Machine Learning CS726 Fall '11

Angjoo Kanazawa

October 13, 2011

1 September 1st: Class one

1.1 Logistics

- Uname: cmsc726 Pass: generalize
- Programming projects **27%**: 3 total, after 48hrs 50% down, teams. (Python with numpy).
- Written hw **18%**: 13 of them, one per week out of 3 scales {0, 0.5, 1}. Individual, can't belate.
- Midterm **25%**:
- Final "practical" exam **25%**: Canned or your choice/teams, presentations during the final slot
- Class/Piazza participation **5%**

Time expectation

- 3 hrs in class
- 2 hrs reading
- 2 hrs on written asg
- 2 hrs on programming projects
- Don't ask questions that have already been answered

Do now:

- **Do hw00**: Due 6 Sept, next tuestday (Submit in .pdf only using *handin*)
- **Do the first reading**: pass protected, read the web page.
- **Sign up**: Subscribe to the Piazza group

1.2 Contents

Classification

- When there's an ambiguity in the data, which function are you going to learn? =¿ **Inductive Bias** (i.e. bird vs non-bird, fly vs land, background focus vs bg blurred)
- Which is the more reasonable distinction to make?
- The primary way different learning algorithms vary. There's no right answer. Does your inductive bias match your problem?
- 3 primary components: labels, features, model

Ingredients for classification

1. Feature representation
   - not typically a focus of machine learning
   - Considered as "problem specific", but very problematic if bad.
2. Training data: labeled examples
   - Often expensive
   - Sometimes available for free
3. Model
   - No single learning algorithm is always good ("no free lunch" theorem)
   - Requires some control over **generalization**

Regression is like classification, except that the *labels are real values.*
Structured Prediction

2 September 6th: Class 2

Answer hw questions in few sentences.

2.1 Intro cont

- 95% of things in ML are binary classification. But many ML problems reduce to binary classification. We don't have to develop solutions for different problems
- **Reinforcement Learning** - unlike classification, regression, and unsupervised learning, RL doesnot recieve examples, but it **experiences** by interacting with the world.
- RL always has *time* as a variable. i.e. chess, robot control. Comes with the "exploration vs exploitation" trade-off. Humans are really bad at exploitation.
- Defined by *state space*, the world, *actions*, and the *reward.*
- Is a natural extention to search etc we won't cover this too much.

Why we need some math:

- **Calc/Linear Algebra**: finding maxima/minima of functions, allows high-dimensional data analysis.
- **Probability**: the study of the outcome of repeated experiments (frequentist), the study of the plausibility of some event (bayesian)
- **Statistics**: the analysis and interpretation of data. Uses probability theory a lot. Stat is very similar to ML. But stat is about analysis/interpretation of data, but ML is about generalizing and using the analysis to make predictions.
  - A lady drinking tea..
  - model vs predictions
  - model fit vs generalization
  - explain the world vs predict the future

2.2 Decision Trees

- Asks questions at nodes, edges are possible options.
- Super efficient, the amount of time is the depth of the tree.
- Histogram representation: in the absence of extra information (no info, no question), make a frequency histogram of the end result. i.e. 62% play, 38% no play.
- Hope that you get about 62% right. Assumption: relative frequency of play/no play is indicative of the real distribution of play/no play.
- Try to ask one question at a time that's most benefitial.

- *example: is sunny?* =¿{no (play-80%, noplay-20%), yes(play-33%, noplay-66%)}. If no, I'll guess play and expect to get 80%, if yes, I'll guess no and expect about 2/3rds right.
- On 5/8th of the days it's not sunny, 3/8th of the days is sunny.
- $5/8 \times 80\% + 3/8 \times 66\% \approx 75\%$
- *example: is windy?* =¿ { yes(p-50%, np-50%), no(p-75%, np-25%). Frequecy of windy days is 1/2.
- So the training accuracy under this feature(question): 1/2(75%)+1/2(50%) = 5/8 = 62%
- So, do I want to ask "is windy" or "is sunny"? Is sunny!
- For each of the featuers, we're going to compute the training accuracy, get the feature that maximizes the accuracy. Then make that the first question on the decision tree.
- After the root of the tree is found, remove that feature, partition the data associated with the correct edge (not sunny days or sunny days) and repeat the step == The algorithm for decision tree:

```
1. guess - get the most frequent answer
2. base case:  labels are umbiguous, no more features.
3. else for all features, compute the score (training accuracy).
   Find the guy with max score, split the data, remove that guy,
   recurse on left and right
```

- A recursive function, so the depth of the recursion is the number of features, if the features are binary (other wise it's the number of possible feature values etc). So this will terminate.
- Eventually you'll ask every question, so we'll hit the second base case only if there is an inconsistency (two examples with exact same features but with different labels)
- **goal**: good future predictions given (label, example) pairs. A probability distribution, D, is the guy generating the examples of what the problem looks like. Is the God. Good future predictions means minimum test error.
- $P_D(label, example)$. If I have learned well, I should be correctly assign labels to examples that D thinks is likely.
- test error=$\epsilon = E_{(x,y)\ D}[f(x) \neq y]$, where $f(x)$ is the prediction, y is the true label.
- x may be reasonable, or not, because it may not fit the type of data for the problem.
- We don't have D, but we have the training data (x, y), examples drawn from D. Without D, minimizing ϵ is too hard.

- **goal'**: good training predictions, i.e. minimize training error.
- $\hat{\epsilon} = 1/N \sum_{(x,y)\in train}[f(x) \neq y]$
- Can the decision tree algorithm minimize $\hat{\epsilon}$? if there is no ambiguity in the training data, we can ask all the question (worst case), then we pinned down all examples (and there are no diff labels), and we can do this. If there is an ambiguity/inconsistency, it's not possible to always get the 0 training error.
- The best we can hope for is that whatever this minimum error is, it will achieve that. For this goal', the algorithm is optimum.
- Does solving goal' help us help solve goal? if the assumption that test comes from the same D, we can.
- Doesn't work when test data is small, and is not representative of the D.
- Say there are spurious correlations between features and the labels in the training data. (looks useful in training, but not in real test). The decision tree will pick these features, it will make $\hat{\epsilon}$ low, but ϵ will be high. This is **overfitting**.
- Solutions: constrain the height of the tree (you can only ask x questions and hope that it won't ask questions about this spurious feature), get more data.

3 September 8th Class 3

Project 1 is online now. Implement basic classifiers. Start early! Due 27th September.

