CPSC 340: Machine Learning and Data Mining

Naïve Bayes classification

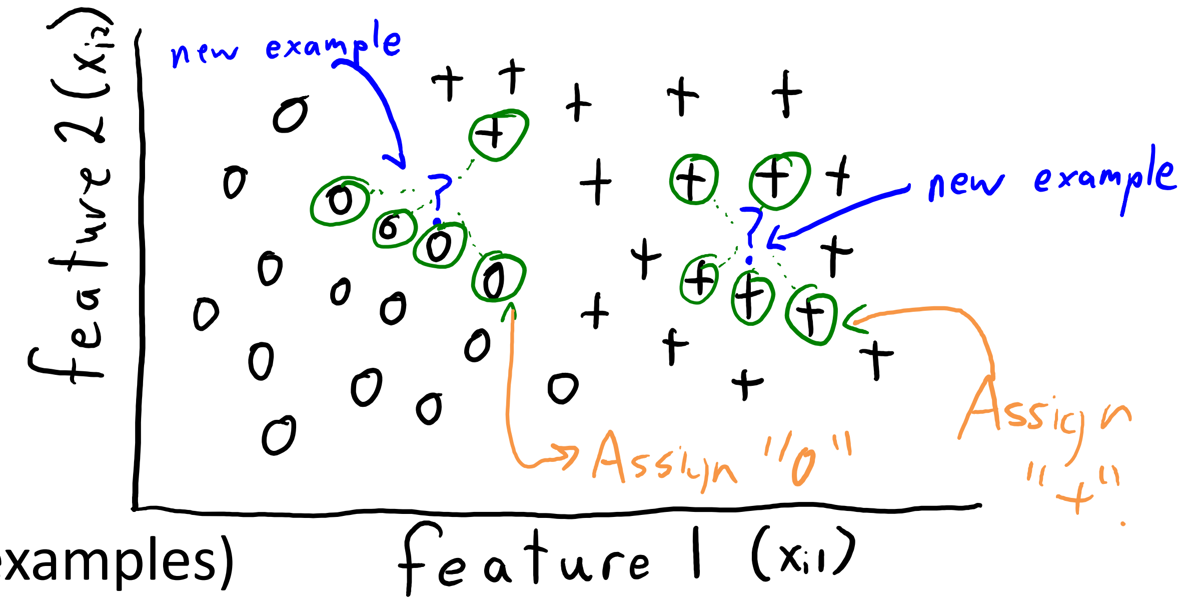
Original version of these slides by Mark Schmidt, with modifications by Mike Gelbart.

Admin

- Assignment 0 solutions posted
- Assignment 1 due Wednesday
  - You should have started by now.
- Assignment 2 released by the end of the week
- Add/drop deadline is Wednesday

Last Time: K-Nearest Neighbours (KNN)

- K-nearest neighbours algorithm for classifying $\tilde{x}_i$:
  - Find ‘k’ values of x_i that are most similar to $\tilde{x}_i$.
  - Use mode of corresponding y_i.
- Lazy learning:
  - To “train” you just store X and y.
- Non-parametric:
  - Size of model grows with ‘n’ (number of examples)
  - Nearly-optimal test error with infinite data.
- But high prediction cost and may need large ‘n’ if ‘d’ is large.



KNN questions from Piazza

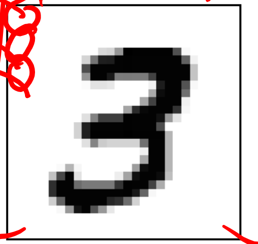
1. What does the red and blue transparent shading represent?
2. By looking at these plots, can you visually identify the training examples that are correctly and incorrectly classified?
3. Why is KNN "smoother" for larger k?
4. Why does KNN (almost) always get zero training error when k=1?
5. From the plots we see that KNN allows "islands" of one colour surrounded entirely by the other colour. Could such a thing happen with decision trees?
6. Why can’t KNN just predict by checking the shading colour for a test example instead of computing all those distances?

Application: Optical Character Recognition

- To scan documents, we want to turn images into characters:
  - "Optical character recognition" (OCR).

Application: Optical Character Recognition

- To scan documents, we want to turn images into characters:
  - "Optical character recognition" (OCR).

$x_i =$  → "3"

  - Turning this into a supervised learning problem (with 28 by 28 images):

$X =$

	(1,1)	(2,1)	(3,1)	…	(28,1)	(1,2)	(2,2)	…	(14,14)	…	(28,28)
x_i	0	0	0		0	0	0		1		0
	0	0	0		0	0	0		1		0
	0	0	0		0	0	0		0		0
	0	0	0		0	0	0		1		0

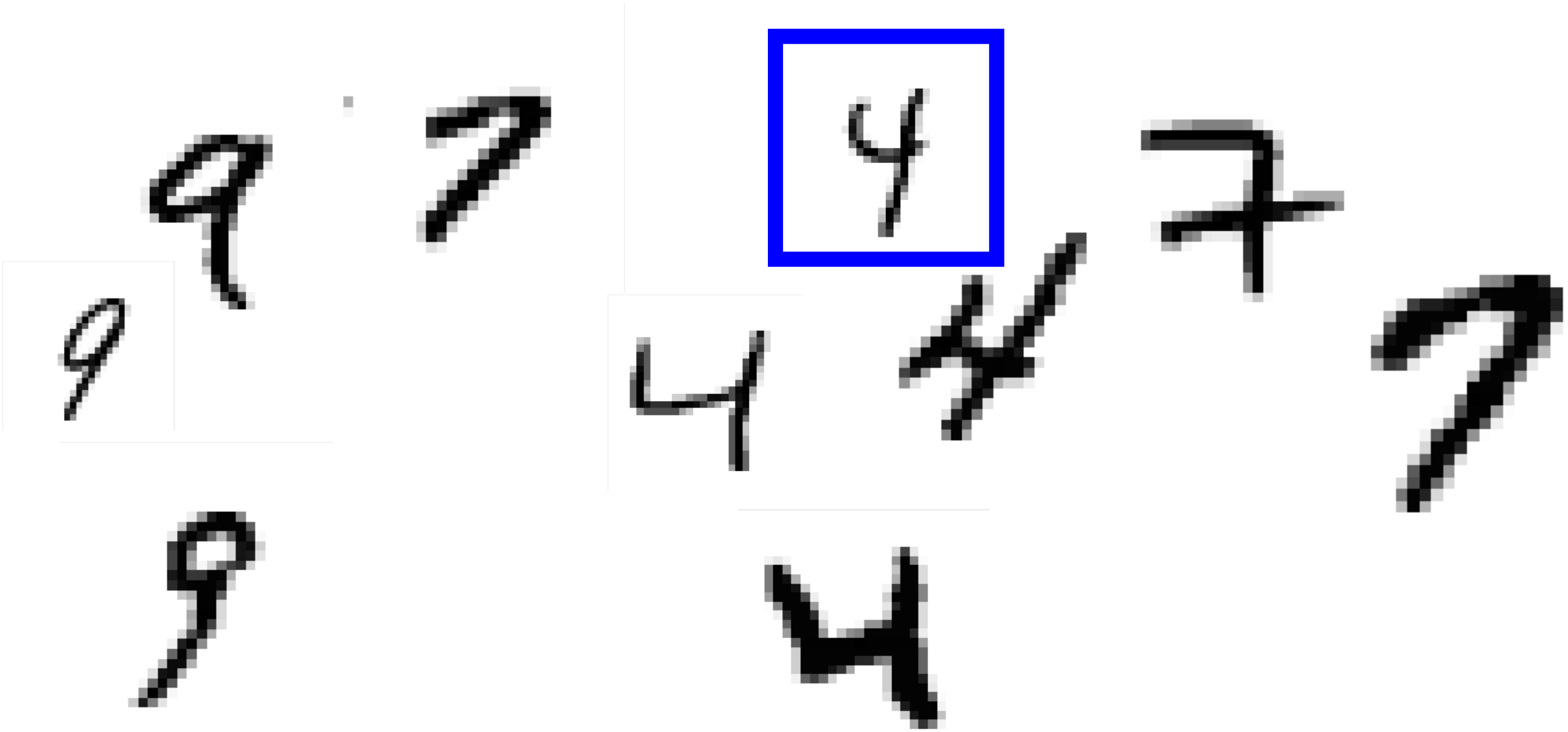
$\underbrace{\qquad}_{\text{Each feature is grayscale intensity of one of the 784 pixels}}$

