L2 penalty): Penalty = the magnitude of the vector.
- Shapes that are pointy on the axes are characteristic of Lasso (with L1 penalty): the vector components get added.
- Where the likelihood function touches in this w_1, w_2 space represents the coefficients of the weights. For Ridge Regression, we see that small but nonzero values of the coefficients can be obtained. For Lasso Regression, the curves are most likely to touch the diamond on the axes, resulting in coefficients that are truly zero.

Ridge Coefficent Path



Larger λ ← $||\hat{w}||_2$ → Smaller λ

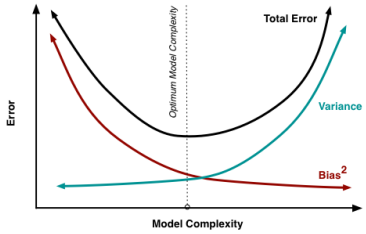
LASSO Coefficent Path



Larger λ ← $||\hat{w}||_1$ → Smaller λ

Don't compare coefficient magnitudes at given λs, but do note that for Ridge the gradually come away from the zero axis and in Lasso they are zero until they pop out.

Bias-Variance Tradeoff



Your choice of hypothesis class (e.g. degree of polynomial) introduces learning bias.
The more complex the model, the more the ____ set accuracy goes down
A more complex class → less bias and more variance.

From Wikipedia:
Ideally, one wants to choose a model that both accurately captures the regularities in its training data, but also generalizes well to unseen data. Unfortunately, it is typically impossible to do both simultaneously. High-variance learning methods may be able to represent their training set well, but are at risk of overfitting to noisy or unrepresentative training data. In contrast, algorithms with high bias typically produce simpler models that don't tend to overfit, but may underfit their training data, failing to capture important regularities.

Models with low bias are usually more complex (e.g. higher-order regression polynomials), enabling them to represent the training set more accurately. In the process, however, they may also represent a large noise component in the training set, making their predictions less accurate - despite their added complexity. In contrast, models with higher bias tend to be relatively simple (low-order or even linear regression polynomials), but may produce lower variance predictions when applied beyond the training set.

Adding features (predictors) tends to decrease bias, at the expense of introducing additional variance.

Solutions:

- Dimensionality reduction and feature selection can decrease variance by simplifying models.
- Similarly, a larger training set tends to decrease variance.
- Use tunable modeling parameters such as applying more significant regularization.

If held-out is too small, we can end up over-fitting.

Error Definitions

True ("Prediction") Error:
Since the training set error can be a poor measure of the "quality" of the solution, we can use prediction error ("true error"). The error over all possibilities. Instead of sum, take expectation.

$$error_{true}(\boldsymbol{w}) = E_X[(t(\boldsymbol{x_j}) - \sum_i w_i h_i(\boldsymbol{x_j}))^2]$$

Gold Standard:

$$error_{true}(\boldsymbol{w}) = \int_x (t(\boldsymbol{x_j}) - \sum_i w_i h_i(\boldsymbol{x_j}))^2 p(\boldsymbol{x}) d\boldsymbol{x}$$

How to get $p(\boldsymbol{x})$? Need to know the true distribution of the data (?). You almost never know how to compute p(x). And, the integral is a very big sum.

Find $p(x)$ using monte-carlo integration. Sample some points, estimate p(x). That leads to a fair approximation.

The true prediction error is the expectation over **future** test cases you don't have. Since you don't have the x values, you go to probability. Pick a point from the distribution, and calculate the _____.

Sampling approximation of predicted error:

$$error_{true}(\boldsymbol{w}) \approx \frac{1}{M} \sum_{j=1}^{M} (t(\boldsymbol{x_j} - \sum_i w_i h_i(\boldsymbol{x_j}))^2$$

Don't use the training data to predict true error; you've already trained to that data! You would have too optimistic of a prediction for true error.
Prediction error is high when the model is too simple and too complex, unlike training set error which only penalizes too simple.

Training Set Error: **optimistically biased** (a.k.a. training error)

$$error_{train}(\boldsymbol{w}) = \frac{1}{N_{train}} \sum_{j=1}^{N_{train}} (t(\boldsymbol{x_j}) - \sum_i w_i h_i(\boldsymbol{x_j}))^2$$

Decreases exponentially with model complexity.
Training error is a poor prediction of prediction error!
You expect to see training error to decrease with complexity, but that doesn't mean you have a good solution!

