# CPSC 440: Machine Learning

Generative Classifiers

Winter 2022

# Last Time: Product of Bernoullis

- We discussed multivariate binary density estimation:
  - Input: 'n' IID samples of binary vectors $x^1, x^2, x^3, \ldots, x^n$ from population.
  - Output: model giving probability for any assignment of values $x_1, x_2, \ldots, x_d$.

X =

| Inter 1 | Inter 2 | Inter 3 | Inter 4 | Inter 5 | Inter 6 | Inter 7 | Inter 8 | Inter 9 |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |

$p(x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 1, x_5 = 1, x_6 = 1, x_7 = 0, x_8 = 0, x_9 = 1) = 0.11$

(Estimates probability for all $2^9$ values)

- We discussed the product of Bernoullis model:
  - Assumes $x_j$ are mutually independent (strong assumption, easy computation).

$$p(x_1, x_2, \ldots, x_d) = p(x_1)p(x_2)\cdots p(x_d) = \theta_1^{x_1}(1-\theta_1)^{1-x_1}\theta_2^{x_2}(1-\theta_2)^{1-x_2}\cdots\theta_d^{x_d}(1-\theta_d)^{1-x_d}$$

- We discussed generative classifiers:
  - Supervised learning methods that model $p(x_1, x_2, \ldots, x_d, y)$.
    - Compute $p(y \mid x_1, x_2, \ldots, x_d)$ to make predictions.

# Naïve Bayes Generative Classifier

- Naïve Bayes: generative classifier, used for spam detection in 90s.
- Naïve Bayes Assumes features $x_j$ are mutually independent given y:
  - $p(x_1,x_2,\dots,x_d \mid y) = p(x_1 \mid y)p(x_2 \mid y)\dots p(x_d \mid y)$.
    - Unlike product of Bernoullis where we all variables are mutually independent.
  - "We assume the features are independent within each class."
    - Another view: we use a different product of Bernoullis for each class.
- How it this used within a generative classifier?

$$p(x_1, x_2, \dots, x_d, y) = p(x_1, x_2, \dots, x_d \mid y)\, p(y) \quad \text{(product rule)}$$

$$= p(x_1 \mid y)\, p(x_2 \mid y) \cdots p(x_d \mid y)\, p(y) \quad \text{(under naive Bayes assumption)}$$

conditional univariate density estimation

figuring this out is a univariate density estimation problem.

# Naïve Bayes Generative Classifier

- Naïve Bayes inference:
  - We have that $p(x_1,x_2,\ldots,x_d, y) = p(x_1 \mid y)p(x_2 \mid y)\ldots p(x_d \mid y)p(y)$.
  - Use $p(y \mid x_1,x_2,\ldots,x_d) \propto p(x_1,x_2,\ldots,x_d, y)$ (definition of conditional prob), to determine if $p(y = 1 \mid x_1,x_2,\ldots,x_d) > p(y = 0 \mid x_1,x_2,\ldots,x_d)$.
  - You could also do other inference tasks:
    - Normalization:
      - Sum up $p(x_1,x_2,\ldots,x_d,y)$ for y=1 and y=0 to get $p(x_1,x_2,\ldots,x_d)$ by the marginalzation rule.
    - Conditional decoding:
      - Find "most spammy" features possible: $\text{argmax}_{x1,\ldots,xd}\ p(x_1,\ldots,x_d \mid y = 1)$.
    - Find fewest words to add to your spam message that make it appear as non-spam.

# Conditional Binary Density Estimation

- To train naïve Bayes, we want to build a model of $p(x_j \mid y)$.
  - "Probability of this $x_j$, given the class label y".
- For binary $x_j$ and 'y', an obvious Bernoulli-like parameterization:

$$p(x_j = 1 \mid y = 1) = \theta_{j1}$$
$$p(x_j = 1 \mid y = 0) = \theta_{j0}$$

You can get the other probabilities from "sum to 1".