⇒ $y =$

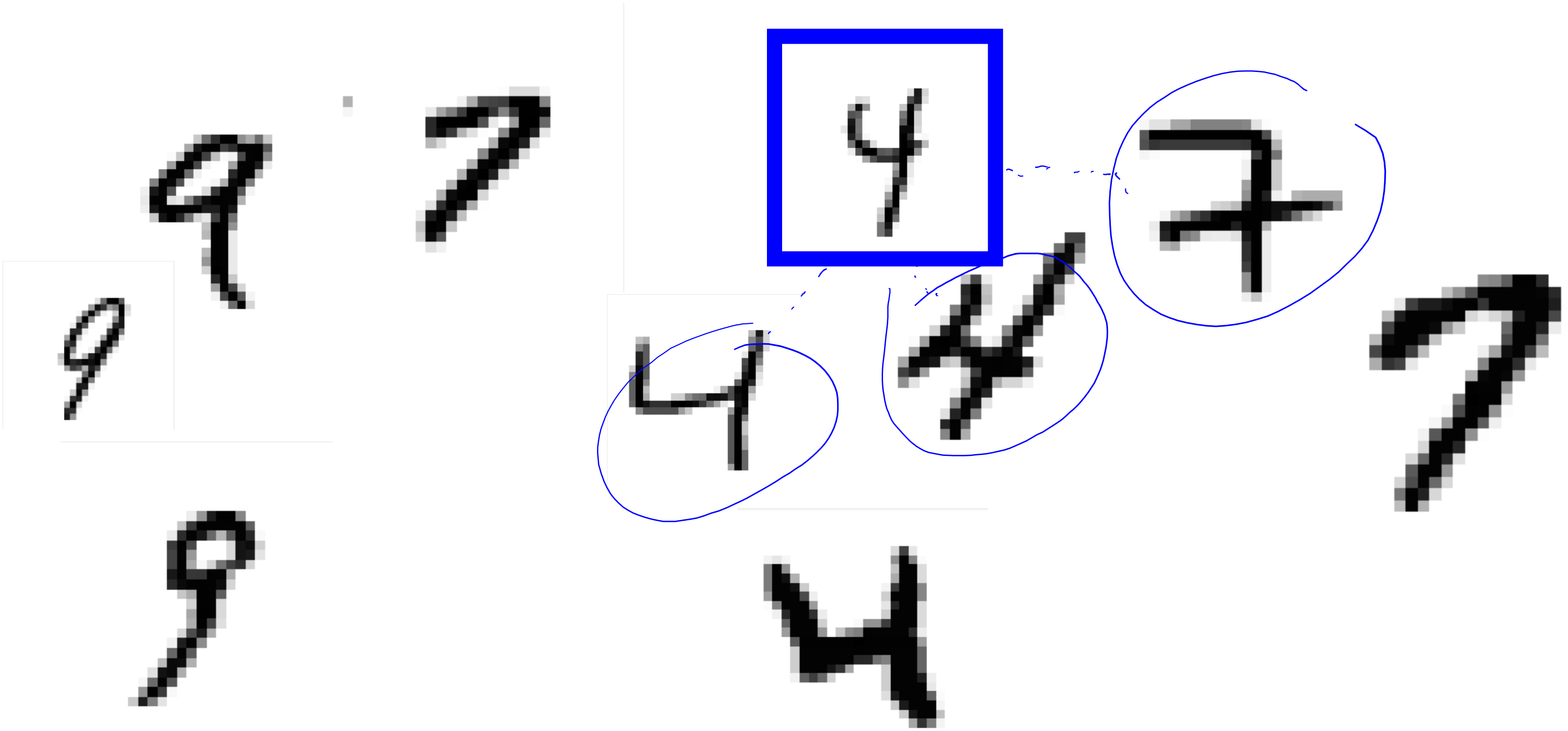
char
3
6
0
9

KNN for Optical Character Recognition

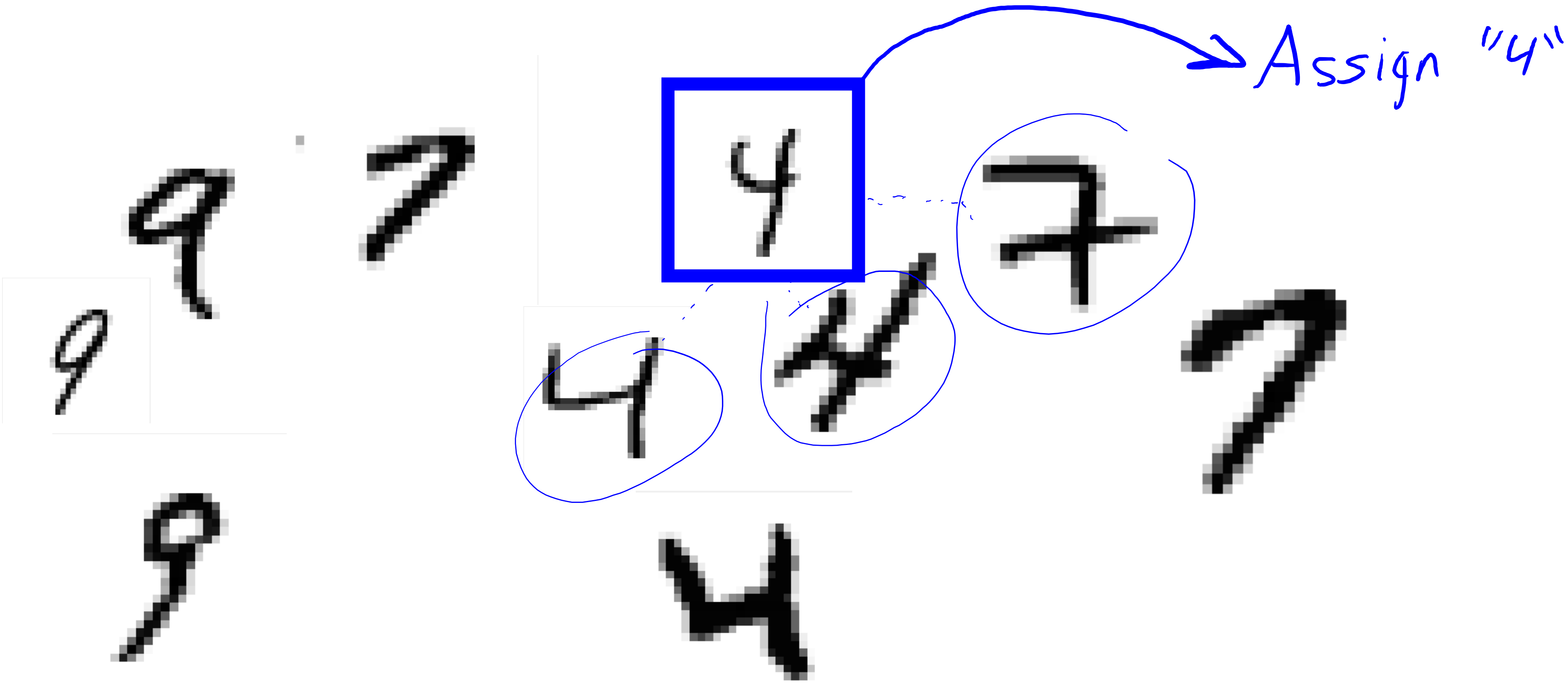
KNN for Optical Character Recognition



KNN for Optical Character Recognition



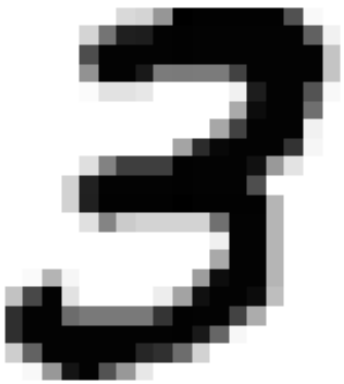
KNN for Optical Character Recognition



Human vs. Machine Perception

- There is huge difference between what we see and what KNN sees:

What we see:

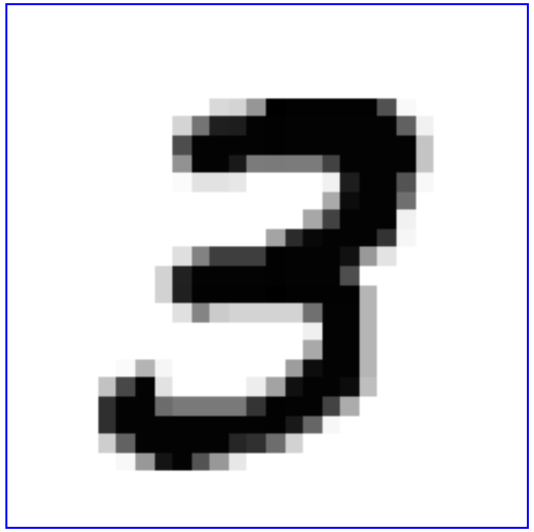


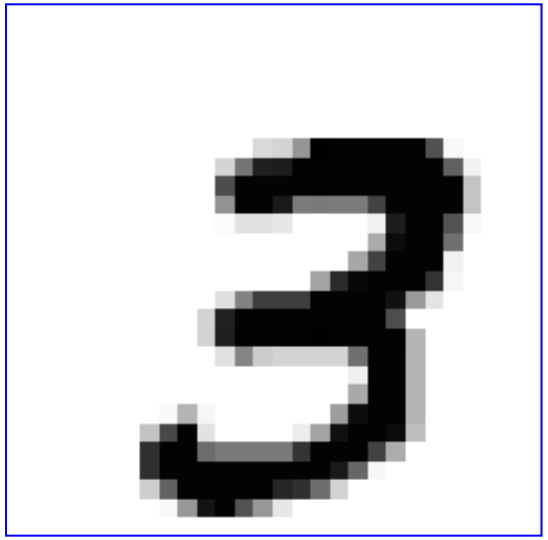
What the computer "sees":

Actually, it's worse:

What the Computer Sees

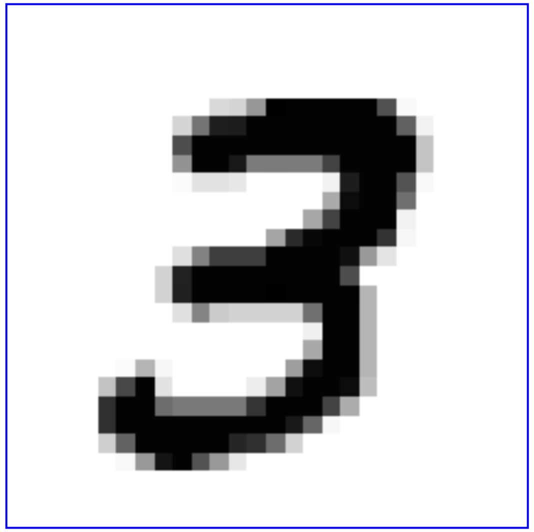
- Are these two images “similar”?

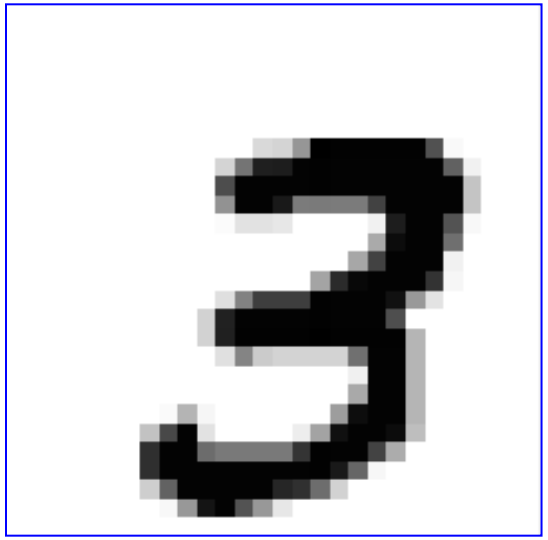




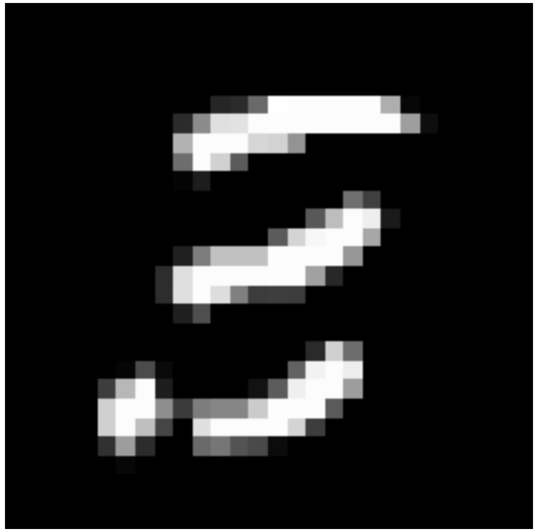
What the Computer Sees

- Are these two images “similar”?





Difference:

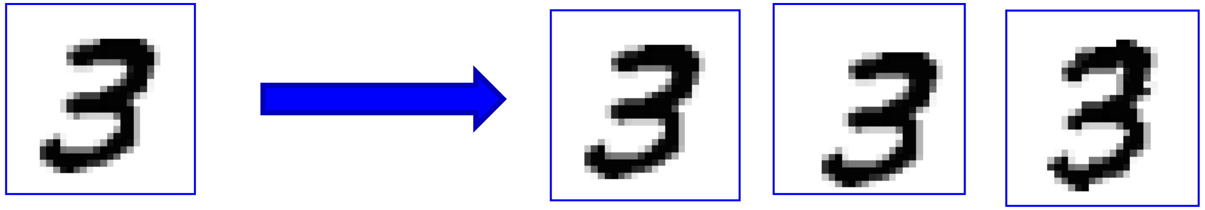


- KNN does not know that labels should be translation invariant.

Encouraging Invariance

- May want classifier to be invariant to certain feature transforms.
  - Images: translations, small rotations, changes in size, mild warping,…
- The hard/slow way is to modify your distance function:
  - Find neighbours that require the ‘smallest’ transformation of image.

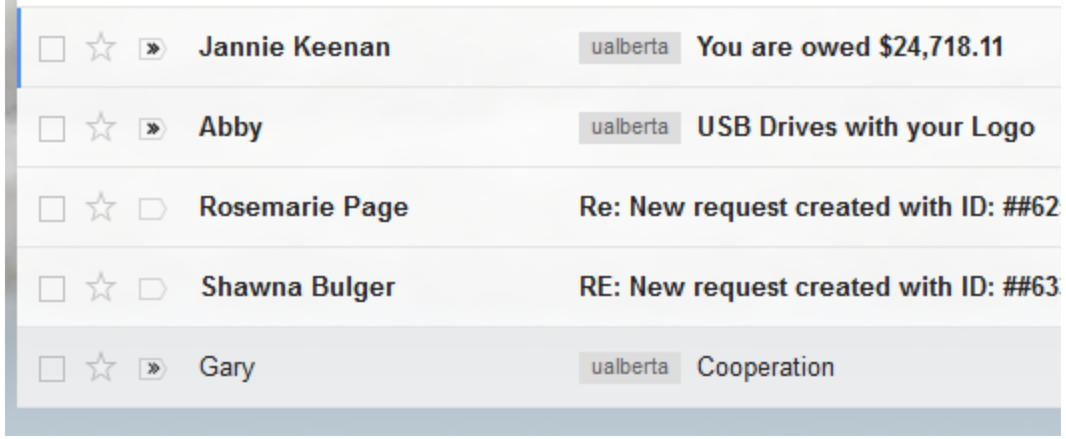
- The easy/fast way is to just add transformed data during training:
  - Add translated/rotate/resized/warped versions of training images.