3.1 HW1

1. Memorizing doesn't generalize, and overfits.

2. Look at 3.2

3. Goal: minimize $\epsilon = \mathbf{E}_{(x,y)\ D}[f(x) \neq y]$. We can calculate $\hat{\epsilon} = \frac{1}{N}\sum_n [f(x_n) \neq y_n]$. If overfit, we do too well on the training data so $\hat{\epsilon}$ will be tiny but not the ϵ.

4. The algorithm should never look at the data because it'll overfit and we won't be able to say anything about the model's performance. We shouldn't look at it because we'll get a sense of what features will be important and fix the model accordingly.

3.2 Decision Tree

Claim: shallow trees are less prone to overfitting.
Example: two coins, A, B. Result of coin A will be the feature,x, that of coin B will be the label,y (they aren't correlated). Got $(1,1),(0,0),(0,0),(0,1)$. Looks like the feature is very useful, but not. The hope is that these accidental correlation will go down the tree, and the actual features that work go up the tree.

The depth is a *hyper parameter*. The goal of using hyperparameters is to reduce overfitting. If we treat it as a parameter, it'll just use max depth and lower $\hat{\epsilon}$. *Parameters* are what you estimate on the training data.

3.3 Geometry

We represent our inputs as vectors in high dimensional space. $y = \pm 1$, $x = (x_1, x_2, \cdots, x_n) \in \mathbf{R}^D$
K-NN: Look at K nearest neighbors, label the new point as the majority/mean of the neighbors. If inf-NN, that's just taking the majority. This is underfitting. 1-NN would be overfitting. Choose odd K so that there won't be any ties.
K-mediods Clustering.

1. Pick K datapoints to be representatives

2. $\forall n$, put n in the closest cluster (closest K datapoints)

3. Choose a new prepresentative that minimizes the average dist to all cluster members

4. Go back to 2

K-means Clustering.

1. Pick K datapoints randomly $\mu_1, \ldots, \mu_k$

2. $\forall n$, $z_n = \arg\min_k |x_n - \mu_n|$.

3. $\forall k$, $\mu_k = \frac{1}{n_k}\sum_{n;z_n=k}[x_n]$ (Make a new mean prepresentative who is an average of all cluster members.)

4. Go back to 2

Does K-means always converge? Measure the quality of the solution by the average distance from each point to its mean (score). (Convergence as in this measure of cluster quality won't change.)
Observation 1: Everytime step 2 or 3 is executed, the score goes down or stays the same.
Oberservation 2: If there are n members and k clusters, the possible clustering assignment is finite, k^n.
Does K-means always find the optimal clustering?
No! Not necessarily. The initial point assignment is random and it changes the result. The practical method is to do multiple initialization and pick the smallest score.
How many iteration does it take to converge?
In theory it takes a time that's exponential to k. In practice, it converges in like 4 iterations. Smooth analysis of why this converges quickly. Only certain initial points take a long time. If it's taking a loong time, if you perturbe the initial points and move it away (like how simplex is really fast in real life).

$K = n$ ($\hat{\epsilon} = 0$) or $K = 1$ gives us no information about the data. As an implementation note, some cluster centers will disappear and never come back. So be able to handle that situation.

3.4 Curse of High Dimentionality

as we make the dimention higher, the distance between points get smaller. If every point is about the same distance from everything else, the k-nn are just random points. Picking the majority of those random points, is just another random guessing. As D increases, the distance between points concentrates to a point.

```
D = 100; hist(flatten(XtoD(rand(1000, D )) / sqrt(D)));
%XtoD takes the distance between points
```

Seems like K-nn shouldn't work, but because the D is uniform. It does work in real life, because in real life the actual data is correlated.

4 September 15th Class 4

4.1 HW2

2. Decision boundary for a one nearest neighbor classifiers on two datapoints is a linear plane perpendicular to both points.

3. Clustering as in given labled data (x_n, y_n), $\forall n \in N$ for each label run K-means on subset of data with this label, then run KNN using μ_is. A lot faster on KNN test time but more time on training, in a sense clustering throws away bunch of data. This could avoid overfitting. (Assuming that we're working on relevant feature set).

4. in 2D space, the dot product, $< \bar{w}, \bar{x} >$, projects all the points, X, onto $\bar{w}$ and makes it 1D.
   If $y = sign(w \cdot x)$, is the decision rule, then the boundary B is $\{x : w \cdot x = 0\}$. The disadvantage is that it assumes the decision boundary lies on the origin. This won't work if all the points are positive. So now, modify y to $y = sign(w \cdot x + b)$, s.t. $B = \{x : w \cdot x = -b\}$. Bias is always $-b$, as $b \to -\infty$, nothing can be classified as positive.
   Claim: using bias is the same as not using the bias but adding a new feature. i.e. x or $< 1, x >= \tilde{x}$, where x uses w (to get $w \cot x + b$) and $\tilde{x}$ uses $\tilde{w} =< bw >$ so it's the same thing i.e. $w \cot x + b = \tilde{w} \cdot \tilde{x} =< b, w > \cdot < 1, x >$. So we can always write as if there's not bias.

4.2 Perceptron

General algorithm

```
for each (x,y)
   //if error, do an update
   if y(w\cdot x + b) \le 0: or if sign(y) == sign(w\cdot x + b):
      w = w + yx
      b = b + y
```

Usually people scale the update with some constant $\gamma < 1$, the learning rate, so that it won't be so influenced by x. (each single error);

Question: If you add γ does it change?? Doesn't really change anything, because after one update, the scale of w changes, but the direction is the same. With a bias, it'll shift it a lot of a little. After any number of updates, without using the learning rate, you'll get (w, b) in the end but with γ, all you'll get is $(\gamma w, \gamma b)$., the decision boundary will have the same direction.
In the case of perceptrons, adding a γ makes no difference (assuming we have infinite floating point). In the case that w is a unit vector, b shifts the decision boundary b units, if w is not unit, it shifts $b/||w||$ units. So if $||w|| >>$, b won't really make that much difference.
More important point: the more you run, the less sentisitive the algorithm will be for any update. Early on, the decision boundary moves all over the place, but as the number of epoches increases, the relative magnitude of the update is smaller than that of the beginning updates. I.e. even if it won't converge, it'll slow down in the end.

5 September 22nd Class 6

5.1 Linear Seperability

Perceptrons can't do XOR, because 2D XOR is not linearly seperable. But if you add a 3rd dimension, where $-$'s have 0 and $+$'s have 1. Then you can have a hyperplane that seperates them.

