CSE/STAT 416

Precision/Recall
k-Nearest Neighbors

Pemi Nguyen
Paul G. Allen School of Computer Science & Engineering
University of Washington

May 16, 2021

Pre-Lecture Video

Detecting Spam

Imagine I made a “Dummy Classifier” for detecting spam

The classifier ignores the input, and always predicts spam.

This actually results in 90% accuracy! Why?

- Most emails are spam...

This is called the **majority class classifier.**

A classifier as simple as the majority class classifier can have a high accuracy if there is a **class imbalance**.

A class imbalance is when one class appears much more frequently than another in the dataset

This might suggest that accuracy isn’t enough to tell us if a model is a good model.

Assessing Accuracy

Always digging in and ask critical questions of your accuracy.

Is there a **class imbalance**?

How does it compare to a baseline approach?

- Random guessing
- Majority class
- ...

Most important: **What does my application need?**

- What's good enough for user experience?
- What is the impact of a mistake we make?

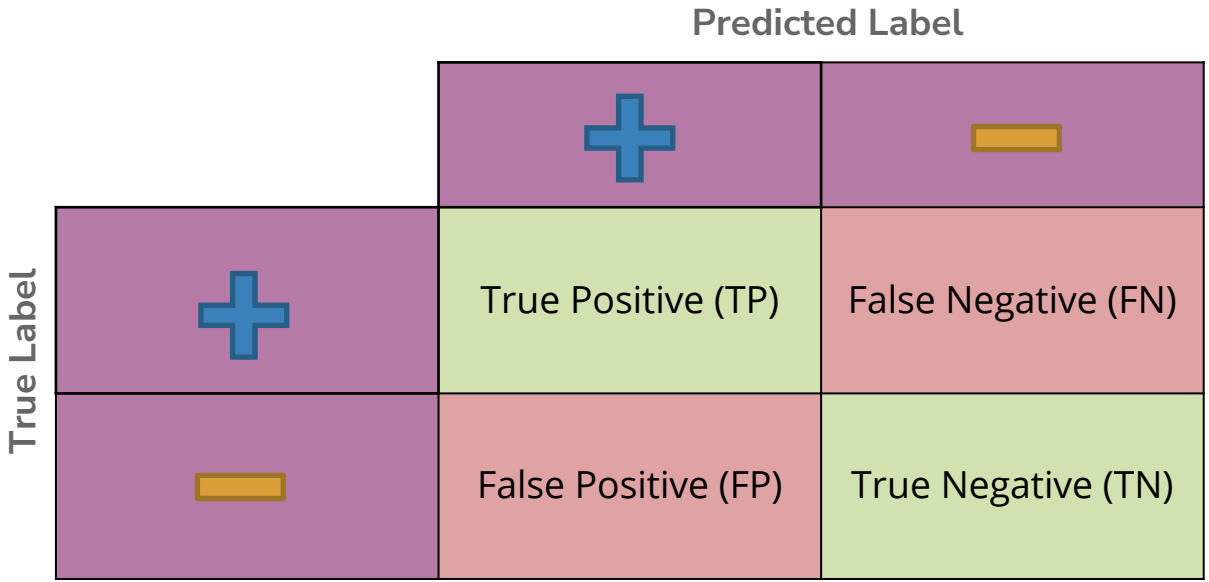
Confusion Matrix

For binary classification, there are only two types of mistakes

$\hat{y} = +1, \quad y = -1$

$\hat{y} = -1, \quad y = +1$

Generally we make a **confusion matrix** to understand mistakes.



Binary Classification Measures

Notation

$$C_{TP} = \#\text{TP}, \quad C_{\text{FP}} = \#\text{FP}, \quad C_{\text{TN}} = \#\text{TN}, \quad C_{\text{FN}} = \#\text{FN}$$

$$N = C_{TP} + C_{FP} + C_{TN} + C_{FN}$$

$$N_P = C_{TP} + C_{FN}, \quad N_N = C_{FP} + C_{TN}$$

Error Rate

$$\frac{C_{FP} + C_{FN}}{N}$$

Accuracy Rate

$$\frac{C_{TP} + C_{TN}}{N}$$

False Positive rate (FPR)

$$\frac{C_{FP}}{N_N}$$

False Negative Rate (FNR)

$$\frac{C_{FN}}{N_P}$$

True Positive Rate or Recall

$$\frac{T_P}{N_P}$$

Precision

$$\frac{T_P}{C_{TP} + C_{FP}}$$

F1-Score

$$2\frac{Precision \cdot Recall}{Precison + Recall}$$

See more!

Change Threshold

What if I never want to make a false positive prediction?

Always predict negative $(\alpha = \infty)$

What if I never want to make a false negative prediction?

Always predicting positive $(\alpha = -\infty)$

One way to control for our application is to change the scoring threshold. (Could also change intercept!)

If $Score(x) > \alpha$:

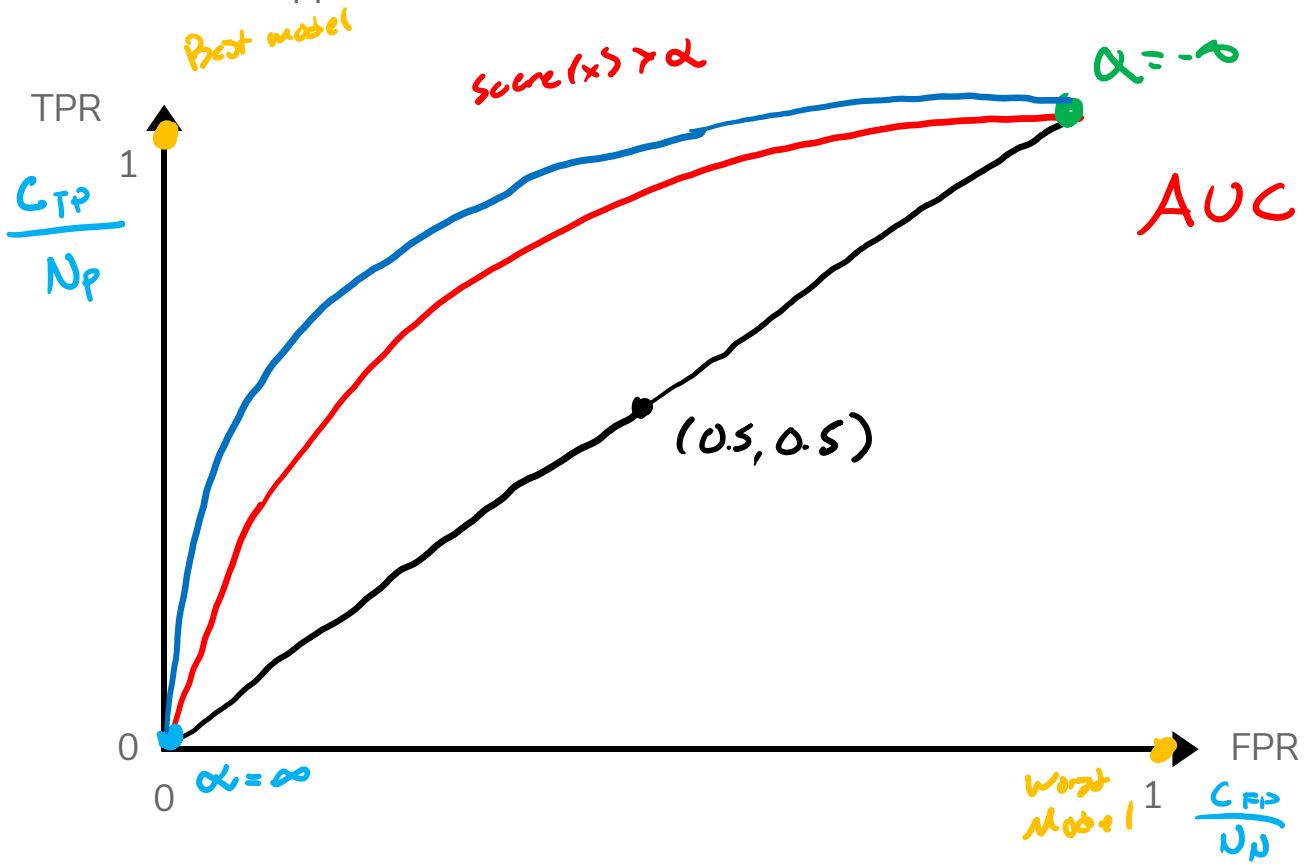
- Predict $\hat{y} = +1$

Else:

- Predict $\hat{y} = -1$

ROC Curve

What happens to our TPR and FPR as we increase the threshold?



Assessing Accuracy

Often with binary classification, we treat the positive label as being the more important of the two. We then often then focus on these metrics:

Precision: Of the ones I predicted positive, how many of them were actually positive?

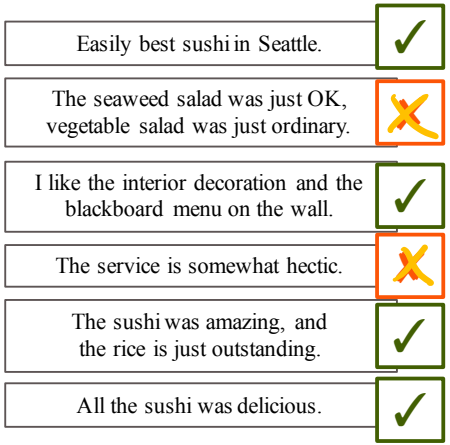
Recall: Of all the things that are truly positive, how many of them did I correctly predict as positive?

Precision

What fraction of the examples I predicted positive were correct?

Sentences predicted to be positive:

$\hat{y}_i = +1$

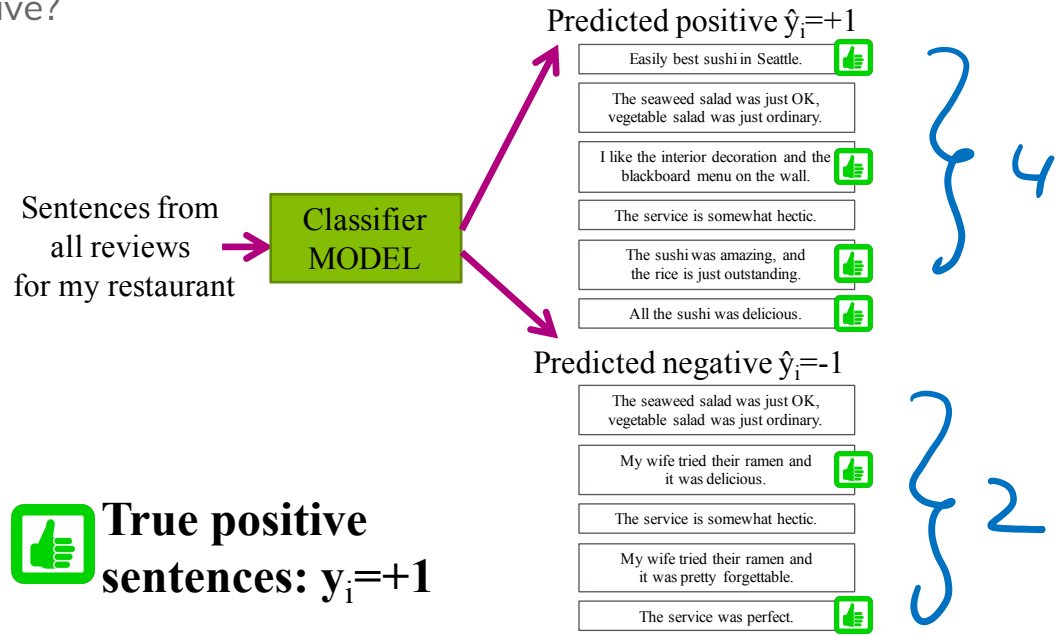


Only 4 out of 6 sentences predicted to be positive are actually positive

$$precision = \frac{C_{TP}}{C_{TP} + C_{FP}} = \frac{4}{4+2} = \frac{2}{3}$$

Recall

Of the truly positive examples, how many were predicted positive?



$$recall = \frac{C_{TP}}{N_P} = \frac{C_{TP}}{C_{TP} + C_{FN}} = \frac{4}{4+2} = \frac{2}{3}$$

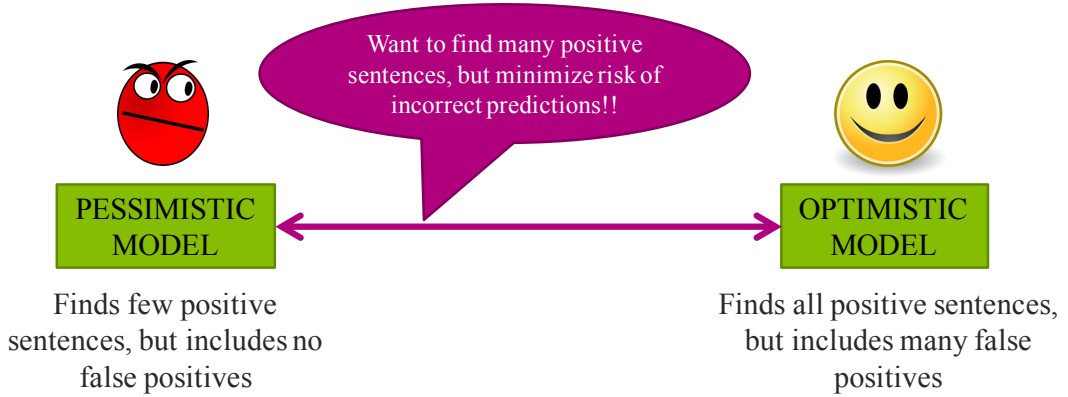
Precision & Recall

An optimistic model will predict almost everything as positive

High recall, low precision

A pessimistic model will predict almost everything as negative

High precision, low recall



Controlling Precision/Recall

Depending on your application, precision or recall might be more important

> Ideally you will have high values for both, but generally increasing recall will decrease precision and vice versa.

For logistic regression, we can control for how optimistic the model is by changing the threshold for positive classification

Before

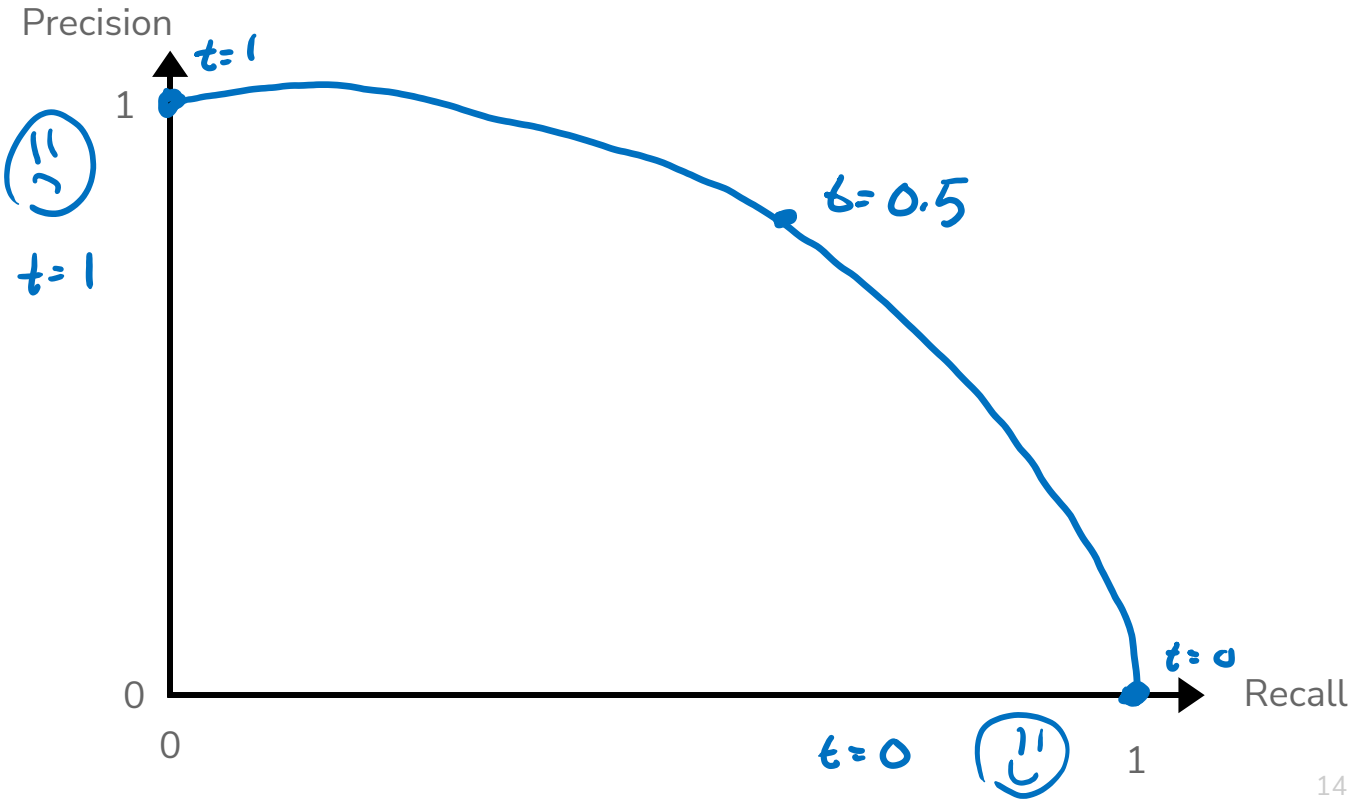
$\hat{y}_i = +1$ if $\hat{P}(y = +1|x_i) > 0.5$ else $\hat{y}_i = -1$