$$p(x_j = 0 \mid y = 1) = 1 - \theta_{j1}$$

  - For each 'j', this has 2 parameters:
    - $\theta_{jk}$: probability of $x_j$ being '1' when in class 'k'.
  - Given the 'y' value, this is a Bernoulli distribution.
    - Value 'y' causes you to "pick" between the two Bernoulli distributions.
    - With a fixed 'y', inference will work as it did for Bernoullis.
  - MLE is given by (exercise):

$$\hat{\theta}_{j1} = \frac{n_{j=1,1}}{n_1}$$

← number of times $x_j = 1$ and $y = 1$

← number of times $y = 1$

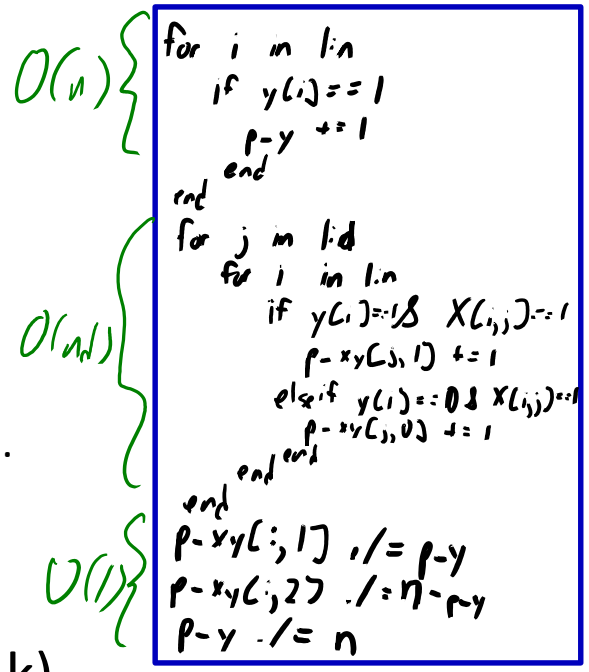
$$\hat{\theta}_{j0} = \frac{n_{j=1,0}}{n_0}$$

← number of times $x_j = 1$ and $y = 0$

← number of times $y = 0$

# Generative Classifier: Implementation

- Training phase for a generative classifier:
  1. Fit parameters of p(y).
     - For binary 'y', use Bernoulli and do MLE/MAP.
  2. For each class 'k':
     - Fit parameters of $p(x_1, x_2, \ldots, x_d \mid y = k)$ using examples in class 'k'.
       - For naïve Bayes, fit $p(x_1 \mid y = k)$, then fit $p(x_2 \mid y = k)$,…, and finally fit $p(x_d \mid y = k)$.
         - » You can view this as fitting a product of Bernoullis model for each class.

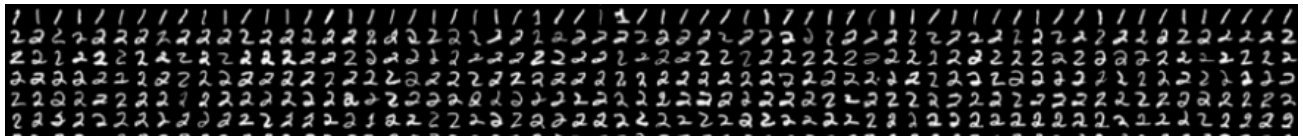



- Cost for naïve Bayes is O(nd):
  - O(n) to fit p(y), O(n) to fit each of the 'd' parameters of p(x | y = k).
  - Can be reduced to O(z) if 'X' only has 'z' non-zeroes.

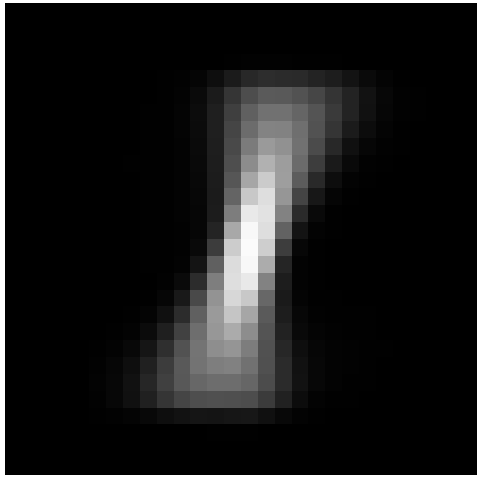
- Inference phase for generative classifier:
  - Use $p(y \mid x) \propto p(x, y)$ to get probabilities for different classes.