You can add all possible combinations linearly, this is what perceptrons do. But that's computationally expensive and makes the feature very high dimension. This is prone to overfitting, and you need more examples.

DT can combine few features but in a complicated way.
DT might fail on XOR because of all ties, features very slightly correlated with the label..

5.2 Evaluation/ROC

Given a ratio of dataset where 1% of them are positive and 99% is negative, the classifier will say all of them are negative and achieve 99% accuracy. Example for cancer detection, false negative is worse than false positive. So there are different costs associated with different classification. Perhaps my goal is to order the data to see what's interesting to look at.

In biology, people look at *Sensitivity vs Specificity*, we call it *Precision vs Recall*. Precision says of all the Xs that you found, how many of them were actually Xs? Recall asks: of all the Xs that were out there, how many of them did you find? (should've found)

Choose the best threshold and get the harmonic mean, or their **f-measure**:

$$F = \frac{2PR}{P+R}$$

5.3 Cross-validation

Get bunch of data, divide it in n equal size. Say $n = 10$, train on $n_2 \dots n_{10}$, and test on n_1, but now, you can train on $n_1 \dots n_9$, then test on n_{10}. You can do this n times. Then you'll get performance over all of the data I have, not just a single slice.

Also you do this to tune your hyperparameters. For each hyperparameter setting, you get the score of each, you want the hyperparameter that does the best on average over all slices.

Say you run this on DT and get that depth = 17 is the best, so you have n trained classifier, get the one that does the best or, retrain a new classifier on all $n - 1$ training slices. Usually learning a new classifier is a better idea, but it might be slightly underfit, because you're regularizing (with depth=17), but using more data.

4 choices: Number of slices $n = 10$ if you're normal, you do $n = 5$ if you're running out of time, you do $n = 2$, you'll make sure your two classifiers aren't trained on the same data points, one more is train on everything except 1, *leave-one-out*, great for *knn* but too expensive for others. Leave-one-out is the least succeptable to underfitting.

5.4 Statistical Significance

Try to answer the question, “given this new method, does it actually work?”. You should use the most commen thing in your field. Two examples we cover is *Paired T-test* and *Jacknifing/Bootstrapping.*

Given DT and Perceptron’s performance, say perceptron always does better. “If I were to do a new experiment, with what probability would Perceptron beats DT”

This is what T-test does, fit gaussian to DT, fit gaussian to Perceptron, and if the overlap is tiny, then there’s small probability that DT will beat perceptron, this gives us a p-value.

Weakness is that there is a normal distribution assumption. LLN argument, it should be kind of, but... not really because accuracy lives between [0, 1], normal lives between $[-\infty, \infty]$.

So some people say that you should do binomial tests, but ppl still do p-value.

Jackknifing/Bootstrapping (The word “Bootstrapping” is used a lot everywhere). This is suited where your evaluation method is holistic over the tneir data. Like F-measure.. Way around when T-test can’t really be applied. Doesn’t make normality assumption, it’s a *non-parametric* test.

If you have a big dataset, it’s easier to seperate the two gaussian curves (because they’ll have more peaks), so you can get more examples to get more statistical significance.

6 September 27th Class 7

6.1 HW3

1. About non-linearly seperable data and perceptron: Do the add one feature augmentation trick, why is this linearly seperable now? For example, $\boldsymbol{w}$ D zeros and all indicators as corresponding y_n the label.
   Does this affect generalization? The above construction will not work for test data at all. Generally it overfits, because it's like cheating after the first couple of passes when you start using the labels y_n as the components of your new augmented $\boldsymbol{w}$
   How long does it take to converge on this augmented data? Previously, $||x|| \leq R$, but now, we have $|| < x, 0, 0, \dots, 1, \cdots > ||$?. Square it:
   $$\begin{aligned} || < x, 0, 0, \dots, 1, \cdots > ||^2 &= || < x, \dots, 1, \cdots > \dot{<} x, \dots, 1, \cdots > || \\ &= x\dot{x} + 1 \\ &= ||x||^2 + 1 \end{aligned}$$
   So $\hat{R} = \sqrt{R^2 + 1}$, but now γ changes also. Using the $\boldsymbol{w}$ above, the margin for this $\boldsymbol{w}$ is $\min_{(x,y)\in D} \frac{y}{||w||} w\dot{x}$ $||w|| = \sqrt{N}$, and max $w\dot{x}$ is 1 so the new margin is at least $\frac{1}{\sqrt{N}}$

2. Centering, Variance scaling vs DT, KNN, perceptron:

   - **KNN** is not sensitive to centering but it is sensitive to variance scaling.
   - **DT** is not sensitive to centering or scaling because the cut off is still the cut off.
   - **Perceptron** is sensitive to centring and variance scaling, because if the feature value is huge, it'll move $\boldsymbol{w}$ a lot to one direction, it'll take more time because R is huge.

6.2 Blue Box

- In the realm of *scale* of features, variance pruning might not work if done prior to scaling because it might be a good feature in the correct scale.
- *How do you get a confidence out of a DT or KNN?* For DT you can calclulate the probability. Each node is a conditional probability. For KNN, we can calculate the average within k neighbors and get a percentage (with $K = 6$, 3 might be VE, 3 might be $-VE$, so 50% chance

6.3 Multiclass Classification

Reduction-based ML: Is multiclass harder than binary? If I could do binary, then I'll use that to solve multiclass.

Error-bounds: If I can achieve ϵ error in binary classification, we can have $\leq f(\epsilon)$ error on multiclass classification.

The big assumption is that we can achieve ϵ error.

Definition: Binary Classification.

- Assume some distribution $\mathbf{D}$ of (x, y) pairs
- get sample D $(x_1, y_1), \ldots, (x_n, y_n)$ drawn from $\mathbf{D}$
- The goal is to compute $f : X \to \{-1, +1\}$ s.t. $\mathbf{E}_{(x,y)\ D}[f(x) \neq y]$ is small.