Test Error: (our final measure)
Uses the same formula as prediction error, except that we have never observed the test data. See formula below. We expect the true error to be smile shaped if the x-axis is model-complexity.

Testing is for the user of your algorithm. The user doesn't care about the values in your model. They just care about how well it works. They dont' care about the value of your hyperparameters.

Effects of λ value on model

- high lambda → simple model → lots of zeros → high error.
- with lambda = 0 → converges ridge regression to regular regression.
- with enough traiing data and lambda = 0, we expect overfitting → small training error.

Choosing λ

How to find lambdas?

- try a bunch and find which does best on the held out data set. Can't touch test data, even w/ stick so use held-out set.
- Pick lambda that gives minimum error. May not find the bottom-of-the-U shape lambda. But it is ok to be off a bit; the red U might be pretty flat at the bottoms.

This use of the held-out data doesn't count as training. Lambda is fixed each time, and we train separately. Training on training data, just watching the number on the held-out set.
The model's parameters are not determined by the held-out data

How do you chose the range of lambda? (practical solution)

- You are limited to a grid search over values of lambda.
- You need a strategy for sweeping that space. Could do a "binary search", or something like a gradient search: take a step until we screw it up, and step back smaller. If it gets worse, you take a smaller step.

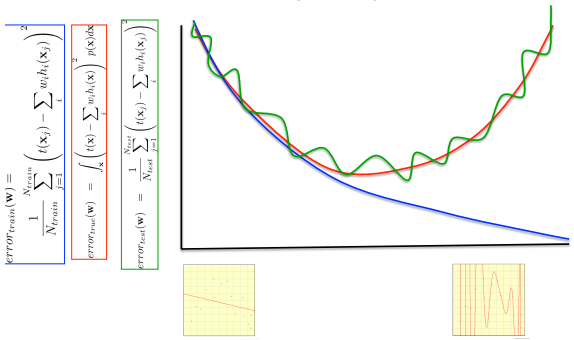
- From your loss, take the value of your loss (compute it). Loss $= (t(x) - \sum w_i h(x_i))^2$ **"You should be able to find loss given w** Can compute norm of W, too. If your loss is on the order of 1000 and your norm is on the order of 1, you can use these for defining search space. Firs try lamba = 1, 10, 100, 1000. Find the best from there then explore around there. If 100 was good, try 200, etc. You just want to know in which order the loss function is going to appear.

How to handle error calculations:

Given a dataset, randomly split it into two parts:
* training data: $\{\boldsymbol{x_1}, \ldots, \boldsymbol{x_{N_t rain}}\}$
* test data: $\{\boldsymbol{x_1}, \ldots, \boldsymbol{x_{N_t est}}\}$
Use the training data to optimize parameters $\boldsymbol{w}$.
To calculate the test set error, you use the final solution $\boldsymbol{w}^*$ and calculate

$$error_{test}(\boldsymbol{w}) \approx \frac{1}{N_{test}} \sum_{j=1}^{N_{test}} (t(\boldsymbol{x_j} - \sum_i w_i h_i(\boldsymbol{x_j}))^2$$

Test set error as a function of model complexity

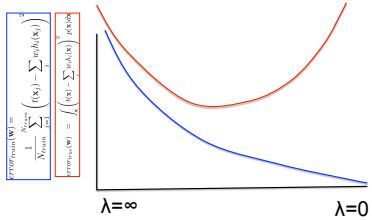


(blue is train, red is true, green is test)

overfitting

Assume:
* Data generated from distribution D(X,Y).
* A hypothesis space H
Define:
* Training error: $error_{train}(h)$
* Data (true) error: $error_{true}(h)$
We say h **overfits** the training data if there exists an $h' \in H$ such that:
* $error_{train}(h) < error_{train}(h)'$ and $error_{true}(h) > error_{true}(h)'$

Error as function of regularization parameter, fixed model complexity



Lambda (the regularization parameter) is fixed. Blue = training set error. Leads to overfitting/too-large-parameters when λ –¿ 0. Red = true error, which likes a happy medium λ.

Warning: your test set is only unbiased if you never ever ever do **any** learning on the data. This includes using the test data to select for the degree of the polynomial you fit.

(Recall, you can create a held-out/validation set from your training data or do k-fold validation.)