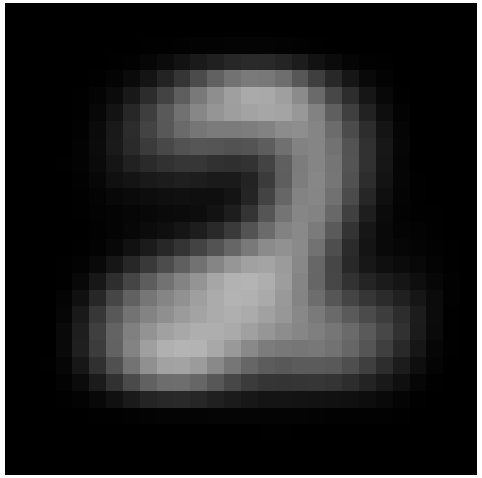
# Naïve Bayes on MNIST

- Consider fitting naïve Bayes on MNIST digits to distinguish "1" vs. "2".
  - Binary supervised learning problem.



- There are 6742 "1" examples and 5958 "2" examples.
  - So with MLE we have: p(y=1) = 6742/(6742+5958), or p(y=1) ≈ 0.53.
- Visualizing the $p(x_j \mid y)$ parameters for each class:



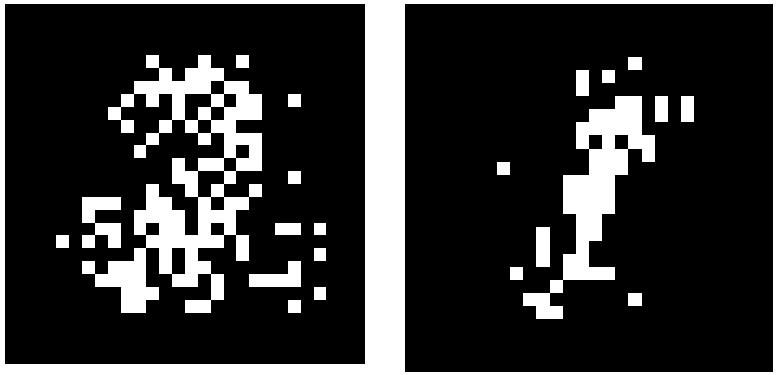


  - These are the product of Bernoullis models for each class.

https://www.kaggle.com/tarunkr/digit-recognition-tutorial-cnn-99-67-accuracy

# Naïve Bayes on MNIST

- To sample from naïve Bayes model:
  - Sample a value '$\tilde{y}$' from p(y), then independently sample each $x_j$ from $p(x_j \mid \tilde{y})$.
    - "First sample whether the number will be a 1 or 2, then sample each pixel independently."
    - We will explain why this works later when we cover "ancestral sampling".
- Two samples from a naïve Bayes model:



- Still a bad model, but they at least now look a bit like digits.
  - For naïve Bayes to classify well, we do not need a perfect density estimator.
    - It might have learned enough to say that images of 2s are more likely to be 2s than 1s, even though it does not have a perfect model of either class.
    - This is why naïve Bayes could accurately classify e-mail spam, even though the product of Bernoullis model is one of the worst density estimators.

# Generative Classifiers - Discussion

- At the moment, generative classifiers are really unpopular.
  - Historically, you need to make a strong assumption like in naïve Bayes.
    - For "real" images, independence assumption makes the model basically useless.

- Instead of modeling $p(x_1, x_2,\dots,x_d, y)$ ("generative model"), we now directly model $p(y \mid x_1, x_2,\dots,x_d)$ ("discriminative model", our next topic).
  - And maybe use a neural network to learn a non-linear mapping (next next topic).

- But this might change in the future:
  - May be able to learn effective classifiers with less data.
    - Discriminative: "find a way to combine the pixels to explain why this is a dog."
    - Generative: "this is an image of a dog, explain every pixel in the image".
  - Modern density estimation methods work much better than classic methods.

# Next Topic: Discriminative Classifiers

# Discriminative Classifiers

- Discriminative classifiers directly model $p(y \mid x_1, x_2, \ldots, x_d)$.
  - Might be easier than modeling $p(x_1, x_2, \ldots, x_d, y)$ as done in generative classifiers.

- Key advantage:
  - Only need to figure out how features affect the label.
    - Do not need to model the features, which themselves could be complicated.
    - Do not model p(y) either, we only focus on the mapping from 'x' to 'y'.