Weighted binary classification: Weight the negative sample less and positive samples more, so that the classifier has to put more effort to classify positive examples correctly. Now the goal is different it is to minimize (arbitrarly assuming that possitive error costs more here):

$$\mathbf{E}_{(x,y)\ D}\begin{cases} \alpha[f(x) \neq 1] & \text{if y=1} \\ [f(x) \neq -1] & \text{if y=-1} \end{cases}$$

For some $\alpha > 0 \in \mathbf{R}$. Above can be re-written as $\mathbf{E}_{(x,y)\ D}[\alpha^{y=1}[f(x) \neq y]$ Algorithm to use: **Subsampling**. We end up with a "balanced" data. If the cost is $\alpha = 10$, I want 10th as many positive examples. This is optimal for us. What we want

1. given $\mathbf{D}^{WBC}$, we want to reduce it to $\mathbf{D}^{BC}$
2. learn a function f on the binary classification problem.
3. given a test point $\hat{x}$, use f to solve the weighted binary classification problem.

The algorithm

1a Retain all positive examples

1b For each negative examples, sample $1/\alpha$ of them.

2 $\hat{y} = f(\hat{x})$ (i.e. 100 exmples, 98 -ve, 2ve, $\alpha = 2$, randomly select 49 negative examples.

We want $\mathbf{E}_{(x,y)\ D^{WBC}}[\alpha^{y=1}[f(x) \neq y]]$, from weighted D.

$$\begin{aligned}
\mathbf{E}_{(x,y)\ D^{WBC}}[\alpha^{y=1}[f(x) \neq y]] &= \sum_{(x,y)} \mathbf{D^{WBC}}(x, y)[\alpha^{y=1}[f(x) \neq y]] \\
&= \sum_x \begin{cases} \mathbf{D^{WBC}}(x, +1)[\alpha[f(x) \neq 1]] \\ \mathbf{D^{WBC}}(x, -1)[f(x) \neq -1] \end{cases} \\
&= \sum_x \begin{cases} \mathbf{D^{BC}}(x, +1)[\alpha[f(x) \neq 1]] \\ \alpha\mathbf{D^{BC}}(x, -1)[f(x) \neq -1] \end{cases} \\
&= \alpha \sum_x \begin{cases} \mathbf{D^{BC}}(x, +1)[[f(x) \neq 1]] \\ \mathbf{D^{BC}}(x, -1)[f(x) \neq -1] \end{cases} \\
&= \alpha\epsilon
\end{aligned}$$

Because we downselected negative examples, so to go from D^{WBC} to D^{BC}, we need to multiply it with α.

Can we do better than $\alpha\epsilon$ (on the weighted problem)? No, the error grows linearly with the size of α. You could imagine to do slightly better because of the sampling more of +ve but something around there.

Sampling seems wasteful. SO instead, we can keep all negative examples and put α times more positive examples (replacement of). This is **over-sampling**. Both gives us the same bounds, but ϵ can be different. ϵ is the error rate of the BC classifier. The big assumption is that we can achieve ϵ, it's easier to achive low ϵ in oversampling than undersampling. *Even though the bound for both sampling is $\alpha\epsilon$, ϵ is smaller from oversampling, generally.*

6.4 Turning MC to BC

1. **OVA** (one-vs-all): with k classifiers $f_1 \dots f_k$, where f_1 seperates $y = 1$ from $y \neq 1$ and f_k seperates $y = k$ from $y \neq k$. Two bad things can happen:no one says yes, more than one says yes. Just pick at random for each, but these are the cases when the error occurs. Not great because of this. In practice, you pick the one with the highest confidence. Called brittle, because it takes only one error will shoot up the probability of error on MC. Any errors can cause grave harm, if each error is ϵ, and you have k chances, it's pretty bad.

2. **AVA** (all-vs-all): $\binom{k}{2}$ classifices where f_{ij} seperates class i from j.

7 September 29th Class 8

7.1 HW 4

1. Task for squared error regression: Given: input space $\mathcal{X}$ and an unknown distribution $\mathcal{D}$ over $\mathcal{X} \times \mathbf{R}$, Compute $f : X \to \mathbf{R}$, minimize $\mathbf{E}_{(x,y)\ \mathcal{D}}[f(x) \neq y]$

2. *To what types of error do these theorems apply?* All error in respect to data taken from the same distribution $\mathcal{D}$.

3. OVA vs AVA. If training for each classifier is $O(N)$, OVA: $O(NK)$, AVA:$O(K^2N/K) = O(K)$, if training for each classifier is $O(N^2)$, OVA: $O(N^2K)$, AVA: $O(K^2(N/K)^2) = O(N^2)$, so AVA is better! so the worst it is to train, the better and better AVA gets.

4. A ranking preference function ω that penalizes mispredictions linearly up to K:
$$\omega(x, y) = \begin{cases} min\{k, |i - j|\} & \text{if } i \neq j \\ 0 & \text{else} \end{cases}$$

7.2 Continue on Multiclass classification

There's OVA, AVA(the default), and **Tree-based** classification. Idea is to divide and conquer, split all the classes in two, then keep on splitting. At the leaf we make decisions. So if we have K classes, we have $K - 1$ nodes and so we train $K - 1$ classifiers. This divides the data, so we beat OVA's $O(K\epsilon^8)$, K factor.

As soon as we make an error, we're done. The chance of making an error is ϵ^b, the number of times you can make an error is how many nodes you go down (pf union bound: $P(A_1 \cup A_2 \cup \dots A_n) \leq \sum P(A_i)$, so at most the error caused by tree based is
$$\epsilon^{MC} \leq O(\epsilon^b log(K))$$
If the number of features is not a power of 2, it might not be a full binary tree with height $log(K)$, so more of the emphasis on $\leq$.

But here we can't specify which order/the layout of the tree. Certain layouts could make making a binary classifier easier. I.e. if data were clustered in $1 : (0, 0), 2 : (0, 1), 3 : (1, 0), 4 : (1, 1)$, shouldn't split classes $\{1, 4\} vs \{2, 3\}$.

7.3 Ranking

Bipartite Ranking: Some are labeled good and some are bad. The job is to rank the good, relevant ones before the bad ones. Given nodes a, b, the labels is 1 is a is good and comes before b and should, -1 if b comes before a.
We use $\omega(x, y)$ as the preference function just like binary classification's loss function. Where x is the true position, y is my ranking.