The height of the true error (red curve) is the "bias".

What you need to know

Regression:

- Basis functions = features
- Optimizing a sum of squared errors
- Relationship between regression and Gaussians

Regularization

- Ridge regression math
- LASSO formulation
- How to set lambda

Bias-Variance trade-off

Naive Bayes

Vocabulary

- $h_{NB}(x)$ the function that returns the best class.
- .
- .
- .

Advantages:

- Fast to train (single scan). Fast to classify
- Not sensitive to irrelevant features
- Handles real and discrete data
- Handles streaming data well

Disadvantages:

- Assumes independence of features
- All observations have equal weight in prediction.

Vocab:

- **prior**: $P(Y)$, the probability of a label/class.
- **Likelihood**: $P(\boldsymbol{X}|Y)$.

Conditional independence

X is conditionally independent of Y given Z if the probability distribution for X is independent of the value of Y, given the value of Z:
$(\forall i, j, k)\ P(X = i|Y = j|Z = k) = P(X = i|Z = k)$.
E.g. P(Thunder | Rain, Lightening) = P(Thunder | Lightning)
Equivalent to P(X, Y | Z) = P(X | Z) P(Y | Z)

TODO: put in plain english.

Naive Bayes Assumption

Features are independent given the class:

$$P(X_1, X_2|Y) = P(X_1|X_2, Y)P(X_2|Y)$$
$$= P(X_1|Y)P(X_2|Y)$$
more generally:
$$P(X_1, \ldots, X_n|Y) = \prod_i P(X_i|Y)$$

This reduces the number of parameters a lot! Say you had 5 features. Before this assumption, each of your 5 features could be dependent. Then you have to assign a probability to each state. If each is binary, then you can say 2^5. After this assumption, you would just have 5 parameters.

Naive Bayes Classifier

Given:

- a prior $P(Y)$
- n conditionally independent features $\boldsymbol{X}$ given the class Y
- calculated likelihood for each X_i of the form $P(X_i|Y)$

Your decision rule is: (note h_{NB} is Naive Bayes, not Neg Binom)

$$y^* = h_{NB}(x) = \arg\max_y P(y)P(x_1, \ldots, x_n|y)$$
$$= \arg\max_y P(y) \prod_i P(x_i|y)$$

Although the assumption that the predictor (independent) variables are independent is not always accurate, it does simplify the classification task dramatically, since it allows the class conditional densities $p(x_k|C_j)$ to be calculated separately for each variable, i.e., it reduces a multidimensional task to a number of one-dimensional ones. In effect, Naive Bayes reduces a high-dimensional density estimation task to a one-dimensional kernel density estimation. Furthermore, the assumption does not seem to greatly affect the posterior probabilities, especially in regions near decision boundaries, thus, leaving the classification task unaffected.

Nave Bayes is NOT sensitive to irrelevant features. However, this assumes that we have good enough estimates of the probabilities, so the more data the better.

Digit classification example

Simplify images of digits to pixels, and assign them True or False for whether they are "on".
Each input maps to a feature in a vector. E.g. pixel in the 0th for and 0th column is $F_{0,0}$.
The Naive Bayes model is:
$P(Y|F_{0,0}, \ldots, F_{15,15}) \propto P(Y) \prod_{i,j} P(F_{i,j}|Y)$. We assume the features are independent given the class. We need to learn the distribution of pixels on at each pixel given each number.
How to calculate the prior, P(Y):
$P(Y) = \frac{count(Y = y)}{\sum_{y'} Count(Y = y'}$ (denominator is summing over all y values)
How to calculate the likelihood:
$P(X_i = x|Y = y) = \frac{Count(X_i = x, Y = y)}{\sum_{x'} Count(X_i = x', Y = y)}$

For binary features, use the Beta prior and MAP.

Just like likeihood of binomial previously!
$P(\theta|D) = \frac{\theta^{\beta_H+\alpha_H-1}(1-theta)^{\beta_T+\alpha_T-1}}{B(\beta_H+\alpha_H, \beta_T+\alpha_T)} \approx Beta(\beta_H+\alpha_H, \beta_T+\alpha_T)$
Chose θ using MAP:
$\widehat{\theta} = \arg\max_\theta P(\theta|D) = \frac{\alpha_H+\beta_H-1}{\alpha_H+\beta_H+\alpha_T+\beta_T-2}$.
Once again, the Beta prior is equivalent to adding extra observations for each feature.
If you don't have a lot of observations, the prior is important.
And as the number of observations goes to ∞, the prior is "forgotten".