- Simple example: a dataset with a binary label and one binary feature.
    - For example, predict "hospitalization" based on "vaccinated".
      - We only focus on predicting "hospitalization" wit a known value of "vaccinated", and ignore p("vaccinated").
  - Conditional binary parameterization (like we did with naïve Bayes):
    - $p(y = 1 \mid x = 1) = \theta_1$.
    - $p(y = 1 \mid x = 0) = \theta_0$.
    - Feature 'x' "switches" between 2 Bernoulli distributions for 'y'.
  - Fit with MLE/MAP, compute $p(y \mid x)$ for new examples directly from relevant Bernoulli.
    - But can't do inferences about 'x', since does not model 'x'.

# Tabular Parameterization of Conditionals

- Now consider a dataset with binary label and 2 binary features.
  - For example, predict “hospitalization” based on “vaccinated” and “Paxlovid”.
  - The tabular parameterization of the conditional probability:
    - $p(y = 1 \mid x_1 = 0, x_2 = 0) = \theta_{00}$.
    - $p(y = 1 \mid x_1 = 0, x_2 = 1) = \theta_{01}$.
    - $p(y = 1 \mid x_1 = 1, x_2 = 0) = \theta_{10}$.
    - $p(y = 1 \mid x_1 = 1, x_2 = 1) = \theta_{11}$.
    - Makes a different Bernoulli for each combination of ‘x’ values.
  - Basic probability question: why do we need 4 parameters here and not only 3?
- Advantage of tabular representation:
  - Can represent any binary conditional (no restriction on distribution).
- Disadvantage of tabular representation:
  - With ‘d’ features we need $2^d$ parameters.

# Linear Parameterization of Conditionals

- Tabular parameterization will overfit when you have many features.
  - You may not see some of the $2^d$ combinations of features in training data.

- Common solution: use a "parsimonious" parameterization.
  - "Parsimonious": has fewer parameters.
  - Hope to need less data by giving up the ability to model any conditional.
- Standard choice parameterizes a linear combination of features:

$$p(y = 1 \mid x_1, x_2, \ldots, x_d, w) = f(w_1 x_1 + w_2 x_2 + \ldots + w_d x_d) = f(w^T x)$$

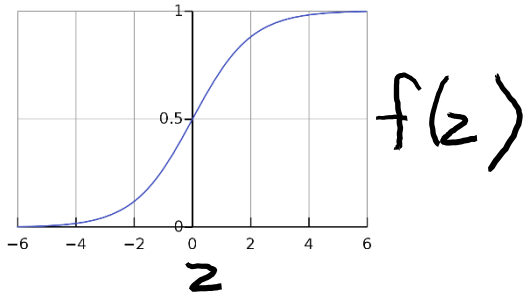
Function 'f' maps from reals $\mathbb{R}$ to $[0, 1]$

parameter $w_1$ is the "weight" on $w_1$.

# Sigmoid Function and Logistic Regression

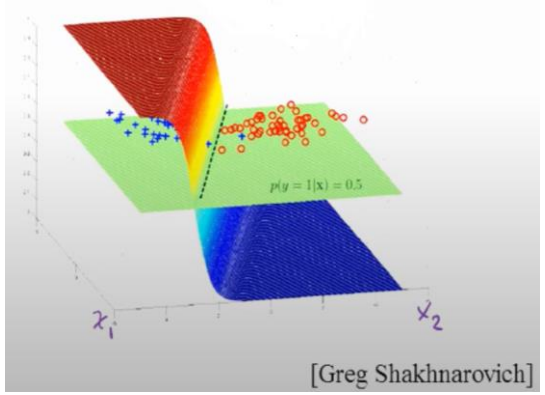
- Sigmoid function is common choice for mapping (-∞,∞) to [0,1]:

$$f(z) = \frac{1}{1+\exp(-z)}$$




- Using sigmoid to model conditional based on linear combination:

$$p(y=1 \mid x, w) = f(w^T x) = \frac{1}{1+\exp(-w^T x)}$$




- This model is called logistic regression.
  - Usually fit with MLE or MAP.
  - Works well in many applications (usually beats naïve Bayes).

https://en.wikipedia.org/wiki/Sigmoid_function
https://www.youtube.com/watch?v=Zc7ouSD0DEQ

