Evaluating classifiers: 
Precision & Recall

Use the sentiment classifier model!

Sentences from all reviews for my restaurant
The seaweed salad was just OK, vegetable salad was just ordinary.
I like the interior decoration and the blackboard menu on the wall.
All the sushi was delicious.
My wife tried their ramen and it was pretty forgettable.
The sushi was amazing, and the rice is just outstanding.
The service is somewhat hectic.
Easily best sushi in Seattle.

Sentences predicted to be positive
\( \hat{y} = +1 \)
- Easily best sushi in Seattle.
- I like the interior decoration and the blackboard menu on the wall.
- All the sushi was delicious.
- The sushi was amazing, and the rice is just outstanding.

Sentences predicted to be negative
\( \hat{y} = -1 \)
- The seaweed salad was just OK, vegetable salad was just ordinary.
- My wife tried their ramen and it was pretty forgettable.
- The service is somewhat hectic.
Automated marketing campaign cares about something else...

Website shows 10 sentences from recent reviews

**PRECISION**
Did I (mistakenly) show a negative sentence???

**RECALL**
Did I not show a (great) positive sentence???

Accuracy doesn’t capture these issues well...

---

**Precision:** Fraction of positive predictions that are actually positive

Sentences predicted to be positive: $\hat{y}_i = +1$

- Easily best sushi in Seattle: ✓
- The seaweed salad was just OK, vegetable salad was just ordinary: X
- I like the interior decoration and the blackboard menu on the wall: ✓
- The service is somewhat hectic: X
- The sushi was amazing, and the rice is just outstanding: ✓
- All the sushi was delicious: ✓

Only 4 out of 6 sentences predicted to be positive are actually positive.
**Precision - Formula**

Fraction of positive predictions that are correct

\[
\text{precision} = \frac{\# \text{ true positives}}{\# \text{ true positives} + \# \text{ false positives}}
\]

- Best possible value : 1.0
- Worst possible value : 0.0

**Recall:** Fraction of positive data predicted to be positive

Found 4 positive sentences

Model could not find 2 sentences that were actually positive

Missed 2 positive sentences
**Recall - Formula**

Fraction of positive data points correctly classified

\[
\text{Recall} = \frac{\# \text{ true positives}}{\# \text{ true positives} + \# \text{ false negatives}}
\]

- Best possible value : 1.0
- Worst possible value : 0.0

**Tradeoff precision and recall**
Optimistic model:
High recall, low precision

Predicted positive $\hat{y}_i = +1$
- Easily best sushi in Seattle.
- The seaweed salad was just OK, vegetable salad was just ordinary.
- I like the interior decoration and the blackboard menu on the wall.
- The service is somewhat hectic.
- The sushi was amazing, and the rice is just outstanding.
- All the sushi was delicious.
- The seaweed salad was just OK, vegetable salad was just ordinary.
- My wife tried their ramen and it was delicious.
- The service was perfect.
- My wife tried their ramen and it was pretty forgettable.

Sentences from all reviews for my restaurant

Predicted negative $\hat{y}_i = -1$
- The service is somewhat hectic.

True positive sentences: $y_i = +1$
- The service is somewhat hectic.
- Easily best sushi in Seattle.
- The seaweed salad was just OK, vegetable salad was just ordinary.
- I like the interior decoration and the blackboard menu on the wall.
- The service is somewhat hectic.
- The sushi was amazing, and the rice is just outstanding.
- All the sushi was delicious.
- The seaweed salad was just OK, vegetable salad was just ordinary.
- My wife tried their ramen and it was delicious.
- The service was perfect.
- My wife tried their ramen and it was pretty forgettable.

Pessimistic model:
High precision, low recall

Predicted positive $\hat{y}_i = +1$
- Easily best sushi in Seattle.
- The seaweed salad was just OK, vegetable salad was just ordinary.
- I like the interior decoration and the blackboard menu on the wall.
- The service is somewhat hectic.
- The sushi was amazing, and the rice is just outstanding.
- All the sushi was delicious.
- The seaweed salad was just OK, vegetable salad was just ordinary.
- My wife tried their ramen and it was delicious.
- The service was perfect.
- My wife tried their ramen and it was pretty forgettable.

Sentences from all reviews for my restaurant

Predicted negative $\hat{y}_i = -1$
- The service is somewhat hectic.