Multinomials: Laplace Smoothing

Laplace's estimate:
Pretend you saw every outcome k extra times:
$P_{LAP,k}(x) = \frac{c(x)+k}{N+k|X|}$
N = number of observations.
$|X|$ = the number of categories you are counting.
k is the strength of the prior.

Example: $P_{LAP,0}(X) = \langle\frac{2}{3}, \frac{1}{3}\rangle$.
Set $k = 1$. $|X|$ is 2. N is 3. $P_{LAP,1}(X) = \langle\frac{3}{5}, \frac{2}{5}\rangle$
$P_{LAP,100}(X) = \langle\frac{102}{203}, \frac{101}{203}\rangle$
Laplace for conditionals: Smooth each condition independently:
$P_{LAP,k}(x|y) = \frac{c(x,y)+k}{c(y)+k|X|}$

Subtleties of the NB classifier

(1) Usually the features are not conditionally independent:
$P(X_1, \ldots, X_n|Y) \neq \prod_i P(X_i|Y)$
The actual probabilities $P(Y|\boldsymbol{X})$ are often biased towards 0 or 1.
Nonetheless, NB is the single most used classifier out there. It performs well even when the independence assumption is violated.
(2) Overfitting
Conditional probabilities can easily be calculated as zero. Zero probabilities kill that class' chance at being called.
??? If the feature is binary, we can use MAP with a beta prior. ???
That's equivalent to adding extra observations for each feature.

NB for text classification

- i is the i^{th} word
- NB assumption(___) helps a lot. $P(X_i = x_i|Y = y)$ is just the probability of observing word x_i in a document on topic y.
  $h_{NB}(x) = \arg\max_y P(y) \prod_{i=1}^{LengthDoc} P(x_i|y)$
- Additional assumption: bag of words model.
  Order of words ignored. Works really well.
  $P(X_i = x_i|Y = y) = P(X_k = x_i|Y = y)$ (k is the k^{th} word (?); all positions have the same distribution).
  $P(y) = \prod_{i=1}^{LengthDoc} P(x_i|y)$
- Prior, $P(Y)$, is the fraction of documents of each topic.
- Likelihood, $P(X_i|Y)$ is count for how many times you saw the word in documents of this topic. This distribution is shared across all positions i.
- Testing: Use Naive Bayes decision rule.
  $h_{NB}(x) = \arg\max_y P(y) \prod_{i=1}^{LengthDoc} P(x_i|y)$

NB for continuous X_i

- k is an index over all possible labels.
- i is the i^{th} feature. Here it is the pixel.
- j is the j^{th} training example.
- X_i^j is the i^{th} pixel in the j^{th} training sample.
- Y^j is the label corresponding to the j^{th} training example.
- y_k is the k^{th} label
- j is j^{th} training example.
- $\delta(x) = 1$ if x true, else 0.
- h: the function that returns the best class.
- in plain english, ------

Example: character recognition where the darkness of each pixel is continuous.
Gausian Naive Bayes (GNB):
$P(X_i = x|Y = y_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} e^{\frac{-(x-\mu_{ik})^2}{2\sigma_{ik}^2}}$

- μ_{ik} is the mean of the values for the i^{th} feature for the k^{th} class.
- σ_{ik} is the standard deviation of the values for the i^{th} feature for the k^{th} class.

Sometimes we assume one or both of these:

- variance is independent of Y (i.e. σ_i)
- variance is independent of X_i (i.e. σ_k)

If we assume both, we assume just one σ without subscripts.

Estimating parameters for discrete Y and continuous X_i:

- **mean**: $\widehat{\mu}_{ik} = \frac{1}{\sum_j \delta(Y^j = y_k)} \sum_j X_i^j \delta(Y^j = y_k)$
  - first term: divide by the number of training examples that are of class k.
  - second term: summing the continuous input of pixel i for all examples in the training set that match label k.
  - so this is just an average brightness for pixel i given class k using all the training data.
- **variance**: $\widehat{\sigma}_{ik}^2 = \frac{1}{\sum_j \delta(Y^j = y_k) - 1} \sum_j (x_i^j - \widehat{\mu}_{ik})^2 \delta(Y^j = y_k)$
  - first term: divide by the number of training points of class k minus 1.
  - second term: sum the squared difference in brightness of pixel i compared to the mean for that pixel and label.