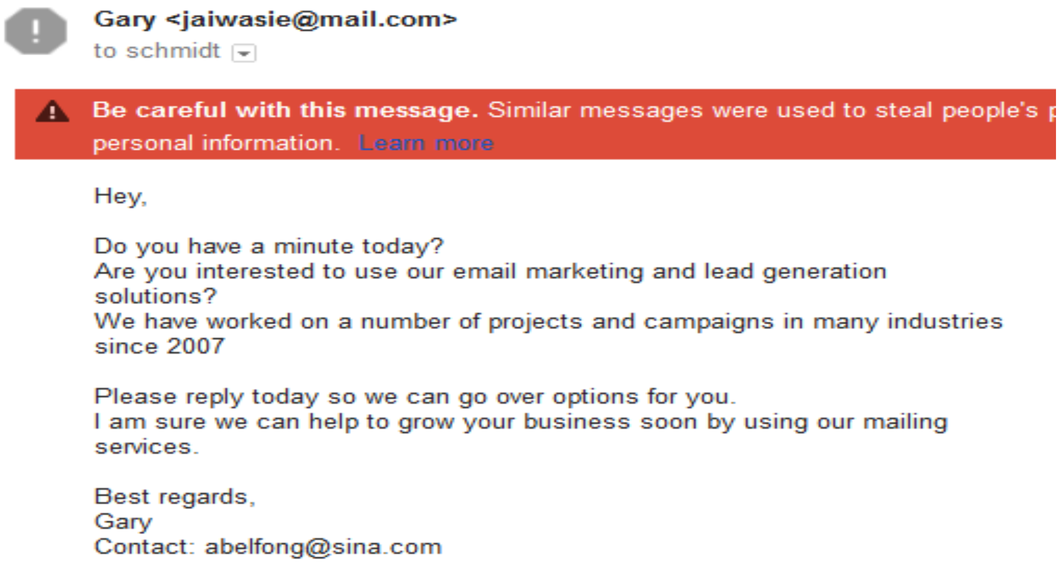


  - Crucial part of many successful vision systems.
  - Bonus slides discuss invariant features for language data.

Application: E-mail Spam Filtering

- Want a build a system that detects spam e-mails.
  - Context: spam used to be a big problem.





- Can we formulate as supervised learning?

Spam Filtering as Supervised Learning

- Collect a large number of e-mails, gets users to label them.

$	Hi	CPSC	340	Vicodin	Offer	…	Spam?
1	1	0	0	1	0	…	1
0	0	0	0	1	1	…	1
0	1	1	1	0	0	…	0
…	…	…	…	…	…	…	…

- We can use ($y_i = 1$) if e-mail 'i' is spam, ($y_i = 0$) if e-mail is not spam.
- Extract features of each e-mail (like bag of words).
  - ($x_{ij} = 1$) if word/phrase 'j' is in e-mail 'i', ($x_{ij} = 0$) if it is not.

Feature Representation for Spam

- Are there better features than bag of words?
  - We add bigrams (sets of two words):
    - "CPSC 340", "wait list", "special deal".
  - Or trigrams (sets of three words):
    - "Limited time offer", "course registration deadline", "you're a winner".
  - We might include the sender domain:
    - <sender domain == "mail.com">.
  - We might include regular expressions:
    - <your first and last name>.

- Also, note that we only need list of non-zero features for each x_i.

Review of Supervised Learning Notation

- We have been using the notation ‘X’ and ‘y’ for supervised learning:

$X =$

$	Hi	CPSC	340	Vicodin	Offer	...
1	1	0	0	1	0	...
0	0	0	0	1	1	...
0	1	1	1	0	0	...
...	...	...	...	...	...	...

x_{26}

x_3

$y =$

Spam?
1
1
0
...

y_3

- X is matrix of all features, y is vector of all labels.
  - We use y_i for the label of object ‘i’ (element ‘i’ of ‘y’).
  - We use x_{ij} for feature ‘j’ of object ‘i‘.
  - We use x_i as the list of features of object ‘i’ (row ‘i’ of ‘X’).
    - So in the above x_3 = [0 1 1 1 0 0 …].

Probabilistic Classifiers

- For years, best spam filtering methods used naïve Bayes.
  - A probabilistic classifier based on Bayes rule.
  - It tends to work well with bag of words.
  - Last year shown to improve on state of the art for CRISPR "gene editing" (link).

- Probabilistic classifiers model the conditional probability, $p(y_i \mid x_i)$.
  - "If a message has words x_i, what is probability that message is spam?"

- Classify it has spam if probability of spam is higher than not spam:
  - If $p(y_i = \text{"spam"} \mid x_i) > p(y_i = \text{"not spam"} \mid x_i)$
    - return "spam".
  - Else
    - return "not spam".

Spam Filtering with Bayes Rule

- To model conditional probability, naïve Bayes uses Bayes rule:

$$p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)}$$

- So we need to figure out three types of terms:
  - Marginal probabilities $p(y_i)$ that an e-mail is spam.
  - Marginal probability $p(x_i)$ that an e-mail has the set of words x_i.
  - Conditional probability $P(x_i \mid y_i)$ that a spam e-mail has the words x_i.
    - And the same for non-spam e-mails.

Spam Filtering with Bayes Rule

$$p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)}$$

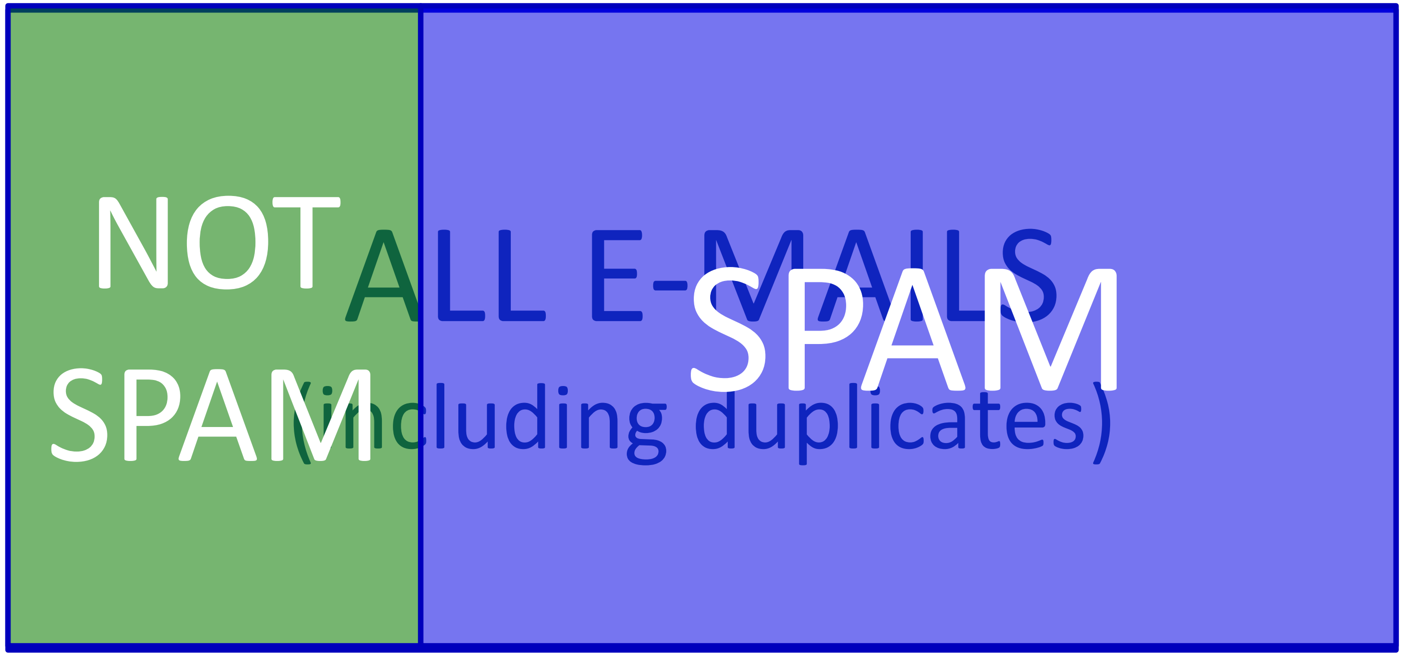
- What do these terms mean?

ALL E-MAILS
(including duplicates)

Spam Filtering with Bayes Rule

$$p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"})\,p(y_i = \text{"spam"})}{p(x_i)}$$

- $p(y_i = \text{“spam”})$ is probability that a random e-mail is spam.
  - This is easy to approximate from data: use the proportion in your data.



$$p(y_i = \text{"spam"}) = \frac{\#\text{ spam messages}}{\#\text{ total messages}}$$

This is a “maximum likelihood estimate”, a concept we’ll discuss in detail later. If you’re interested in a proof, see here.