Now

$\hat{y}_i = +1$ if $\hat{P}(y = +1|x_i) > t$ else $\hat{y}_i = -1$

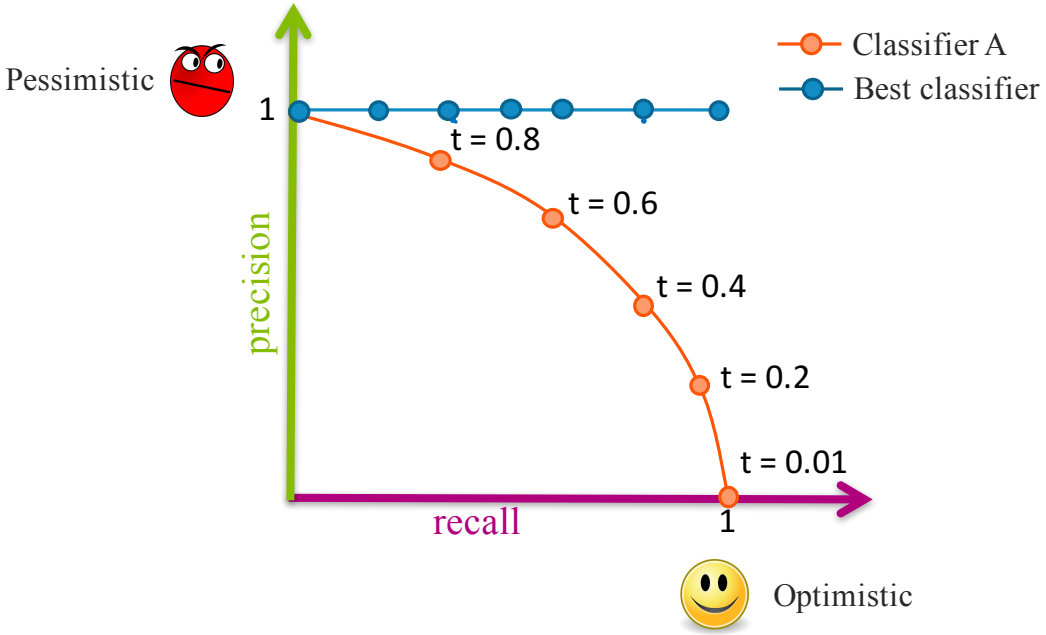
Precision-Recall Curve

$\frac{C_{TP}}{C_{TP}+C_{FP}} = \frac{0}{0} = 1$



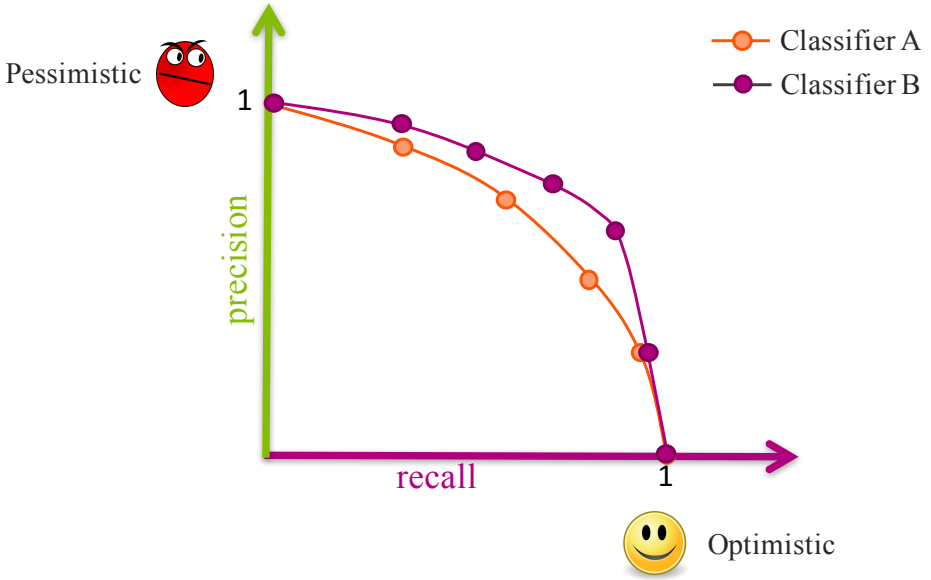
Precision-Recall Curve

Can try every threshold to get a curve like below



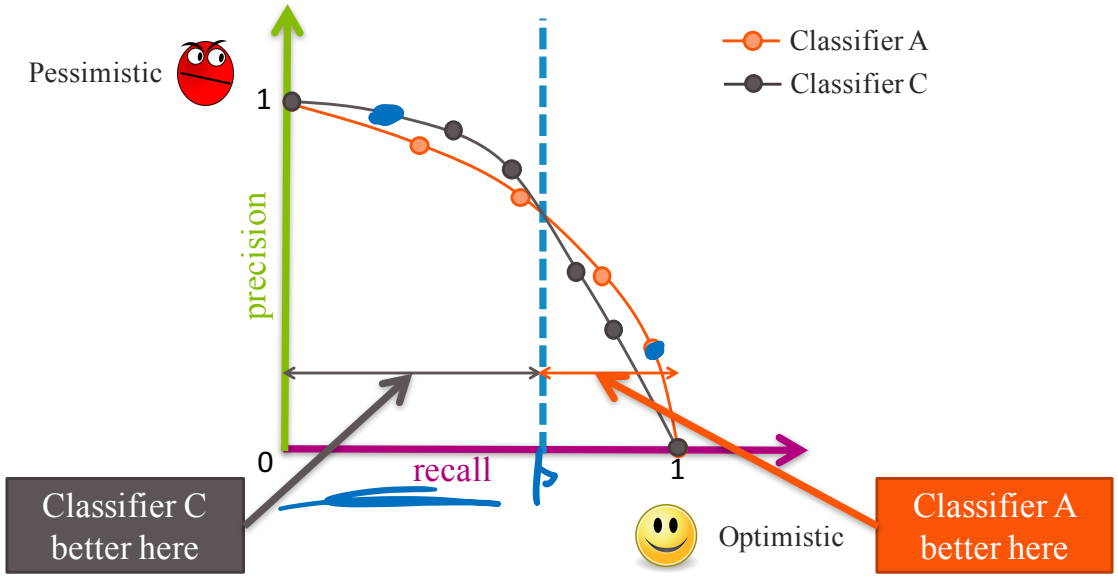
Precision-Recall Curve

Sometimes, Classifier B is strictly better than Classifier A



Precision-Recall Curve

Most times, the classifiers are incomparable



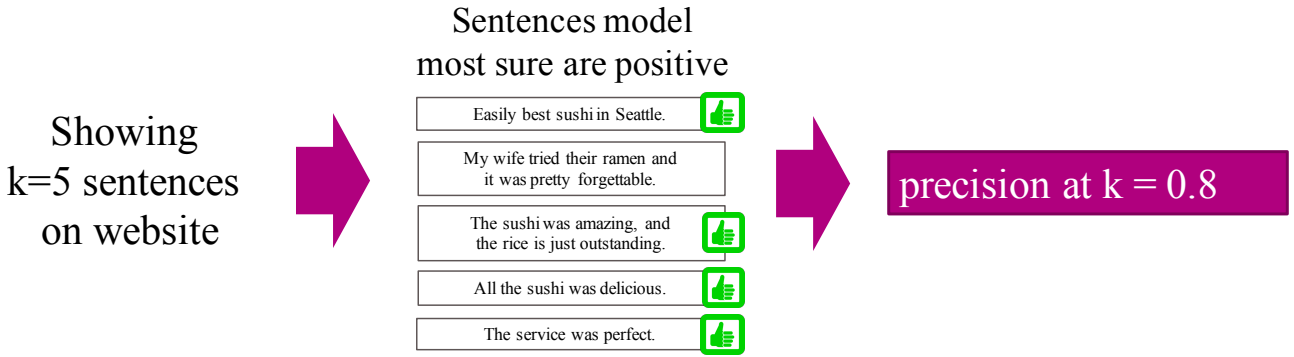
Compare Classifiers

Often come up with a single number to describe it

- F1-score, AUC, etc.
- Remember, what your application needs is most important

Also common to use **precision at k**

- If you show the top **k** most likely positive examples, how many of them are true positives



Class Session

Roadmap

1. Housing Prices - Regression
   - Regression Model
   - Assessing Performance
   - Ridge Regression
   - LASSO
2. Sentiment Analysis – Classification
   - Classification Overview
   - Logistic Regression
   - Bias / Fairness
   - Decision Trees