Objective *Given* an input space $\mathcal{X}$, and an unknown distribution $\mathbf{D}$ over $\mathcal{X}$, *compute:* a function $f : \mathcal{X} \to \Sigma_M$ minimizing

$$\mathbf{E}[\sum_{\mathbf{u} \neq \mathbf{v}} [\mathbf{sig_u} < \mathbf{sig_v}][\hat{\sigma}_\mathbf{u} < \hat{\sigma}_\mathbf{v}]\omega(\sigma_\mathbf{u}, \sigma_\mathbf{v})$$

Where $\hat{s}ig = f(x)$. Basically did we swap u adn v, if so, pay $\omega(u, v)$.

$$f(x_{uv}) \begin{cases} +1 & \text{if } u < v \\ -1 & \text{if } v < u \end{cases}$$

$f(x_{uv})$ answers "is $u <_f v$?" Now use this as a comparison function and use any sorting algorithm, like quicksort. So it's just a quicksort with a minor tweak:

Consider $f(x_{uv})$ as $P(u < v)$, so in quicksort, pick a pivot, is this other number greater than the pivot or not? We answer this by running f, and f will tell us $x\%$ it's less. So it's like a randomized quicksort.

So the weighted problem looks like:, given $(y = \{+1, -1\}, x, w)$, (where $w = \omega$, $x = x_{uv}$) we want to minimize

$$\epsilon^w = \mathbf{E}_{(x,y,w)\ \mathcal{D}}[w(y \neq f(x))]$$

How would you solve this? Over sampling, add 2 more (x, y) if $w = 2$. Undersampling, we want to keep if w is high, so throw out data with probability $1/w$. So if w large, we most likely never throw out the data. And we have $\epsilon^w \leq \epsilon^b[\mathbf{E}_w[w]]$ is the best we can hope for.

7.4 Collective Classification

Given a graph, where node is a feature, like a website, and if we were to label each node as +1 not offensive, or −1 offensive, the idea is that some information in the graph structure will tell us about the label. i.e. offensive websites tend to link to offensive websites.

So train a classifier with a bigger feature, wher you include information about the node itself plus the number of positive neighbors and negative neighbors. Problem with this is that.......TODO!

The **Stacking algorithm**: Forget that we have a graph structure. First try to classify each page based on its individual features alone. So think: $f_0 : \mathcal{X} \to \{+1, -1\}$, and now I have guesses for all. Now we can train a second round classifier, that takes an input from the pages themselves and the neighbor information from the previous classifier (the new added info on the feautures are given by f_0). So train $f_1 : \mathcal{X} + f_0$'s output $\to \{+1, -1\}$. Now, the hope is that f_1 wil get everything correct. Now train f_2 that takes in $\mathcal{X}$ and f_0 and f_1's output. etc.

Problem: it never get's the true label of the neighbors, but only the predicted data. This might be beneficial because at test time we never get to see the true label of the neighbors. But the issue is overfitting: f_0 is trained and tested on the original data, so chances are it'll do really good on the input data. So input to f_1 will be unrealistically close to perfect. In the real world f_0 will do much worse.

Concrete example: task for each 3D point, guess if it corresponds to a tree or not, don't get info but angle and distance. It's nearly impossible to classify the point just by using it's features, you really need the entire graph structure. So at train time we do well, but ... not in test..

8 Oct 4th Class 8

8.1 Linear Classifiers

Linear decision boundary: $sign(\boldsymbol{w} \cdot \boldsymbol{x} + b)$. If $b = 0$, the plane passes through the origin. Computing the distance of this plane from any point $\boldsymbol{z}$, is $\frac{\boldsymbol{w} \cdot \boldsymbol{z} + b}{||\boldsymbol{w}||}$. So from the origin is $\frac{b}{||\boldsymbol{w}||}$. If margin ¡ 0, loss is 1, if margin ¿ 0, loss is 0.

Indicator function $\infty(\cdot) : \begin{cases} 1 & \text{if argument is true} \\ 0 & \end{cases}$ Using 0-1 loss finding the decision boundary is NP-hard.

8.2 Regularization

We're concerned about test error, not training error. Just minimizing the training error will overfit. So Regularized Learning (Tikhonov($\lambda R(\boldsymbol{w})$), Ivanov(minimize the loss with constraints that $R(\boldsymbol{w}) \leq \delta$), Morozov (minimize the regularizer with constraints on the loss function $le\delta$)) All three do the same thing, but most common is Tikhonov:

$$w = argmin_{w,b} \frac{1}{N} \sum_n \infty(y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) < 0) + \lambda R(\boldsymbol{w})$$

This balances empirical loss and complexity of the classifier - prefers simpler one. $R(\boldsymbol{w})$ is the regularizer for linear hyperplanes. We want both the loss and regularizaer to be convex.

8.3 Convexity

Definition:A set S is convex if $\forall x, y \in S, \alpha x + (1 - \alpha)y \in S$ where $\alpha \in [0, 1]$. THe line segment joining x and y is contained in S. $S = \{x : x^2 > 2\}$ is non-convex because $x \geq \pm\sqrt{2}$, inside it's empty $S = \{x : \boldsymbol{x}^T\boldsymbol{x} < 1\}$. Show in two points norm will always be ¡ 1 with α $S = \{U : U^TU = 1\}$, orthogonal matrices, is *not* convex. So eigenvalue problems are non-convex.

Operations that preserve convexity:

- Intersetion
- Affine function $f(x) = Ax + b$)

Definition: f defined on a *convex* set is a *convex* if

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y) \forall \alpha \in [0, 1]$$

The line is always above the function. Or equivalently, f is convex if its *epigraph* is a convex set:

$$E = \{(x, \mu) : \mu \in \mathbf{R}, f(x) \leq \mu\}$$

(region(set of points) that lies above the graph of the function f.

Also, f convex $\Rightarrow -f$ concave.

Checking for convexity: First check if the domain is convex

1. Use definition (chord lies above hte function)

2. For differentiable f, function lies above all tangents i.e.
$$f(y) \geq f(x) + f'(x)(y-x)$$
3. For twice differentiable f, second derivative is non-negative. (These are positive definite matrices)

If the inequalities are strict, then f is *stritctly convex.* Operations that preserve convexity:

- Positive scaling/addition
- Affine function composition: f convex $\rightarrow f(Ax+b)$ convex
- Pointwise maximum: f, g, convex $\rightarrow h(x) = max(f(x), g(x))$ is convex
- f convex, g convex non-decreasing =¿ $h(x) = g(f(x))$ convex
- f concave, g convex non-increasing $\rightarrow h(x) = g(f(x))$ convex

exp good example of strictly convex, non-decreasing Examples:

- e^x, x, x^2, x^4 take the second derivati
- Any vector or matrix norm
- $h(x) = max(|x|, x^2), x \in \mathbf{R}$
- $exp(f(x))$, convex, by the 4th property
- $\frac{1}{log(x)}, x > 0$. $log(x)$ is concave, $1/x$ is convex non-increasing in $\mathbf{R}^+$, so by 5 this is convex.