Spam Filtering with Bayes Rule

$$p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)}$$

- $p(x_i)$ is probability that a random e-mail has features x_i:
  - This is hard to approximate (there are so many possible e-mails).

$$p(x_i) = \frac{\#\text{ e-mails with features } x_i}{\#\text{ e-mails total}}$$

Spam Filtering with Bayes Rule

$$p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)}$$

- $p(x_i)$ is probability that a random e-mail has features x_i:
  - This is hard to approximate (there are so many possible e-mails), but it turns out we can ignore it:

Naive Bayes returns "spam" if $p(y_i = \text{"spam"} \mid x_i) > p(y_i = \text{"not spam"} \mid x_i)$.

By Bayes rule this means $\frac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)} > \frac{p(x_i \mid y_i = \text{"not spam"})\, p(y_i = \text{"not spam"})}{p(x_i)}$

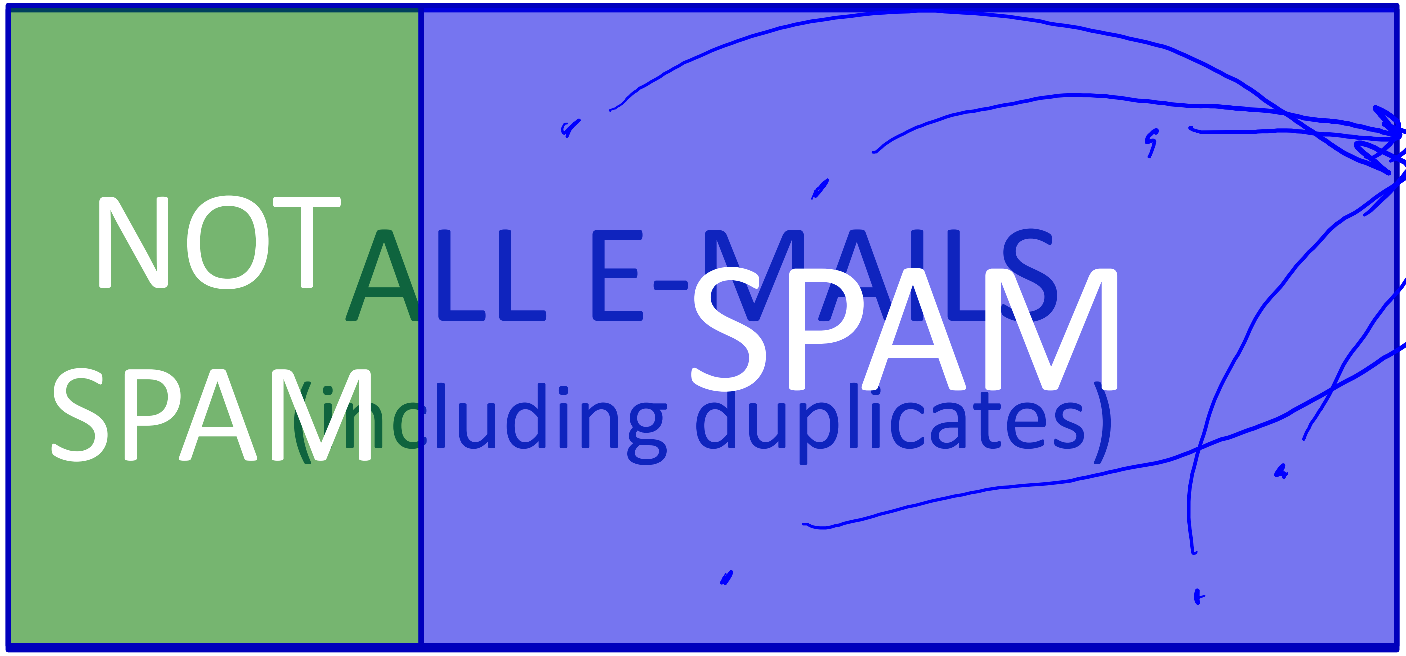
Multiply both sides by $p(x_i)$:

$$p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"}) > p(x_i \mid y_i = \text{"not spam"})\, p(y_i = \text{"not spam"})$$

Spam Filtering with Bayes Rule

$$p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"})\, p(y_i = \text{"spam"})}{p(x_i)}$$

- $p(x_i \mid y_i = \text{"spam"})$ is probability that spam has features x_i.



$$p(x_i \mid y_i = \text{"spam"}) = \frac{\#\ \text{spam messages with features } x_i}{\#\ \text{spam messages}}$$

- Also hard to estimate.
  - And we need it.

Naïve Bayes

- Naïve Bayes makes a big assumption to make things easier:

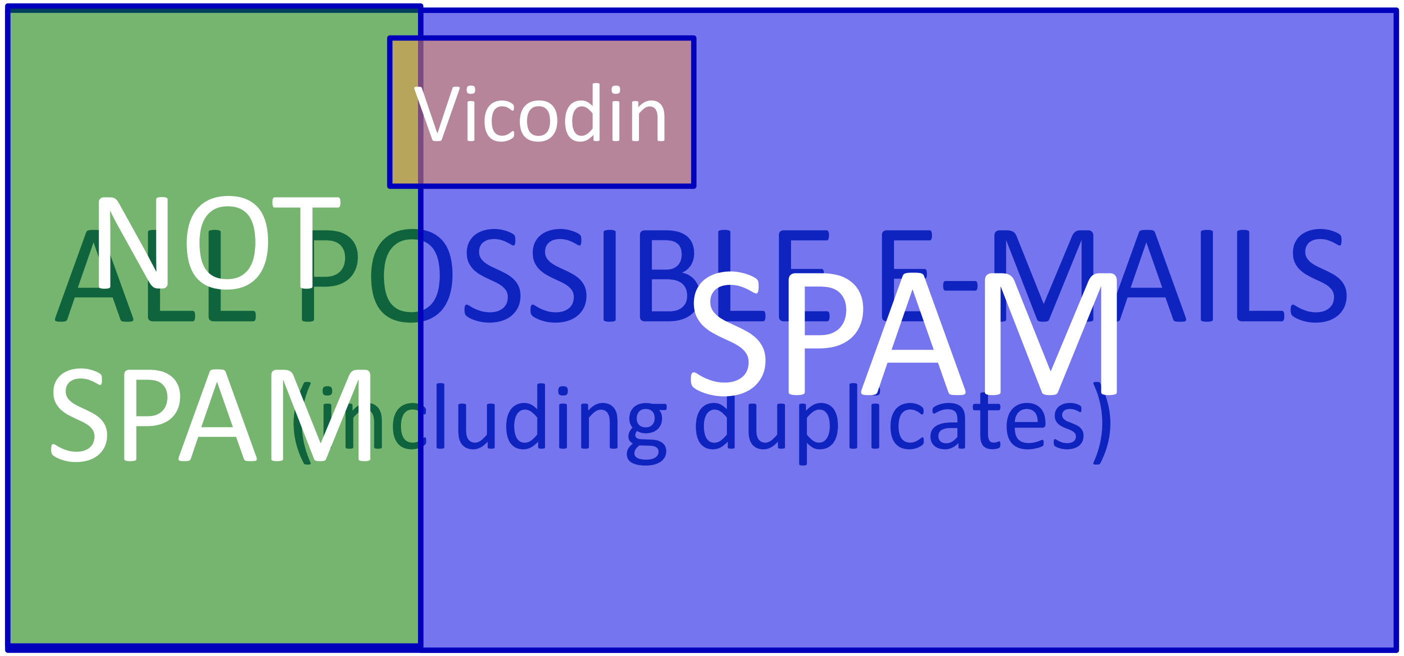
$$\underbrace{p(\text{hello}, \text{vicodin}, \text{CPSC } 340 \mid \text{spam})}_{\text{HARD}} \approx \underbrace{p(\text{hello} \mid \text{spam})}_{\text{easy}}\underbrace{p(\text{vicodin} \mid \text{spam})}_{\text{easy}}\underbrace{p(\text{CPSC } 340 \mid \text{spam})}_{\text{easy}}$$

- We assume *all* features x_i are conditionally independent give label y_i.
  - Once you know it's spam, probability of "vicodin" doesn't depend on "CPSC 340".
  - Definitely not true, but sometimes a good approximation.
  - Allows a training email with "vicodin" to influence all test emails with "vicodin".

- And now we only need easy quantities like p("vicodin" = 1| y_i = "spam").

Naïve Bayes

- p(“vicodin” = 1 | “spam” = 1) is probability of seeing “vicodin” in spam.