When Bayes Classifier is Optimal:

In Bayes we are learning the function h that produces labels Y based on inputs $\boldsymbol{X}$. More formally:
$h : \boldsymbol{X} \mapsto Y$, or
we are learning "the function h that maps features $\boldsymbol{X}$ to labels Y".

If you know the true $P(Y|\boldsymbol{X})$, then
$h_{Bayes} = \arg\max_y P(Y = y|\boldsymbol{X} = x)$.

Note the subscript is Bayes, not Naive Bayes; no assumption of conditional independence. Also, the conditionality is back to likelihood instead of posterior.

Theorem: Bayes (not NB) classifier h_{Bayes} is optimal.
$error_{true}(h_{Bayes}) \leq error_{true}(h)$, $\forall h$

This is theoretical result: we don't know $P(\boldsymbol{x})$. We can't calculate the true Bayes classifier b/c we don't know the distribution of all the data.) We also don't know $P(Y|\boldsymbol{X})$, the true class' highest probability. Usually that's hidden; if we knew it we would go home happy.
Plain english: the predictions you get from Bayes are better than any other function/prediction available.

Proof:

$$P_h(error) = \int_x P_h(error|\boldsymbol{X})P(\boldsymbol{X})$$

def. of error: $P_h(error|\boldsymbol{x}) = \int_y \delta(h(\boldsymbol{X}), Y)P(Y|\boldsymbol{X})$

(note, usually we'd sum over the classes, Y)

$$= \int_x \int_y \delta(h(\boldsymbol{X}), Y)P(Y|\boldsymbol{X})P(\boldsymbol{X})$$

(the double integral is zero when $P(Y|\boldsymbol{X})$ is largest, which is when the correct classification was selected.)

We are averaging over novel data sets that are generated under the same conditions.
Different notation: delta has a comma in the parentheses and not an equality.

- $P_h(error)$ is probability of error across all classifications.
- $\delta(h(\boldsymbol{X}, Y)$ is 1 if your X is classified right and 0 if not.
- $P_h(error|\boldsymbol{x})$ is the probability that your classification is wrong.
- $\int_x P_h(error|\boldsymbol{X})P(\boldsymbol{X})$ is the expectation of the errors.
- $\delta(h(\boldsymbol{X}), Y)$ is

Proof in words: ???
Aside: note that for one classification y is not a vector. It is a point.

Misc NB

Naive Bayes continuous video:
`https://www.youtube.com/watch?v=r1in0YNetG8`

Logistic Regression

Another probabilistic approach to classification (categorical predictions).
Can use discrete or continuous outputs.

Summary from non-class sources:
We are still using linear regression in the inputs, but putting the result into a sigmoid function.
Recall $w_0 + w_1x_1 + w_2x_2 + w_3x_3 = w^Tx$ and $x = (1, x_1, x_2, x_3)$.
$P(death|x) = \sigma(w^Tx)$ where σ, the sigmoid function, converts your regression output into a sigmoid curve.
$\sigma(a) = \frac{1}{1+e^{-a}} = \frac{1}{1+e^{-(w_0+w_1x_1+w_2x_2+w_3x_3)}} = \frac{1}{1+e^{-(w^Tx)}}$
We can convert this to a linear relationship by "taking the logit".
The logit (log odds) is the inverse of the logistic.
$F(x) = \sigma(a)$ above. It is the probability that the dependent variable equals a case, given some linear combination of the predictors. It can range from $-\infty$ to ∞ The logit is ln $\frac{F(x)}{1-F(x)}$, or equivalently, after exponentiating both sides:
$\frac{F(x)}{1-F(x)} = e^{w^Tx}$
The logit (i.e., log-odds or natural logarithm of the odds) is equivalent to the linear regression expression.
Note used odds ratio: $\frac{p}{1-p}$

$\text{logit}(\frac{1}{1+e^{-(w^T x)}}) = \log(\frac{\frac{1}{1+e^{-(w^T x)}}}{1-\frac{1}{1+e^{-(w^T x)}}})$

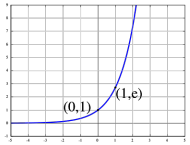
We can now proceed with linear regression.
Note that our predictions are now on the log scale; this impacts interpretation of the coefficients.