8.3.1 Convex Loss Functions

Hinge loss, exponential loss, and logistic loss

- Hinge loss: $L(y, \hat{y}) = max\{0, 1 - y\hat{y}\}$
- Exponential loss: $L(y, \hat{y}) = exp(-y\hat{y})$
- Logistic Loss:$L(y, \hat{y}) = log_2(1 + exp(-y\hat{y})$

Even if margin is good, won't get 0 for exponential and logistic.

8.4 Weight Regularization

Why?

- We need our weights to be small i.e. we DON'T want ϵ change in input($\hat{x} = (x + \epsilon)$) to make $\hat{y}$ chnage a lot i.e. $\boldsymbol{w}\epsilon$ will have a lot of influence if $||\boldsymbol{w}|| >>$.
- Regularization is generalization, regularized learning is statistically stable and bounds generalization error. Stability here means that differences in ϵ is bounded..
- We want regularizer to be convex for computational reasons

8.4.1 Norm based Regularizers

l_p norm:

$$||w||_p = (\sum_i |w_i|^p)^{1/p}, p \in [1, \infty]$$

if $p < 1$, it won't satisfy triangle inequality and convexity. We use l_1,l_2 usually. A unit ball in l_p is a set $S = \{w : ||w||_p \leq 1\}$, so in l_2 it will be a unit circle, with l_1, it will be a unit square rotated 90 degrees. As $p \to 0$, the sides approach the axis, l_0 ball looks like a plus sign. As $p \to \infty$ the circle approaches square. $(l_\infty(\boldsymbol{w}) = max\boldsymbol{w})$.

Properties:

- Solution often lies on the singularity (corners) of the ball when constrained enough. Why? (Why l_1 gives us sparsity, because the corners/edges are 4 or constant)
- Rotational Invariance: l_2 is invariant to rotations
- When to use l_1: less number of samples, lot of irrelevant features
- When to use l_2: otherwise, leads to good generalization.

Now we our final objective is:

$$L(\boldsymbol{w}, b) = \frac{1}{N}\sum_n l(\hat{y}, y_n) + \frac{\lambda}{2}||w||_2^2$$

8.4.2 Gradient Descent

Gradient of a function

$$\bigtriangledown f(x) = (\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \ldots, \frac{\partial f(x)}{\partial x_n}$$

Directional derivative Rate of change o fcuntion along a direction $\boldsymbol{v}$. $D_v(f) = \bigtriangledown f \cdot \boldsymbol{v}$ The function increases the most along the direction of the gradient. Why? because $\bigtriangledown f \cdot \boldsymbol{v} = cos(\theta)| \bigtriangledown ||\boldsymbol{v}|$, when the angle is 0, $cos(0) = 1$ and gives us the biggest increase.

Geometric interpretation:

- **Level sets**: for function $f(x)$, the level set containing a point a is a set of points $S = \{\boldsymbol{x} : f(\boldsymbol{x}) = f(\boldsymbol{a})\}$. Function admits same value at all points in the level set S
- Gradient at a point $\boldsymbol{y}$ is perpendicular to the level set containing $\boldsymbol{y}$. i.e. a gradient at point (x, y) will be perpendicular ot the level set going through (x, y). (from definition, gradient is always normal to the tangent going through the point)
- i.e. $f(x, y) = \sqrt{x^2 + y^2}$, level set is a cone, for $f(x, y, z) = \sqrt{x^2 + y^2 + z^2}$, the level set is a sphere.

Gradient Descent:

- Move in the negative direction of gradient to minimize a function.

- Steepest radient descent: To minimize a function f, start wiht some initial guess $x^(0)$ with update rule:

$$x^{(t)} = x^{(t-1)} - \eta^{(t)} \bigtriangledown f(x^{(t-1)})$$

9 October 11th Class 10

9.1 Probabilistic Modeling

Joint:$p(x, y)$, **Conditional**: $p(y|x)$ (some ppl think this is the discriminative).

9.1.1 Joint Modeling Setup

Assumption $\mathcal{D}$ (x, y), $f(\hat{x}) = argmax_y \mathcal{D}(\hat{x}, y)$ (pick the most likely thing). If there's unique y for every x with probability 1 there will be no error.

Training Data: $(x_n, y_n)_n$ $\mathcal{D}$, we want to estimate $\hat{p}(x, y) \approx \mathcal{D}$, where $f(\hat{x}) = argmax_y \hat{p}(\hat{x}, y)$

Typical Distributions:

- For labels with $\{1, -1\}$ (coin flips), when $p(h) = \beta$, $p(t) = 1 - \beta$, use *Bernoulli* distribution.
- For labels in the reals, people use *Gaussians.*

People approach joint modeling by factorizing $p(x, y)$ into $p(y)p(x|y)$ (chain rule). Backwards of what you think about classification. First decide if there's a bird or not, then draw the picture depending on if i decided if there's a bird or not.

xs are usually feature vectors, so it's sometimes to come up with a good model over $p(\boldsymbol{x}|y)$.

Example: label: play or not, features: wind, sunny, temp =¿ many combinations. For each, given Y, assign probabilities to each combinations. where sum of all combinations is 1. But when the number of features grow, it's just a lot of combinations and there might be not enough data.

Naive Bayes Assumption Make smaller tables, condition on play for each feature, so $p(W|P) = .6$, then $p(\not W|P) = .4$, and if $p(W|\ /P) = .2$, then $p(\not W|\ \not P) = .8$ Using

$$p(x_1, \ldots, x_n|y) = \prod_d p(x_d|x_1, \ldots, x_{d-1}, y) \approx \prod p(x_d|y)$$

i.e. *Features are independent given the label.*

Note: In the real world it's clear that $p(temperature|outlook) \neq p(temp)$. But naive bayes is not this bad, because $p(temp|outlook, play) = p(temp|play)$, we are conditioning on the label. Not the best assumption but not totally wrong.

So we have, $p(x, y) = p(y)p(x|y) \approx p(y) \prod_d p(x_d|y)$

Telling this fictional tale about where the data came from, the close the tale is the better my model will be.

Probabilistic assumption = inductive bias.