- Easy to estimate:

$$p(\text{vicodin}=1 \mid \text{spam}=1) = \frac{\#\ \text{spam messages w/ vicodin}}{\#\ \text{spam messages}}$$

Naïve Bayes

- Naïve Bayes more formally:

$$p(y_i | x_i) = \frac{p(x_i | y_i)p(y_i)}{p(x_i)} \quad \text{(first use Bayes rule)}$$

$$\propto p(x_i | y_i) p(y_i) \quad \text{("denominator doesn't matter")}$$

$$\approx \underbrace{\prod_{j=1}^{d} \left[p(x_{ij} | y_i) \right] p(y_i)}_{\text{Only needs easy probabilities.}} \quad \text{(conditional independence assumption)}$$

Laplace Smoothing

- Our estimate of p('lactase' = 1| 'spam') is:

$$\frac{\text{\# spam messages with lactase}}{\text{\# spam messages}}$$

  – Problem if you have no spam messages with lactase:
    - p('lactase' | 'spam') = 0, and message automatically gets through filter.
  – Common fix is Laplace smoothing estimate:
    - Add 1 to numerator, and add 1 for each possible label to denominator.

$$\frac{(\text{\# spam messages with lactase}) + 1}{(\text{\# spam messages}) + 2}$$

    - A common variation is to use a different number β rather than 1.
  – This is like pretending you've seen 1 of everything before you start.

Decision Theory

- Are we equally concerned about "spam" vs. "not spam"?
- True positives, false positives, false negatives, false negatives:

Predict / True	True 'spam'	True 'not spam'
Predict 'spam'	True Positive	False Positive
Predict 'not spam'	False Negative	True Negative

- The costs mistakes might be different:
  - Letting a spam message through (false negative) is not a big deal.
  - Filtering a not spam (false positive) message will make users mad.

Decision Theory

- We can give a cost to each scenario, such as:

Predict / True	True 'spam'	True 'not spam'
Predict 'spam'	0	100
Predict 'not spam'	10	0

- Instead of most probable label, take yhat minimizing expected cost:

$$\underbrace{\mathbb{E}}_{\text{expectation of model with respect to } \tilde{y}_i}[\underbrace{\text{cost}(\hat{y}_i, \tilde{y}_i)}_{\text{cost of predicting } \hat{y}_i \text{ if it's really } \tilde{y}_i}]$$

- Even if "spam" has a higher probability, predicting "spam" might have a higher cost, so predict "not spam".

Decision Theory Example

Predict / True	True ‘spam’	True ‘not spam’
Predict ‘spam’	0	100
Predict ‘not spam’	10	0

- If for a test example we have p($\tilde{y}_i$ = “spam” | $\tilde{y}_i$) = 0.6, then:

$$\mathbb{E}[\text{cost}(\hat{y}_i = \text{"spam"}, \tilde{y}_i)] = p(\tilde{y}_i = \text{"spam"} \mid \tilde{x}_i)\,\text{cost}(\hat{y}_i = \text{"spam"}, \tilde{y}_i = \text{"spam"})$$
$$+\, p(\tilde{y}_i = \text{"not spam"} \mid \tilde{x}_i)\,\text{cost}(\hat{y}_i = \text{"spam"}, \tilde{y}_i = \text{"not spam"})$$
$$= (0.6)(0) + (0.4)(100) = 40$$

$$\mathbb{E}[\text{cost}(\hat{y}_i = \text{"not spam"}, \tilde{y}_i)] = (0.6)(10) + (0.4)(0) = 6$$

- Even though “spam” is more likely, we should predict “not spam”.

Other Performance Measures

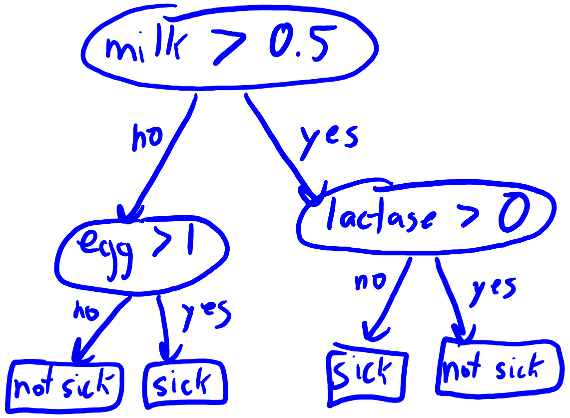
- Often, we report precision and recall (want both to be high):
  - Precision: "if I classify as spam, what is the probability it actually is spam?"
    - Precision = TP/(TP + FP).
    - High precision means the filtered messages are likely to really be spam.
  - Recall: "if a message is spam, what is probability it is classified as spam?"
    - Recall = TP/(TP + FN)
    - High recall means that most spam messages are filtered.
- Plotting precision vs. recall is a common performance visualization.
- See post-lecture bonus slides for more on this.

Unbalanced classes

- Some machine learning models don’t work well with unbalanced data.
  - If 99% of days you did not get sick, you get 99% accuracy by always predicting “not sick”
  - Decision theory approach can avoid this with high cost on false negatives
- See post-lecture bonus slides for more on this.

Decision Trees vs. Naïve Bayes

- Decision trees:



1. Sequence of rules based on 1 feature.
2. Training: 1 pass over data per depth.
3. Greedy splitting as approximation.
4. Testing: just look at features in rules.
5. New data: might need to change tree.
6. Accuracy: good if simple rules based on individual features work ("symptoms").

- Naïve Bayes:

$$p(\text{sick} \mid \text{milk}, \text{egg}, \text{lactase})$$
$$\approx p(\text{milk} \mid \text{sick})\, p(\text{egg} \mid \text{sick})\, p(\text{lactase} \mid \text{sick})\, p(\text{sick})$$

1. Simultaneously combine all features.
2. Training: 1 pass over data to count.
3. Conditional independence assumption.
4. Testing: look at all features.
5. New data: just update counts.
6. Accuracy: good if features almost independent given label (text).

Hyperparameters

- Decision trees: max depth (larger depth => more complex model)
- KNN: k (smaller k => more complex model)
- Naïve Bayes: β (smaller β => more complex model?)

Summary

- Probabilistic classifiers: try to estimate $p(y_i \mid x_i)$.
- Naïve Bayes: simple probabilistic classifier based on counting.
  - Uses conditional independence assumptions to make training practical.
- Decision theory allows us to consider costs of predictions.

- Post-lecture slides: how to train/test by hand on a simple example.

Naïve Bayes Training Phase

- Training a naïve Bayes model:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Naïve Bayes Training Phase

- Training a naïve Bayes model:

1. Set n_c to the number of times $(y_i = c)$.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$n_1 = 6$

$n_0 = 4$

Naïve Bayes Training Phase

- Training a naïve Bayes model:

1. Set n_c to the number of times $(y_i = c)$.
2. Estimate $p(y_i = c)$ as $\frac{n_c}{n}$.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$p(y_i = 1) = \frac{6}{10} \leftarrow n_1 = 6$

$p(y_i = 0) = \frac{4}{10} \leftarrow n_0 = 4$

Naïve Bayes Training Phase

- Training a naïve Bayes model:

1. Set n_c to the number of times $(y_i = c)$.
2. Estimate $p(y_i = c)$ as $\frac{n_c}{n}$.
3. Set n_{cjk} as the number of times $(y_i = c, x_{ij} = k)$

$$p(y_i = 1) = \frac{6}{10} \leftarrow n_1 = 6$$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$n_{121} = 4$$

$$p(y_i = 0) = \frac{4}{10} \leftarrow n_0 = 4$$

Naïve Bayes Training Phase

- Training a naïve Bayes model:

1. Set n_c to the number of times $(y_i = c)$.
2. Estimate $p(y_i = c)$ as $\frac{n_c}{n}$.
3. Set n_{cjk} as the number of times $(y_i = c, x_{ij} = k)$
4. Estimate $p(x_{ij} = k, y_i = c)$ as $\frac{n_{cjk}}{n}$

$$p(y_i = 1) = \frac{6}{10} \leftarrow n_1 = 6$$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$n_{121} = 4$

$p(x_{i1} = 1, y_i = 1) = \frac{4}{10}$

$$p(y_i = 0) = \frac{4}{10} \leftarrow n_0 = 4$$

Naïve Bayes Training Phase

- Training a naïve Bayes model:

1. Set n_c to the number of times $(y_i = c)$.
2. Estimate $p(y_i = c)$ as $\frac{n_c}{n}$.
3. Set n_{cjk} as the number of times $(y_i = c, x_{ij} = k)$
4. Estimate $p(x_{ij} = k, y_i = c)$ as $\frac{n_{cjk}}{n}$
5. Use that $p(x_{ij} = k \mid y_i = c) = \frac{p(x_{ij} = k, y_i = c)}{p(y_i = c)}$

$$= \frac{n_{cjk}/n}{n_c/n} = \frac{n_{cjk}}{n_c}$$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$p(y_i = 1) = \frac{6}{10} \leftarrow n_1 = 6$

$n_{121} = 4$

$p(x_{i1} = 1, y_i = 1) = \frac{4}{10}$

$p(x_{i1} = 1 \mid y_i = 1) = \frac{4}{6} = \frac{2}{3}$

$p(y_i = 0) = \frac{4}{10} \leftarrow n_0 = 4$

Naïve Bayes Prediction Phase

- Prediction in a naïve Bayes model:

Given a test example $\tilde{x}_i$ we want to find the 'c' maximizing $p(\tilde{x}_i \mid \tilde{y}_i = c)$

Under the naive Bayes assumption we can maximize:

$$p(\tilde{y}_i = c \mid \tilde{x}_i) \propto \prod_{j=1}^{d} \left[p(\tilde{x}_{ij} \mid \tilde{y}_i = c) \right] p(\tilde{y}_i = c)$$

Naïve Bayes Prediction Phase

- Prediction in a naïve Bayes model:

Given a test example $\tilde{x}_i$ we set prediction $\hat{y}_i$ to the 'c' maximizing $p(\tilde{x}_i \mid \tilde{y}_i = c)$

Under the naive Bayes assumption we can maximize:

$$p(\tilde{y}_i = c \mid \tilde{x}_i) \propto \prod_{j=1}^{d} \left[p(\tilde{x}_{ij} \mid \tilde{y}_i = c) \right] p(\tilde{y}_i = c)$$

Naïve Bayes Prediction Phase

- Prediction in a naïve Bayes model:

Consider $\tilde{x}_i = [1 \ 1]$ in this data set $\longrightarrow$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Naïve Bayes Prediction Phase

- Prediction in a naïve Bayes model:

Consider $\tilde{x}_i = [1\ 1]$ in this data set $\longrightarrow$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$p(\tilde{y}_i = 0 \mid \tilde{x}_i) \propto p(\tilde{x}_{i1} = 1 \mid \tilde{y}_i = 0)\, p(\tilde{x}_{i2} = 1 \mid \tilde{y}_i = 0)\, p(\tilde{y}_i = 0)$$
$$= (1)\,(0.25)\,(0.4) = 0.1$$

Naïve Bayes Prediction Phase

- Prediction in a naïve Bayes model:

Consider $\tilde{x}_i = [1 \ 1]$ in this data set →

$$p(\tilde{y}_i = 0 \mid \tilde{x}_i) \propto p(\tilde{x}_{i1} = 1 \mid \tilde{y}_i = 0)\, p(\tilde{x}_{i2} = 1 \mid \tilde{y}_i = 0)\, p(\tilde{y}_i = 0)$$
$$= (1) \qquad (0.25) \qquad (0.4) = 0.1$$

$$p(\tilde{y}_i = 1 \mid \tilde{x}_i) \propto p(\tilde{x}_{i1} = 1 \mid \tilde{y}_i = 1)\, p(\tilde{x}_{i2} = 1 \mid \tilde{y}_i = 1)\, p(\tilde{y}_i = 1)$$
$$= (0.5) \qquad (0.666\ldots) \qquad (0.6) = 0.2$$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Naïve Bayes Prediction Phase

- Prediction in a naïve Bayes model:

Consider $\tilde{x}_i = [1\ 1]$ in this data set $\longrightarrow$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$p(\tilde{y}_i = 0 \mid \tilde{x}_i) \propto p(\tilde{x}_{i1} = 1 \mid \tilde{y}_i = 0)\, p(\tilde{x}_{i2} = 1 \mid \tilde{y}_i = 0)\, p(\tilde{y}_i = 0)$$
$$= (1) \quad (0.25) \quad (0.4) = 0.1$$

$$p(\tilde{y}_i = 1 \mid \tilde{x}_i) \propto p(\tilde{x}_{i1} = 1 \mid \tilde{y}_i = 1)\, p(\tilde{x}_{i2} = 1 \mid \tilde{y}_i = 1)\, p(\tilde{y}_i = 1)$$
$$= (0.5) \quad (0.666\ldots) \quad (0.6) = 0.2$$

Since $p(\tilde{y}_i = 1 \mid \tilde{x}_i)$ is bigger than $p(\tilde{y}_i = 0 \mid \tilde{x}_i)$, naive Bayes predicts $\hat{y}_i = 1$.

(Don't sum to 1 because we're ignoring $p(\tilde{x}_i)$)

Text Example 1: Language Identification

- Consider data that doesn't look like this:

$$X = \begin{bmatrix} 0.5377 & 0.3188 & 3.5784 \\ 1.8339 & -1.3077 & 2.7694 \\ -2.2588 & -0.4336 & -1.3499 \\ 0.8622 & 0.3426 & 3.0349 \end{bmatrix}, \quad y = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix},$$

- But instead looks like this:

$$X = \begin{bmatrix} \text{Do you want to go for a drink sometime?} \\ \text{J'achète du pain tous les jours.} \\ \text{Fais ce que tu veux.} \\ \text{There are inner products between sentences?} \end{bmatrix}, y = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix}.$$

- How should we represent sentences using features?

A (Bad) Universal Representation

- Treat character in position 'j' of the sentence as a categorical feature.
  - "fais ce que tu veux" => x_i = [f a i s '' c e '' q u e '' t u '' v e u x .]

- "Pad" end of the sentence up to maximum #characters:
  - "fais ce que tu veux" => x_i = [f a i s '' c e '' q u e '' t u '' v e u x . γ γ γ γ γ γ γ γ …]

- Advantage:
  – No information is lost, KNN can eventually solve the problem.
- Disadvantage: throws out everything we know about language.
  – Needs to learn that "veux" starting from any position indicates "French".
    - Doesn't even use that sentences are made of words (this must be learned).
  – High overfitting risk, you will need a lot of examples for this easy task.

Bag of Words Representation

- Bag of words represents sentences/documents by word counts:

> The **International Conference on Machine Learning** (ICML) is the leading international academic conference in machine learning

ICML	International	Conference	Machine	Learning	Leading	Academic
1	2	2	2	2	1	1

- Bag of words loses a ton of information/meaning:
  - But it easily solves language identification problem

Universal Representation vs. Bag of Words

- Why is bag of words better than "string of characters" here?
  - It needs less data because it captures invariances for the task:
    - Most features give strong indication of one language or the other.
    - It doesn't matter *where* the French words appear.
  - It overfits less because it throws away irrelevant information.
    - Exact sequence of words isn't particularly relevant here.

Text Example 2: Word Sense Disambiguation

- Consider the following two sentences:
  - "The cat ran after the mouse."
  - "Move the mouse cursor to the File menu."

- Word sense disambiguation (WSD): classify "meaning" of a word:
  - A surprisingly difficult task.

- You can do ok with bag of words, but it will have problems:
  - "Her mouse clicked on one cat video after another."
  - "We saw the mouse run out from behind the computer."
  - "The mouse was gray." (ambiguous without more context)

Bigrams and Trigrams

- A bigram is an ordered set of two words:
  - Like “computer mouse” or “mouse ran”.
- A trigram is an ordered set of three words:
  - Like “cat and mouse” or “clicked mouse on”.

- These give more context/meaning than bag of words:
  - Includes neighbouring words as well as order of words.
  - Trigrams are widely-used for various language tasks.

- General case is called n-gram.
  - Unfortunately, coupon collecting becomes a problem with larger ‘n’.

Avoiding Underflow

- During the prediction, the probability can underflow:

$$p(y_i = c \mid x_i) \propto \prod_{j=1}^{d} \left[p(x_{ij} \mid y_i = c) \right] p(y_i = c)$$

All these are < 1 so the product gets very small.