# Inference in Logistic Regression

- For fixed ‘w’ and ‘x’, logistic gives binary distribution over $y^i$ values:

$$p(y^i = 1 \mid x^i, w) = \underbrace{\frac{1}{1 + \exp(-w^T x^i)}}_{\text{“}\theta\text{”}}$$

  - Cost for one example is O(d), due to the inner product $w^T x$.
- You can treat this value as the parameter “$\theta$” in a Bernoulli.
  - If $w^T x > 0$ then $\theta > 0.5$, and if $w^T x < 0$ then $\theta < 0.5$.
  - Usually we just do decoding of this distribution to predict most likely ‘y’.
  - But you could then do inference conditioned on the values of the features ‘x’.
    - Sample values of ‘y’ given this value of ‘x’.
    - Compute probability of seeing 5 examples with y=1 among 10 examples for this ‘x’.
    - Compute the number of samples with these features before expect to get one with y=1.
    - Use “decision theory” to make predictions that maximize utility.
    - And so on.

```
θ = 1/(1+exp(-X[i,:]*w))
```

# Maximum Likelihood or Conditional Likelihood?

- MLE in generative compared to discriminative models:
  - In generative models, MLE maximizes $p(X, y \mid w)$.
  - In discriminative models, MLE maximizes $p(y \mid X, w)$.
    - We maximize the conditional likelihood of 'y' (conditioning on features).
      - And we treat the features 'X' as fixed.

- Logistic regression can use binary or continuous features in 'x'.
  - Even though it only uses binary probabilities.
- This is different than we saw with naïve Bayes:
  - Naïve Bayes needed independence assumption even for binary features.
    - Naïve Bayes would need to model continuous probabilities for continuous features.

# Review: Logistic "Negative Log-Likelihood"

- With 'n' training examples, logistic regression NLL is:

$$f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y^i w^T x^i))$$

  - Where for logistic we will assume $y^i \in \{-1, +1\}$ rather than usual {0,1}.
    - Equivalent to what some people call "binary cross entropy".
  - Cost: O(nd), bottleneck is computing the 'n' $w^T x^i$ values for O(d) each.
    - Code to compute 'f' and its gradient 'g':
      - The $w^T x^i$ values are computed via matrix multiplication "X*w".

```
function logisticObj(w,X,y)
    yXw = y.*(X*w)
    f = sum(log.(1 .+ exp.(-yXw)))
    g = -X'*(y./(1 .+ exp.(yXw)))
    return (f,g)
end
```

- This is a convex function, so if $\nabla f(w) = 0$ then w is global minimum.
- Setting $\nabla f(w) = 0$ does not lead to closed-form solution for 'w'.
- But since 'f' is differentiable and convex,
  we can converge to a 'w' with $\nabla f(w) = 0$ by using gradient descent.
  - Or stochastic gradient descent depending on 'n' and desired accuracy.

# Review: Regularization and MAP

- Common to add a regularizer, such as L2-regularization, to the NLL:

$$f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y^i w^T x^i)) + \frac{\lambda}{2}\|w\|^2$$

  - Typically gives better test error with appropriate hyper-parameter $\lambda > 0$.
  - L2-regularization corresponds to MAP estimation with a Gaussian prior.
    - We will cover Gaussians later.

- In both generative/discriminative cases, MAP maximizes posterior:

$$\hat{w} \in \operatorname{argmax}_w \{p(w \mid X, y)\}$$

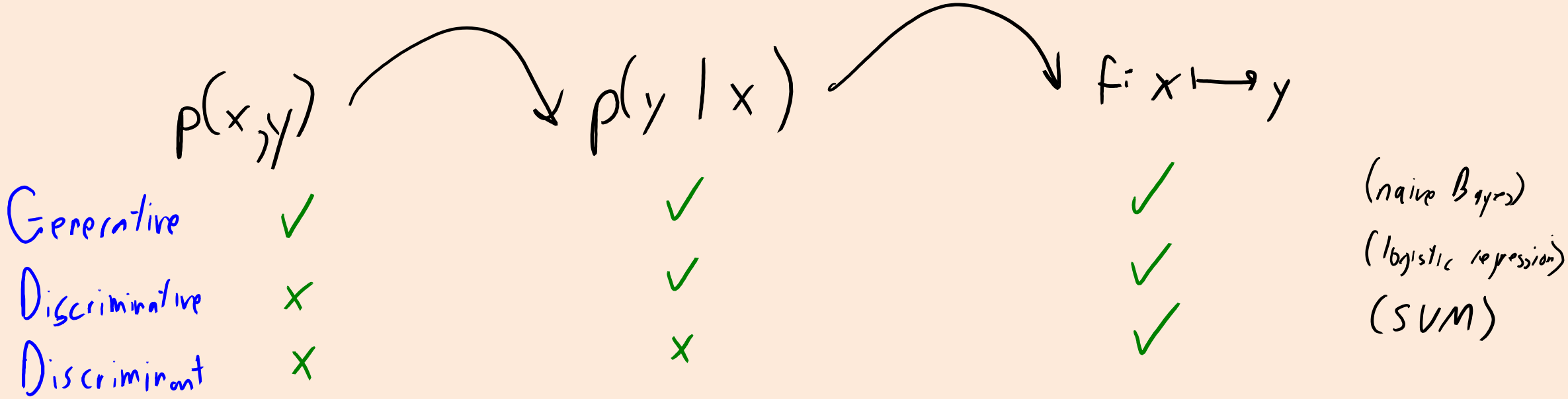
generative: $\equiv \operatorname{argmax}_w \{p(y, X \mid w) p(w)\}$

discriminative: $\equiv \operatorname{argmax}_w \{p(y \mid X, w) p(w)\}$

(assumes $X$ is independent of $w$)

# Generative vs. Discriminative vs. Discriminant

- Also exists "discriminant function" models, such as support vector machines (SVMS):
  - They don't use probabilities but instead try to directly learn map from 'x' to 'y'.




- Accuracy is often higher as you model fewer steps (but not always).
  - But number of inference tasks you can do gets more limited.
    - Discriminative models cannot answer questions involving p(x, y).
    - Discriminant functions cannot answer questions involving p(y | x).

# Summary

- Naïve Bayes:
  - Generative classifier awhere product of Bernoullis is used for p(x | y).
- Discriminative Classifiers:
  - Directly model p(y | x) rather than p(x ,y).
  - Most of modern machine learning is based on discriminative classifiers.
- Tabular parameterization:
  - Fit a parameter for p(y=1 |x) for each possible value of ‘x’.
  - Can model any conditional, but overfits unless ‘d’ is small.
- Logistic regression:
  - Write p(y | x) using the sigmoid function.
  - MLE is a convex optimization problem.
  - Trained using variations on gradient descent.
  - Cannot model any conditional, but tends not to overfit (especially with regularization).
- Fundamental Trade-Off:
  - Learning theory says that simple models do not overfit but may underfit.
  - Learning theory says that complicated models do not unferfit but may overfit.

- Next time: are we really going to get to deep learning in Week 2?

# Logistic Regression Training Code

- Gradient descent for logistic regression:

$$w^{k+1} = w^k - \alpha^k \nabla f(w^k)$$

$$\nabla f(w^k) = X^T r \quad \text{where} \quad r^i = \frac{-y^i}{1+\exp(y^i w^T x^i)}$$

- Simple method for setting the step size:
  - If $f(w^{k+1}) > f(w^k)$, divide $\alpha$ in half and see if that decreases 'f'.
    - There are much-more clever ways to set the step size (for example, Barzilai-Borwein method in *findMin*).
    - There are also better "directions" than using the gradient, such as quasi-Newton and Hessian-free Newton.
    - For stochastic gradient descent, you need a decreasing set of step sizes to guarantee convergence.
- Deciding when to stop:
  - Check if $||\nabla f(w)|| \leq \epsilon$ for some small $\epsilon$.
  - Or check for progress in function/iteration values, and "give up" if you no longer are making progress.
- Cost is O(nd) per iteration.
  - Computing each of 'n' inner-product $w^T x^i$ costs O(d), giving O(nd).
  - Computing $X^T r$ in the gradient costs O(nd).
  - Updating w given the gradient costs O(d) so does not increase cost.
- If the matrix 'X' only has 'z' non-zero values, can be implemented in O(z).
- Cost is only O(d) for stochastic gradient descent, but you will spend a lot of time tuning step sizes.