9.2 Primary BUilding model in Naive Bayes

Short hand for if $x = 1$, probability is θ, else $1 - \theta$:

$$\theta^{(x=1)}(1 - \theta)^{(x \neq 1)}$$

9.2.1 Parameter Estimation

: Goal is given data, estimate $\mathcal{D}$. Best way is **Maximum Likelihood**

- Pick parameters θ that maximize $P_\theta(D)$ where that's the likelihood of θ (not likelihood of data, probability of data given θ $(P(D|\theta))$).

Goal: Maximize $P((x_1, y_1), \ldots, (x_n, y_n)|\theta)$ Make a first big assumption, that xs are i.i.d. Then we get $max_\theta \prod_n p(x_n, y_n|\theta)$. The data isn't really iid because yesterday it was raining so it probably affects your samples.

Now using naive bayes, assume that we have

$$\approx max_\theta \prod_n p(y_n|\theta) \prod_d p(x_{n,d}|\theta, y_n)$$
$$\approx max_\theta \prod_n \theta_0^{(y_n=1)}(1-\theta_0)^{(y_n\neq 1)} \prod_d \theta_{+1,d}^{(y_n=1,x_{n,d}=1)}(1-\theta_{+1,d})^{(y_n=1,x_{n,d}\neq 1)}\theta_{-1,d}^{(y_n\neq 1,x_{n,d}=1)}(1-\theta_{-1,d})^{(y_n\neq 1,x_{n,d}\neq}$$

(θ_0 is how many days i play) Given Play=1, outlook=1 is $\theta_{+1,out}$, so play=1, outlook=0 is $(1-\theta)_{+1,-out}$, and so forth for each feature and it's label combination. So $\theta_{+1,d}^{(y_n=1,x_{n,d}=1)}$ is $\theta_{+1,out}$. (with D many features, we get 2D+1 coins and θ_0 is the guy who choses the label, then for each feature you'll get $2D$ possibilities)

examples: Given 50+ve, 150 -ve examples, $\theta_0 = 50/(50+150) = 1/4 = p(y)$. Take 50+ve ex and look at the outlook feature. Say 30 have outlook=1 and 20 have outlook=0. This is $\theta_{+1,outlook} = 30/50 = 3/5$. So training naive bayes is very easy, all you need to do is count. Very fast.

Pf: Just talking about labels so $\theta_0 = \theta$ Super easy case, get bunch of data $p(y_1, \ldots, y_{200}) = \prod_n p(y_n) = \prod_n \theta^{(y_n=1)}(1-\theta)^{(y_n\neq 1)}$ Now take the log so that it's easier to maximize. We can take the log because it's monotonically increasing. So:

$$= \prod_n \theta^{(y_n=1)}(1-\theta)^{(y_n\neq 1)}$$
$$= \sum_n ([y_n = 1]log(\theta) + [y_n \neq 1]log(1-\theta))$$
$$\frac{\partial}{\partial\theta} = \sum_n (\frac{y_n = 1}{\theta} + \frac{-[y_n \neq 1]}{(1-\theta)}) = 0$$
$$= \frac{1}{\theta}\sum [y_n = 1] \qquad = \frac{1}{1-\theta}\sum [y_n \neq 1]$$
$$(1-\theta)\sum[y_n = 1] = \theta \sum [y_n \neq 1]$$
$$\sum_n [y_n = 1] = \theta(\sum[y_n \neq 1] + \sum[y_n = 1])$$
$$\theta = \frac{\sum_n [y_n = 1}{N}$$

This is what we wanted!!!

9.2.2 a

Objective:

$$\max_\theta \prod_\theta \prod_k \theta_k^{[y_n=k]} \text{ subject to } \sum_k \theta_k = 1$$

Now rolling a die instead of the coin. Without the constraints, θ_k will go to infinity. Take the log of that and get

$$g(\theta) = \sum_n \sum_k [y_n = k] log(\theta_k) \text{ subject to } \sum_k \theta_k - 1 = 0$$

This is concave, with the restriction, it's still concave around the line that sums ot one (y=-1), it's concave in the whole space and over my restriction, so if we maximize we'll get the the most.

Now construct the lagurangian $\mathcal{L}(\theta, \lambda)$ For every constraint, add a λ, a lagurange multiplier. So here we only have one constraints and we get

$$\mathcal{L}(\theta, \lambda) = g(\theta) + \lambda(\sum_k \theta_k - 1)$$

Then,

$$\max_\theta \min_\lambda \mathcal{L}(\theta, \lambda) = \max_\theta g(\theta) \text{ subject to constraints}$$

(What ever i'm trying to do λ tries to do the opposite.) Now we can do $g(\theta)$ without the constraints by solving $\min_\lambda \max_\theta \mathcal{L}(\theta, \lambda)$.

Idea: All i care is making $g(\theta)$ big. But then there's λ who's trying to make $\mathcal{L}$ small. Say I find $\sum_n \theta_n = 2$, and makes $g(\theta)$ happy. But then λ will be set to $-\infty$, because $\sum_n \theta_n - 1 = 1$, so now λ has full control on making $\mathcal{L}$ small. If $\sum_n \theta_n = 0.5$, then $\sum_n \theta_n - 1 = -0.5$, so λ will go to ∞. The only thing I can do is to actually satisfy my constraints $\sum_n \theta_n - 1 = 0$. i.e. if we get $-\infty$, that means we didn't satisfy the constraints, if we do get something other than that we know we've satisfied it.

10 Class 11 October 13th

HW

1. $$\max_\theta \prod_d \theta_d^{x_d} s.t. \sum_d \theta_d^2 = 1, (\theta_d \leq 0)$$

   $\theta_d^2 \leq 0$ will always happen but technically need to include it. Since we're maximizing, we want to insure that the function is concave, and log is concave (second derivative should be non-positive). Make sure to check this! The overall goal is to $\max_\theta$ and λ plays the other role. Take the log:

   $$\begin{aligned} &= \max_\theta \sum_d x_d log(\theta_d) s.t. \sum_d \theta_d^2 = 1 \\ \mathcal{L}(\theta, \lambda) &= \sum_d x_d log(\theta_d) + \lambda(\theta_d^2 - 1) \\ &= \max_\theta \min_\lambda \mathcal{L}(\theta, \lambda) \end{aligned}$$