Lecture's presentation:
Notation:

- y^j: the j^{th} class
- x^j: the j^{th} training example

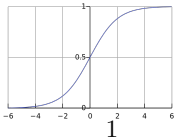
Once again we don't want to try to estimate $P(X, Y)$; that is challenging due to the size of the distribution.
We could make the Naive Bayes assumption and only need to calculate $P(X_i|Y)$, but if we want $P(Y|X)$, why not learn that directly? You can use logistic regression.

Exponential:



$$y = e^x = exp(x)$$

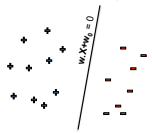
Reuse ideas from regression, but let the y-intercept define the probability.
$P(Y = 1|\boldsymbol{X}, \boldsymbol{w}) \propto exp(w_0 + \sum_i w_i X_i)$
With normalization constants:
$P(Y = 0|\boldsymbol{X}, \boldsymbol{w}) \frac{1}{1 + exp(w_0 + \sum_i w_i X_i)}$
$P(Y = 1|\boldsymbol{X}, \boldsymbol{w}) \frac{exp(w_0 + \sum_i w_i X_i)}{1 + exp(w_0 + \sum_i w_i X_i)}$



Logistic function:

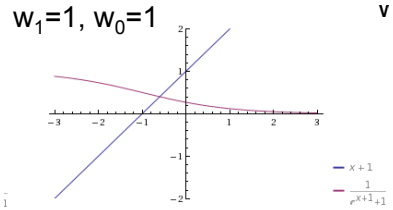
$$y = \frac{1}{1 + exp(-x)}$$

Making a decision boundary out of logistic equations:
Output the Y with the highest $P(Y|X)$.
If binary Y, output Y=1 if $1 < \frac{P(Y = 1|X)}{P(Y = 0|X)}$
That simplifies to just $1 < exp(w_0 + \sum_i w_i X_i)$ or $0 < w_0 + \sum_i w_i X_i$



The decision boundary is a line (or hyperplane), hence we have a linear classifier!

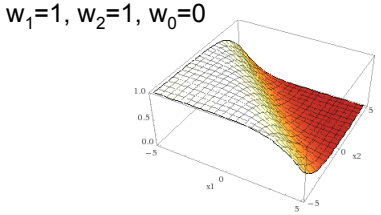
For $P(Y = 0|\boldsymbol{X}, \boldsymbol{w}) = \frac{1}{1 + exp(w_o + w_1 x_1)}$:



(See notes for more w_0, w_1 values plotted.)
In these plots, Y is the probability that the class is 1.
The red curve is the sigmoid. The blue line is the decision boundary.
The decision boundary is from the equation $0 = w_1 X + w_0$.

Larger weights result in a sharper curve. The bias w_0 shifts there the middle of the curve is.
The red sigmoid defines a probability distribution over Y in {0,1} for every possible input X.

The decision boundary leads to $P(Y = 0|X, w) = 0.5$ when you are at the $y = 0$ point on the line.
(E/J words: when the blue line crosses the x axis, that's when the sigmoid curve is above 1/2, which corresponds to classifying it as a no/0.)
The slope of the line defines how quickly the probabilities go to 0 or 1 around the decision boundary.
2D inputs:
For $P(Y = 0|\boldsymbol{X}, \boldsymbol{w}) = \frac{1}{1 + exp(w_o + w_1 x_1 + w_2 x_2)}$:



$P(Y = 0|X, w)$ decreases as $w_0 + \sum_i w_i x_i$ increases. Again, if you set the stuff inside the exponential to zero, you get the decision boundary hyperplane.