   - Ensemble Methods
3. Document Retrieval – Clustering and Similarity
   - Precision / Recall
   - k-Nearest Neighbor
   - Kernel Methods
   - Locality Sensitive Hashing
   - Clustering
   - Hierarchical Clustering

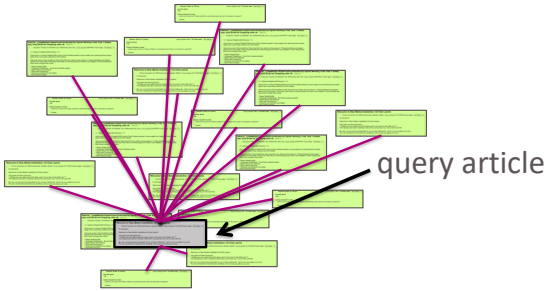
Document Retrieval

Consider you had some time to read a book and wanted to find other books similar to that one.

If we wanted to write an system to recommend books

- How do we measure similarity?
- How do we search over books?
- How do we measure accuracy?

Big Idea: Define an **embedding** and a **similarity metric** for the books, and find the **"nearest neighbor"** to some query book.



Nearest Neighbors

1-Nearest Neighbor

Input

x_q: Query example (e.g. my book)

$x_1, \ldots, x_n$: Corpus of documents (e.g. Amazon books)

Output

The document in corpus that is most similar to x_q

$$x^{NN} = \underset{x_i \in [x_1, \ldots, x_n]}{\arg\min} \; distance(x_q, x_i)$$

It's very critical to properly define how we represent each document x_i and the similarity metric $distance$! Different definitions will lead to very different results.