   The order of $\max_\theta \min_\lambda$: I pick θ then λ picks, if I pick θ that doesn't satisfy the constraint, λ will pick $-\infty$. If we had $\min_\lambda \max_\theta$, λ picks ≤ 0, then θ will be ∞, if $\lambda > 0$, I will still pick bad θ... You need to chose θ first, $\min_\lambda \mathcal{L}(\theta, \lambda)$ is a function of θ, to figure out what λ is, need to choose θ first. That's why the order matters..? (unresolved) Whatever, pick θ first! Now differentiate with respect to θ_d

   $$\begin{aligned} \partial\mathcal{L}/\partial\theta_d &= x_d/\theta_d + 2\lambda\theta_d \\ -x_d/\theta_d &= 2\lambda\theta_d \\ 2\lambda\theta_d^2 &= -x_d \\ \theta_D &= \sqrt{\frac{-x_d}{2\lambda}} \end{aligned}$$

   Now solve for λ. Substitute the constraint $\sum_d \theta_d^2 = 1$

   $$\begin{aligned} \sum_d \frac{x_d}{2\lambda} = \frac{-1}{2\lambda} \sum_d x_d = 1 \\ \lambda = -1/2 \sum_d x_d \end{aligned}$$

   This means, $\theta_d = \sqrt{(\frac{x_d}{\sum_d x_d})}$. Need to make sure that the positive solution is the right solution (unless there's a constraint that $\theta_d \leq 0$.

10.1 Maximum Likelihood

Just to remind ourselves, we were trying to maximize the joint probability $p(x, y)$ i.e. $f(x) = \arg\max_y p(x, y)$. The generative story is:

1. For each example $n = 1 \dots N$

   (a) choose $y_n \sim Ber(\beta)$

(b) for each $d = 1 \dots D$, choose $x_{n,d} \sim N(\mu_{y_n,d}, \sigma^2_{y_n,d})$ or $Ber(\theta_{y_n,d})$ any reasonable distribution. For each class and each feature there is a distribution.

(c)

This says if I know my model and the parameters, μs and σ, we can, so the idea is to reverse engineer these parameters. Example in matlab:

```
y = rand < 0.6
mu = zeros(2); mu(1,1) = 8; % where 1 is pos, 2 is neg, give random number
mu(1,2) = 11; mu(2, 1) = -4;  mu(2,2) = 10;
mu(y+1, :) % ans = 8 11
X = repmat(mu(y+1, :), 100, 1) + randn(100,2))
X2 = repmat(mu(2, :), 100, 1) + randn(100,2))
plot(X(:,1), X(:,2), 'b.') hold on;
plot(X2(:,1), X2(:,2), 'r*')
mean(X) % solution/estimate, gaussian with these two mean
mean(X2) % => pretty close to the actual mean
```

Turns out that if we want the decision boundary, it just becomes the linear classification. **LLR**: Re-write as $\log \frac{p(y=+1|x}{p(y=-1|x} = \boldsymbol{x} \cdot \boldsymbol{w} + b$. The decision boundary is just linear. This is assuming bernoulli model on the features. If we did the gaussian case, it works out too. But not with any distribution. Point is that this is a linear model. The parameters of the model $\theta_{\pm 1,d}$ you can compute by counting. (If gaussian model, find the μ, σ of data for each feature and each label). Constrasting with perceptron or other iterative classifiers, this we get the closed form.
We have a linear model and closed form solution, but there's no guarantee about seperability or large margins.

10.2 Generative vs Conditional

Generative is trying to model $p(x, y)$, but all we care about is $p(y|x)$ looks like. This is the conditional model. The big question in the generative was what probability distribution do we chose, then estimate the parameters. Same in conditional model, choose a distribution for $p(y|x)$, then estimate the parameters.

We like linear models, so maybe we want $\boldsymbol{w} \cdot \boldsymbol{x} + b$, wehre $\boldsymbol{w}$ is the parameters of the model. Since this is in the $\mathbf{R}$, we need to squish it down to interval $[0, 1]$

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

If $wx + b >>$, $\sigma = 1$, if $wx + b = 0$, $\sigma = 0.5$, if $xw + b <<$, $\sigma = 0$. So we have

$$p(y = +1|x) = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

$$p(y = -1|x) = 1 - \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b) = \sigma(-(\boldsymbol{w} \cdot \boldsymbol{x} + b))$$

Since $1 - \sigma(z) = \sigma(-z)$ (symetric property on sig) So

$$p(y|x) = \sigma(y(\boldsymbol{w} \cdot \boldsymbol{x} + b))$$

We're saying the distribution over the label is a function of the margin, which makes sense. Any function that maps $[-\infty, \infty]$ to $[0, 1]$ is good.

What we really care about is $\log(p(y|x)) = \log \sigma(y(wx+b))$ Since $\log(\sigma(z)) = -log(1 + e^{-z})$,

$$\log p(y|x) = -\log(1 + e^{-y(wx+b)}$$

The *logistic loss function*!!

So if you optimize a linear classifier under logistic loss function, it's the same as optimizing $p(y|x)$.

10.3 Discriminative Models:SVM

Generative and Conditional models are always probabilistic, for Discriminative there's no probability. We're explicitly optimizing the decision boundry for discriminative. Conditional models optimizes the probability of label given data. Generative models optimizes predicting both the label and the data.

If you care about probabilities, you should do generative, conditional models. When to prefer either or?

10.3.1 Current State of Generative vs Conditional

1. Assume the generative model is "correct". (Tell the story about how the data got generated and it's actually true, always false but have it for now). Then,
   (a) as $N \to \infty$, $\epsilon^{gen} \to \epsilon^{Bayes}$, $\epsilon^{cond} \to \epsilon^{Bayes}$ Asymptotic
   (b) fininte N, $\epsilon^{gen} \to \epsilon^{Bayes}$ faster than $\epsilon^{cond} \to \epsilon^{Bayes}$
2. Assume model not correct
   (a) as $N \to \infty$, $\epsilon^{gen} > \epsilon^{cond}$, Conditional is better because if the model is even slightly wrong, if we have infinite amount of data, it blows up.
   (b) N finite, $\epsilon^{gen} \to$ it's optimum faster than $\epsilon^{cond} \to$ it's optimum. We'll never get to ϵ^{Bayes}. It's no longer asymptote at the Bayes line, and they asymptote at different places, and conditonal's optimum might be better, but generative goes to its optimum faster.

Hall's heuristic is that if $N > 2D$, use conditional, $N < D$ use generative. Generative adds more information, where data comes from, to the model. So when N is small, it's very good. But when N is large, it relies too much on your inductive bias about the data.

Conditional and Discrimnative only makes sense for supervised.