Finding the w coefficients

Generative (Naive Bayes) loss function: Now j is a data point with observations indexed over i.

$$\ln P(D|\boldsymbol{w}) = \sum_{j=1}^{N} \ln P(x^j, y^j|\boldsymbol{w}) \quad \text{(the full log-likelihood)}$$

use Bayes' rule to rewrite conditionally

$$= \sum_{j=1}^{N} \ln[P(y^j|x^j, \boldsymbol{w})P(x^j|\boldsymbol{w})]$$

$$= \sum_{j=1}^{N} \ln P(y^j|x^j, \boldsymbol{w}) + \sum_{j=1}^{N} \ln P(x^j|\boldsymbol{w})$$

We decide to ignore the 2nd term because it won't help you get better predictions for that data anyway. Or, "From a machine learning perspective, "God gave us the data" and we don't care about the 2nd sum."
Professor Farhadi is calling this first time a discriminative (logistic regression) loss function:
It is helping you discriminate between different classes. It's not going to help you model the data. This is unlike regression; we don't care about the value it puts out. We only care about what the resulting class is.
This is the difference between statistics and machine learning. We only care about getting the best $\boldsymbol{w}$ for discriminating between classes.
Conditional Data Likelihood: "Conditional" because you are conditioning on what $\boldsymbol{X}$ is.

$$\ln P(D_Y|D_{\boldsymbol{X}}, \boldsymbol{w}) = \sum_{j=1}^{N} \ln P(y^j|\boldsymbol{x}^j, \boldsymbol{w})$$

D_Y = ???
$D_{\boldsymbol{X}}$ = ???
Doesn't waste effort learning $P(X)$. Focuses on $P(Y|\mathbf{X})$, which is all that matters for classification.
Discriminative models cann't compute $P(\boldsymbol{x}^j|\boldsymbol{w})$! ???

Conditional Log Likelihood

(the binary case only).
$P(Y = 0|\boldsymbol{X}, \boldsymbol{w}) = \frac{1}{1+\exp(w_0+\sum_i w_i X_i)}$
$P(Y = 1|\boldsymbol{X}, \boldsymbol{w}) = \frac{\exp(w_0+\sum_i w_i X_i)}{1+\exp(w_0+\sum_i w_i X_i)}$
($l(\boldsymbol{w})$ is conditional data log-likelihood.)

$$l(\boldsymbol{w}) \equiv \sum_j \ln P(y^j|x^j, \boldsymbol{w})$$

Since y^j is in $\{0, 1\}$, sum over the two cases:

(the y^j and $(1 - y^j)$ act like delta functions)

$$l(\boldsymbol{w}) = \sum_j y^j \ln P(y^j = 1|x^j, \boldsymbol{w}) + (1 - y^j) \ln P(y^j = 0|x^j, \boldsymbol{w})$$

plug in the definition of the likelihoods and do algebra to get:

$$= \sum_j y^j (w_0 + \sum_i^n w_i x_i^j) - \ln(1 + \exp(w_0 + \sum_i^n w_i x_i^j))$$

While we can't find a closed-form solution to optimize $l(\boldsymbol{w})$, $l(\boldsymbol{w})$ is concave so we can to gradient ascent.

Gradient ascent to optimize w

Conditional likelihood for Logistic Regression is convex. (see above)
Gradient:

$$\nabla_w l(\boldsymbol{w}) = [\frac{\partial l(\boldsymbol{w})}{\partial w_0}, \ldots, \frac{\partial l(\boldsymbol{w})}{\partial w_n}]'$$

(The $'$ at the end is for transpose b/c usually a column vector.)
Update Rule:

$$\Delta \boldsymbol{w} = \eta \nabla_{\boldsymbol{w}} l(\boldsymbol{w})$$

η is the learning rate. $\eta > 0$.
Your next weights $(t + 1)$ become: $w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \frac{\partial l(\boldsymbol{w})}{\partial w_i}$

Gradient ascent is the simplest of optimization approaches. Note that conjugate gradient ascent is much better (see reading).(?)
Vocab

- **discriminative**: estimates joint probabilities. E.g. $p(Data, Zebra)$, $p(Data, NoZebra)$.
- **generative**: E.g. $p(Zebra|Data)$, $p(NoZebra|Data)$.

Andrew Ng:
* gives numbers between 0 and 1 (good for classification)
* called "logistic regression" but it is really for classification. (don't be confused by "regression")
* "sigmoid function" and "logistic function" are essentially synonomous.

General Vocab

- **classification** - ??? Finding a f that converts X to Y where Y are categorical. (Not regression).
- **held-out data**: synonymous with validation data (?)
- **hypothesis space**: ? E.g. binomial distribution for coin flip.
- **prediction error**: measure of fit (?)
- **regularization**: a process of introducing additional information in order to solve an ill-posed problem or to prevent overfitting
- **kernel**: some transformation of your features that improves your classification