1-Nearest Neighbor

How long does it take to find the 1-NN? About n operations

Input: x_q

$x^{NN} = \emptyset$

$nn\_dist = \infty$

$for\ x_i \in [x_1, \dots, x_n]$:

    $dist = distance(x_q, x_i)$

    $if\ dist < nn\_dist$:

        $x^{NN} = x_i$

        $nn\_dist = dist$

Output: x^{NN}

k-Nearest Neighbors

Input

x_q: Query example (e.g. my book)

$x_1, \ldots, x_n$: Corpus of documents (e.g. Amazon books)

Output

List of k documents most similar to x_q

Formally

k-Nearest Neighbors

Same idea as 1-NN algorithm, but maintain list of k-NN

Input: x_q

$X^{k-NN} = [x_1, \dots, x_k]$

$nn\_dists = [dist(x_1, x_q), dist(x_2, x_q), \dots, dist(x_k, x_q)]$

$for\ x_i \in [x_{k+1}, \dots, x_n]$:

$dist = distance(x_q, x_i)$

$if\ dist < \max(nn\_dists)$:

$remove\ largest\ dist\ from\ X^{k-NN}\ and\ nn\_dists$

$add\ x_i\ to\ X^{k-NN} and\ distance(x_q, x_i)\ to\ nn\_dists$

Output: X^{k-NN}

k-Nearest Neighbors

Can be used in many circumstances!

Retrieval

Return X^{k-NN}

Regression

$$\hat{y}_i = \frac{1}{k}\sum_{j=1}^{k} x^{NN_j}$$

Classification

$$\hat{y}_i = majority\_class(X^{k-NN})$$

Embeddings

Important Points

While the formalization of these algorithms are fairly tedious, the intuition is fairly simple. Find the 1 or k nearest neighbors to a given document and return those as the answer.

This intuition relies on answering two important questions

- How do we represent the documents x_i?
- How do we measure the distance $distance(x_q, x_i)$?

Document Representation

Like our previous ML algorithms, we will want to make a vector out of the document to represent it as a point in space.

Simplest representation is the **bag-of-words** representation.

- Each document will become a W dimension vector where W is the number of words in the entire corpus of documents
- The value of $x_i[j]$ will be the number of times word j appears in document i.
- This ignores order of words in the document, just the counts.

Bag of Words

Pros

- Very simple to describe
- Very simple to compute

Cons

- Common words like “the” and “a” dominate counts of uncommon words
- Often it’s the uncommon words that uniquely define a doc.

TF-IDF

Goal: Emphasize important words

Appear frequently in the document (common locally)

Term frequency = word counts

Appears rarely in the corpus (rare globally)

Inverse doc freq. = $\log \frac{\#\text{ docs}}{1 + \#\text{ docs using word}}$

tf * idf

Do a pair-wise multiplication to compute the TF-IDF for each word

Words that appear in every document will have a small IDF making the TF-IDF small!

Distance

Euclidian Distance

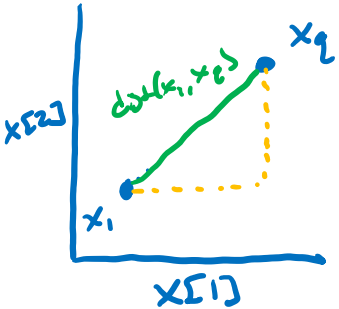
Now we will define what similarity/distance means

Want to define how “close” two vectors are. A smaller value for distance means they are closer, a large value for distance means they are farther away.

The simplest way to define distance between vectors is the **Euclidean distance**

$$distance(x_i, x_q) = \left|\left|x_i - x_q\right|\right|_2$$

$$= \sqrt{\sum_{j=1}^{D} \left(x_i[j] - x_q[j]\right)^2}$$



Manhattan Distance

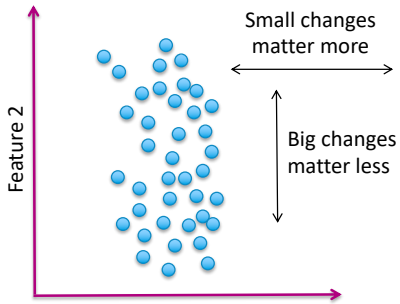
Another common choice of distance is the **Manhattan Distance**

$$distance(x_i, x_q) = \left|\left|x_i - x_q\right|\right|_1$$

$$= \sum_{j=1}^{D} \left|x_i[j] - x_q[j]\right|$$

Weighted Distances

Some features vary more than others or are measured in different units. We can weight different dimensions differently to make the distance metric more reasonable.



Specify weights as a function of feature spread

For feature j: $\frac{1}{\max_i(x_i[j])-\min_i(x_i[j])} = a_j$

Weighted Euclidean distance

$$distance(x_i, x_q) = \sqrt{\sum_{j=1}^{D} a_j^2 \left(x_i[j] - x_q[j]\right)^2}$$

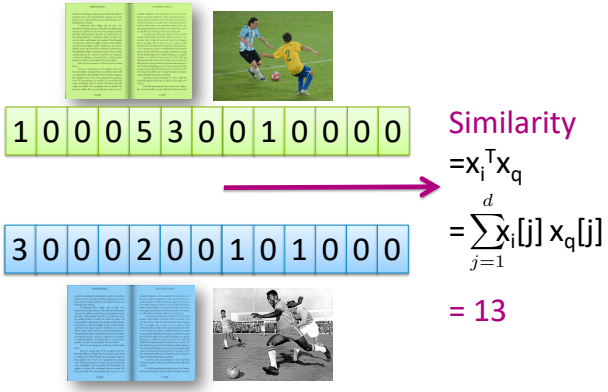
Similarity

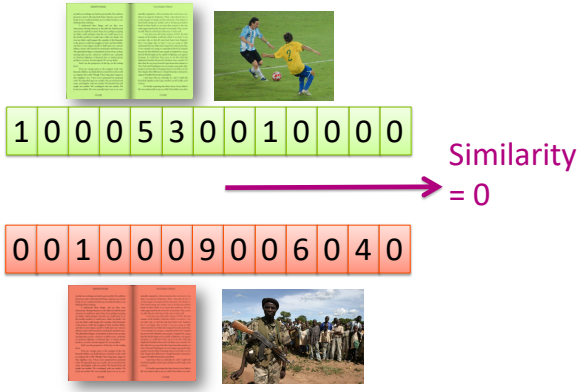
Another natural similarity measure would use