- Standard fix is to (equivalently) maximize the logarithm of the probability:

Rember that $\log(ab) = \log(a) + \log(b)$ so $\log(\prod a_i) = \sum_i \log(a_i)$

Since log is monotonic the 'c' maximizing $p(y_i = c \mid x_i)$ also maximizes $\log p(y_i = c \mid x_i)$,

so maximize $\log\left(\prod_{j=1}^{d} \left[p(x_{ij} \mid y_i = c)\right] p(y_i = c)\right) = \sum_{j=1}^{d} \log(p(x_{ij} \mid y_i = c)) + \log(p(y_i = c))$

Handling Data Sparsity

- Do we need to store the full bag of words 0/1 variables?
  - No: only need list of non-zero features for each e-mail.

$	Hi	CPSC	340	Vicodin	Offer	…
1	1	0	0	1	0	…
0	0	0	0	1	1	…
0	1	1	1	0	0	…
1	1	0	0	0	1	…

vs.

Non-Zeroes
{1,2,5,…}
{5,6,…}
{2,3,4,…}
{1,2,6,…}

  - Math/model doesn't change, but more efficient storage.

Less-Naïve Bayes

- Given features {x1,x2,x3,…,xd}, naïve Bayes approximates p(y|x) as:

$$p(y \mid x_1, x_2, \dots, x_d) \propto p(y)\, p(x_1, x_2, \dots, x_d \mid y)$$

product rule applied repeatedly

$$= p(y)\, p(x_1 \mid y)\, p(x_2 \mid x_1, y)\, p(x_3 \mid x_2, x_1, y) \cdots p(x_d \mid x_1, x_2, \dots, x_{d-1}, y)$$

$$\approx p(y)\, p(x_1 \mid y)\, p(x_2 \mid y)\, p(x_3 \mid y) \cdots p(x_d \mid y) \quad \text{(naive Bayes assumption)}$$

- The assumption is very strong, and there are “less naïve” versions:
  - Assume independence of all variables except up to ‘k’ largest ‘j’ where j < i.
    - E.g., naïve Bayes has k=0 and with k=2 we would have:

$$\approx p(y)\, p(x_1 \mid y)\, p(x_2 \mid x_1, y)\, p(x_3 \mid x_2, x_1, y)\, p(x_4 \mid x_3, x_2, y) \cdots p(x_d \mid x_{d-2}, x_{d-1}, y)$$

    - Fewer independence assumptions so more flexible, but hard to estimate for large ‘k’.
  - Another practical variation is “tree-augmented” naïve Bayes.

Gaussian Discriminant Analysis

- Classifiers based on Bayes rule are called generative classifier:
  - They often work well when you have tons of features.
  - But they need to know $p(x_i \mid y_i)$, probability of features given the class.
    - How to "generate" features, based on the class label.

- To fit generative models, usually make BIG assumptions:
  - Naïve Bayes (NB) for discrete x_i:
    - Assume that each variables in x_i is independent of the others in x_i given y_i.

  - Gaussian discriminant analysis (GDA) for continuous x_i.
    - Assume that $p(x_i \mid y_i)$ follows a multivariate normal distribution.
    - If all classes have same covariance, it's called "linear discriminant analysis".

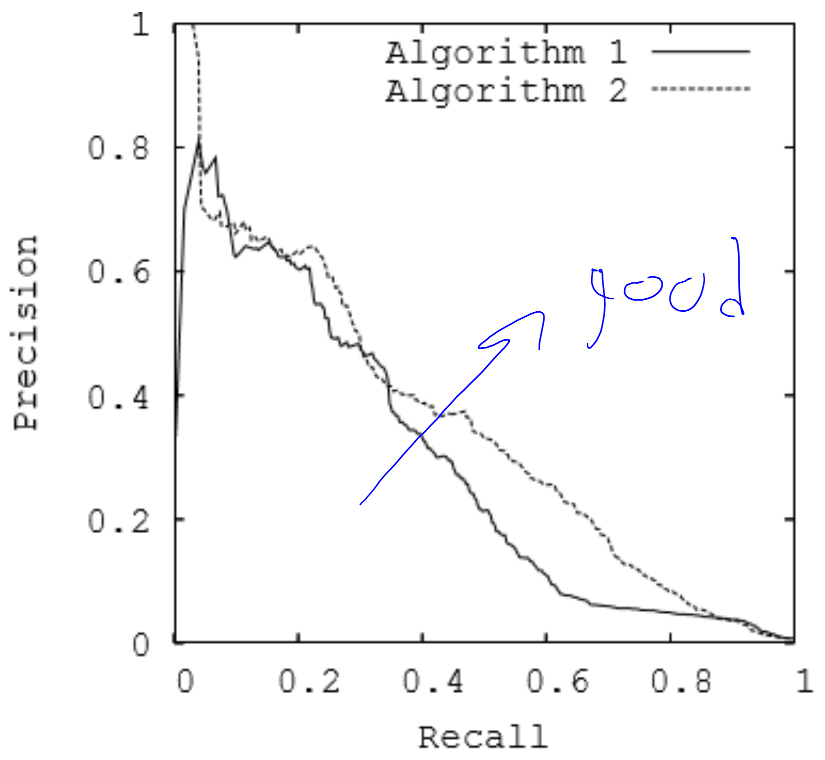
Computing $p(x_i)$ under naïve Bayes

- Generative models don't need $p(x_i)$ to make decisions.
- However, it's easy to calculate under the naïve Bayes assumption:

$$
\begin{aligned}
p(x_i) &= \sum_{c=1}^{K} p(x_i, y=c) && \text{(marginalization rule)}\\
&= \sum_{c=1}^{K} p(x_i \mid y=c)\,p(y=c) && \text{(product rule)}\\
&= \sum_{c=1}^{K} \underbrace{\left[\prod_{j=1}^{d} p(x_{ij} \mid y=c)\right] p(y=c)}_{\text{These are the quantities we compute during training.}} && \text{(naïve Bayes assumption)}
\end{aligned}
$$

Precision-Recall Curve

- Consider the rule $p(y_i = \text{'spam'} \mid x_i) > t$, for threshold ‘t’.
- Precision-recall (PR) curve plots precision vs. recall as ‘t’ varies.



http://pages.cs.wisc.edu/~jdavis/davisgoadrichcamera2.pdf

More on Unbalanced Classes

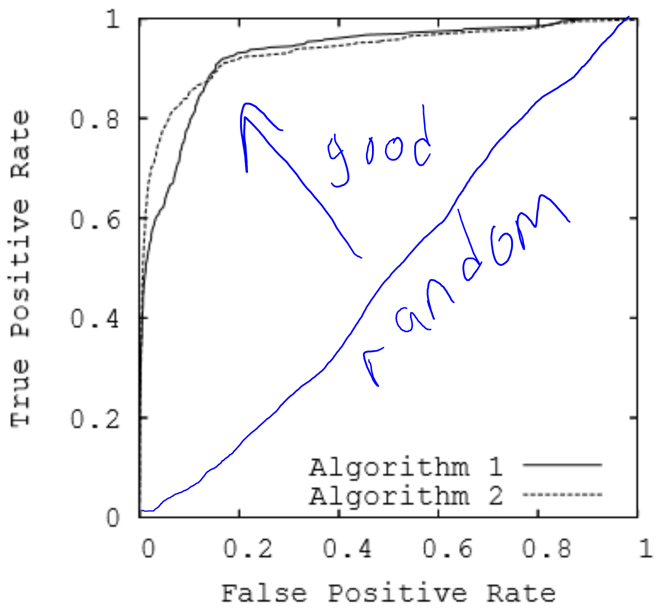
- With unbalanced classes, there are many alternatives to accuracy as a measure of performance:
  - Two common ones are the Jaccard coefficient and the F-score.
  - Jaccard measure: TP/(TP + FP + FN).

- Some machine learning models don't work well with unbalanced data. Some common heuristics to improve performance are:
  - Under-sample the majority class (only take 5% of the spam messages).
    - https://www.jair.org/media/953/live-953-2037-jair.pdf
  - Re-weight the examples in the accuracy measure (multiply training error of getting non-spam messages wrong by 10).
  - Some notes on this issue are here.

ROC Curve

- Receiver operating characteristic (ROC) curve:
  - Plot true positive rate (recall) vs. false positive rate FP/(FP+TN).

    (negative examples classified as positive)



  - Diagonal is random, perfect classifier would be in upper left.
  - Sometimes papers report area under curve (AUC).
    - Reflects performance for different possible thresholds on the probability.

http://pages.cs.wisc.edu/~jdavis/davisgoadrichcamera2.pdf