$$x_i^T x_q = \sum_{j=1}^{D} x_i[j] x_q[j]$$

Notice this is a measure of similarity, not distance

This means a bigger number is better





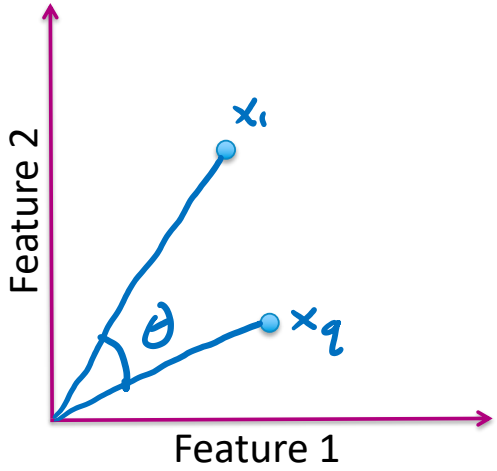
Cosine Similarity

Should we normalize the vectors before finding the similarity?

$$similarity = \frac{x_i^T x_q}{||x_i||_2 \, ||x_q||_2} = \cos(\theta)$$

Note:

- Not a true distance metric
- Efficient for sparse vectors!



Cosine Similarity

In general

$$-1 \leq cosine\ similarity \leq 1$$

For positive features (like TF-IDF)

$$0 \leq cosine\ similarity \leq 1$$

Define

$$distance = 1 - similarity$$

To Normalize or Not To Normalize?

Not normalized

1	0	0	0	5	3	0	0	1	0	0	0	0

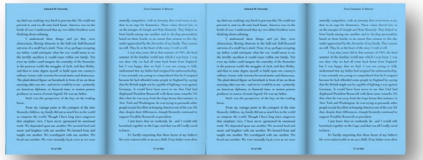
Similarity = 13

3	1	0	0	2	0	0	1	0	1	0	0	0

2	0	0	0	10	6	0	0	2	0	0	0	0

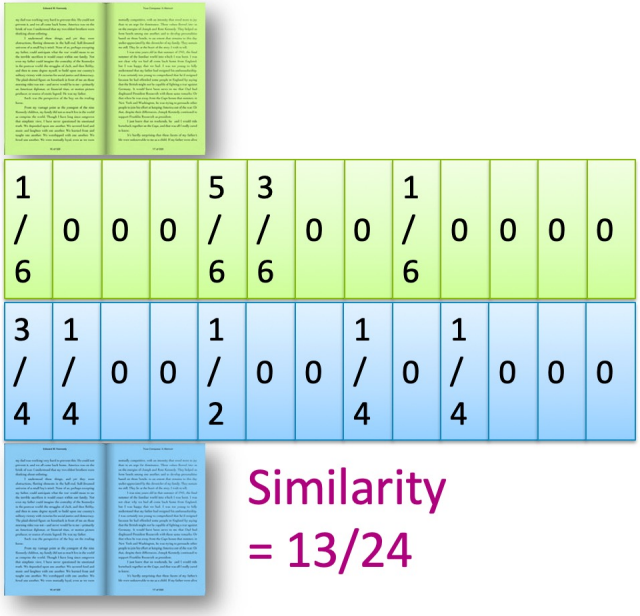
Similarity = 52

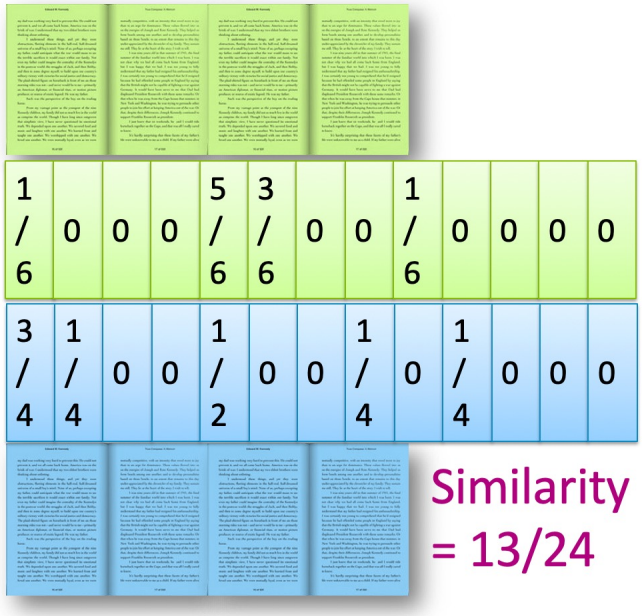
6	2	0	0	4	0	0	2	0	2	0	0	0



To Normalize or Not To Normalize?

Normalized





To Normalize or Not To Normalize?

Normalization is not desired when comparing documents of different sizes since it ignores length.

long document

short tweet

Normalizing can make dissimilar objects appear more similar

long document

long document

Common compromise: Just cap maximum word counts

In practice, can use multiple distance metrics and combine them using some defined weights

Poll Everywhere

Think

3 min

pollev.com/cs416

Not a real Poll Everywhere question, just time to work!

For the given documents, what are their Euclidean Distance and Cosine Similarity?

Assume we are using a bag of words representation

Document 1: “I really like dogs”

Document 2: “dogs are really really awesome”

Steps:

- Write out bag of words vectors
- Compute Euclidean distance
- Compute Cosine similarity

Poll Everywhere

Think

pollev.com/cs416

Doc1 = "I really like dogs" Doc2 = "dogs are really really awesome"

BoW: [I, really, like, dogs, are, awesome]

$x_1 = [1, 1, 1, 1, 0, 0]$ $x_2 = [0, 2, 0, 1, 1, 1]$

Euclidean Distance $= \|x_1 - x_2\|_2$

$dist(x_1, x_2) = \sqrt{(1-0)^2 + (1-2)^2 + (1-0)^2 + (1-1)^2 + (0-1)^2 + (0-1)^2}$

$= \sqrt{5}$

Cosine Distance $= 1 - \frac{x_1^T x_2}{\|x_1\|_2 \|x_2\|_2}$

$dist(x_1, x_2) = 1 - \frac{1\cdot0 + 1\cdot2 + 1\cdot0 + 1\cdot1 + 0\cdot1 + 0\cdot1}{\sqrt{1^2+1^2+1^2+1^2+0^2+0^2}\sqrt{0^2+2^2+0^2+1^2+1^2+1^2}}$

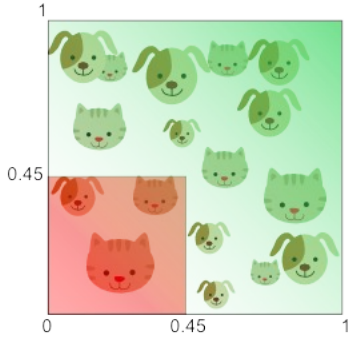
$= 1 - \frac{3}{\sqrt{4}\sqrt{7}} \approx 0.433$

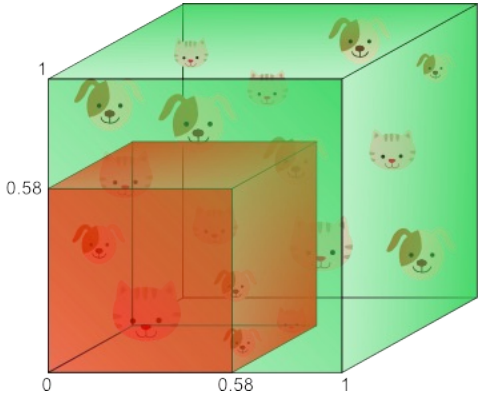
Curse of Dimensionality

High Dimensions

Methods like k-NN and k-means that rely on computing distances start to struggle in high dimensions.

As the number of dimensions grow, the data gets sparser!

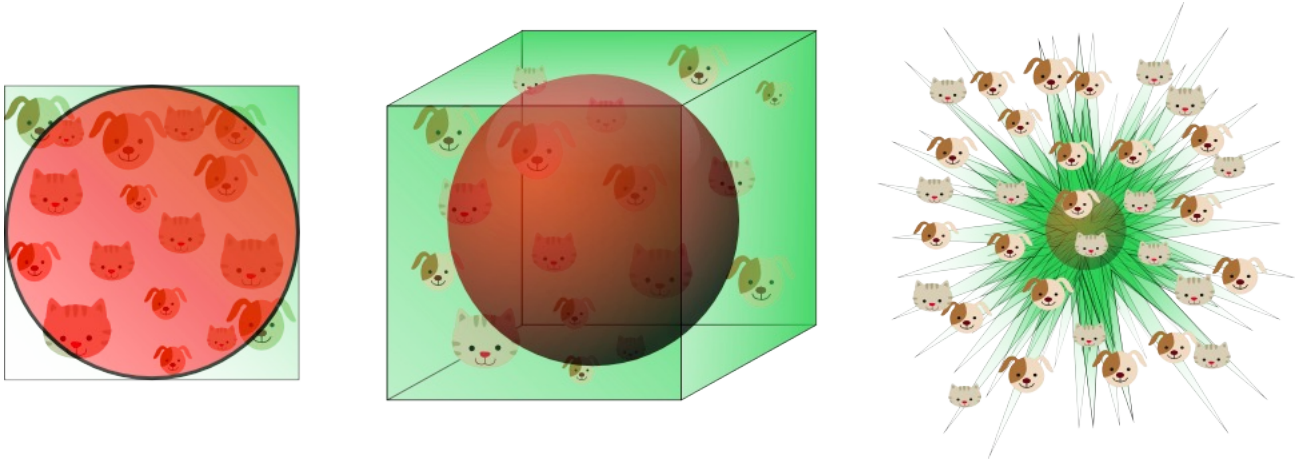




Need more data to make sure you cover all the space in high dim.

Even Weirder

It's believable with more dimensions the data becomes more sparse, but what's even weirder is the sparsity is not uniform!



As D increases, the "mass" of the space goes towards the corners.

- Most of the points aren't in the center.
- Your nearest neighbors start looking like your farthest neighbors!

Practicalities

Have you pay attention to the number of dimensions

Very tricky if $n < D$

Can run into some strange results if D is very large

Later, we will talk about ways of trying to do dimensionality reduction in order to reduce the number of dimensions here.

Recap

Theme: Use nearest neighbors to recommend documents.

Ideas:

Precision and Recall Curves

Implement a nearest neighbor algorithm

Curse of Dimensionality

Compare and contrast different document representations
- Emphasize important words with TF-IDF

Compare and contrast different measurements of similarity
- Euclidean and weighted Euclidean
- Cosine similarity and inner-product similarity