True positive sentences: $y_i = +1$
- The service is somewhat hectic.
- Easily best sushi in Seattle.
- The seaweed salad was just OK, vegetable salad was just ordinary.
- I like the interior decoration and the blackboard menu on the wall.
- The service is somewhat hectic.
- The sushi was amazing, and the rice is just outstanding.
- All the sushi was delicious.
- The seaweed salad was just OK, vegetable salad was just ordinary.
- My wife tried their ramen and it was delicious.
- The service was perfect.
- My wife tried their ramen and it was pretty forgettable.

The service is somewhat hectic.
- Easily best sushi in Seattle.
- The seaweed salad was just OK, vegetable salad was just ordinary.
- I like the interior decoration and the blackboard menu on the wall.
- The service is somewhat hectic.
- The sushi was amazing, and the rice is just outstanding.
- All the sushi was delicious.
- The seaweed salad was just OK, vegetable salad was just ordinary.
- My wife tried their ramen and it was delicious.
- The service was perfect.
- My wife tried their ramen and it was pretty forgettable.

The service is somewhat hectic.
Can we tradeoff precision & recall?

Low precision, high recall
Optimistic Model
Predict almost everything as positive

High precision, low recall
Pessimistic Model
Predict positive only when very sure

Basic classifier

Predict most likely class

If $\hat{P}(y=+1|x_i) > 0.5$:
$\hat{y}_i = +1$
Else:
$\hat{y}_i = -1$
Pessimistic: High precision, low recall

If $\hat{P}(y=+1|x_i) > 0.999$:
\[ \hat{y}_i = +1 \]
Else:
\[ \hat{y}_i = -1 \]

Optimistic: Low precision, high recall

If $\hat{P}(y=+1|x_i) > 0.001$:
\[ \hat{y}_i = +1 \]
Else:
\[ \hat{y}_i = -1 \]
Prediction probability threshold
Probability $t$ above which model predicts true

$$p(y|x, w, x) = \frac{1}{1 + e^{-w^T h(x)}}$$

Set $\hat{y} = +1$ if $\hat{P}(y|x) \geq t$

Example threshold values

$t = 0.99$ (pessimistic)

$t = 0.01$ (optimistic)
Tradeoff precision & recall with threshold

\[ t = 0 \quad \text{to} \quad t = 1 \]

Low precision, high recall

Optimistic Model
Predict almost everything as positive

High precision, low recall

Pessimistic Model
Predict positive only when very sure

Precision-recall curve
The precision-recall curve

What does the perfect algorithm look like?
Which classifier is better? A or B?

Which classifier is better? A or C?

How do we decide???
Compare algorithms

Often, reduce precision-recall to single number to compare algorithms
- F1 measure, area-under-the-curve (AUC), ...

Precision at k

Showing k=5 sentences on website

<table>
<thead>
<tr>
<th>Sentences model most sure are positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easily best sushi in Seattle.</td>
</tr>
<tr>
<td>My wife tried their ramen and it was pretty forgettable.</td>
</tr>
<tr>
<td>The sushi was amazing, and the rice is just outstanding.</td>
</tr>
<tr>
<td>All the sushi was delicious.</td>
</tr>
<tr>
<td>The service was perfect.</td>
</tr>
</tbody>
</table>

precision at k = 0.8

Summary of precision-recall
What you can do now...

- Classification accuracy/error are not always right metrics
- **Precision** captures fraction of positive predictions that are correct
- **Recall** captures fraction of positive data correctly identified by the model
- Trade-off precision & recall by setting probability thresholds
- Plot precision-recall curves.
- Compare models by computing precision at k

Nearest Neighbor Search: Retrieving Documents
Retrieving documents of interest

Document retrieval

• Currently reading article you like
Document retrieval

- Currently reading article you like
- Goal: Want to find similar article
Challenges

• How do we measure similarity?
• How do we search over articles?

Retrieval as $k$-nearest neighbor search
1-NN search for retrieval

Space of all articles, organized by similarity of text

Compute distances to all docs

Space of all articles, organized by similarity of text
Retrieve “nearest neighbor”

Space of all articles, organized by similarity of text

Or set of nearest neighbors

Space of all articles, organized by similarity of text
Nearest neighbor algorithms

1-NN algorithm
1 – Nearest neighbor

- **Input**: Query article $x_q$  
  Corpus of documents $x_1, x_2, ..., x_N$

- **Output**: *Most* similar article  

Formally:

---

1-NN algorithm

Initialize $\text{Dist2NN} = \infty$, $\text{closest} = \emptyset$

For $i=1,2,...,N$

Compute: $\delta = \text{distance}(q, x_i)$

If $\delta < \text{Dist2NN}$

set $\text{closest} = x_i$

set $\text{Dist2NN} = \delta$

Return *most similar document*
k-NN algorithm

k – Nearest neighbor

• **Input:** Query article $x_q$
  Corpus of documents $x_1, x_2, \ldots, x_N$

• **Output:** *List of $k$* similar articles

Formally:
**k-NN algorithm**

Initialize $\text{Dist2kNN} = \text{sort}(\delta_1,...,\delta_k)$

$\text{Dist2kNN} = \text{sort}(\delta_1,...,\delta_k)$

For $i=k+1,...,N$

Compute: $\delta = \text{distance}(x_i, x_q)$

If $\delta < \text{Dist2kNN}[k]$ find $j$ such that $\delta > \text{Dist2kNN}[j-1]$ but $\delta < \text{Dist2kNN}[j]$ remove furthest article and shift queue:

$\text{Dist2kNN}[j+1:k] = \text{Dist2kNN}[j:k-1]$  

$\text{Dist2kNN}[j+1:k] = \text{Dist2kNN}[j:k-1]$  

set $\text{Dist2kNN}[j] = \delta$ and $x_j = x_q$

Return $k$ most similar articles

---

**Critical elements of NN search**

Item (e.g., doc) representation

$x_q \leftarrow \text{query doc}$

Measure of distance between items:

$\delta = \text{distance}(x_i, x_q)$
Document representation

Word count document representation

Bag of words model
- Ignore order of words
- Count # of instances of each word in vocabulary

"Carlos calls the sport futbol. Emily calls the sport soccer."
Issues with word counts – Rare words

Common words in doc: “the”, “player”, “field”, “goal”

Dominate rare words like: “futbol”, “Messi”

TF-IDF document representation

Emphasizes important words

- Appears frequently in document (common locally)

- Appears rarely in corpus (rare globally)
TF-IDF document representation

Emphasizes important words
- Appears frequently in document (common locally)

Term frequency = word counts
- Appears rarely in corpus (rare globally)

Inverse doc freq. = \[ \log \left( \frac{\text{# docs}}{1 + \text{# docs using word}} \right) \]
TF-IDF document representation

Emphasizes important words

- Appears frequently in document (common locally)

**Term frequency** = \[\text{word counts}\]

- Appears rarely in corpus (rare globally)

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

Trade off: local frequency vs. global rarity  
\[\text{tf} \times \text{idf}\]

Distance metrics
Distance metrics: Defining notion of “closest”

In 1D, just Euclidean distance:

$$\text{distance}(x_j, x_q) = |x_j - x_q|$$

In multiple dimensions:
- can define many interesting distance functions
- most straightforwardly, might want to weight different dimensions differently

Weighting different features

Reasons:
- Some features are more relevant than others

# bedrooms
# bathrooms
sq.ft. living
sq.ft. lot
floors
year built
year renovated
waterfront
Weighting different features

Reasons:
- Some features are more relevant than others
- Some features vary more than others

Specify weights as a function of feature spread

For feature $j$: $\frac{1}{\max(x_{ij}) - \min(x_{ij})}$
Scaled Euclidean distance

Formally, this is achieved via

\[
\text{distance}(\mathbf{x}_i, \mathbf{x}_q) = \sqrt{a_1(x_{i[1]} - x_{q[1]})^2 + \ldots + a_d(x_{i[d]} - x_{q[d]})^2}
\]

weight on each feature
(defining relative importance)

Effect of binary weights

Setting weights as 0 or 1 is equivalent to feature selection.

Feature engineering/selection is important, but hard.
Another natural similarity measure

\[
\text{Similarity} = \sum_{j=1}^{d} x_i[j] x_q[j] = 13
\]
Cosine similarity – normalize

\[
\text{Similarity} = \frac{\sum_{j=1}^{d} x_i[j] x_q[j]}{\sqrt{\sum_{j=1}^{d} (x_i[j])^2} \sqrt{\sum_{j=1}^{d} (x_q[j])^2}}
\]

\[
= \frac{x_i^T x_q}{||x_i|| \ ||x_q||} = \cos(\theta)
\]

- Not a proper distance metric
- Efficient to compute for sparse vecs

Normalize

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 5 & 3 & 0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}
\]

\[
\sqrt{(1^2 + 5^2 + 3^2 + 1^2)}
\]

\[
\begin{bmatrix}
1 & / & 0 & 0 & 0 & / & 5 & 3 & / & / & 0 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6
\end{bmatrix}
\]
Cosine similarity

In general, $\angle \text{similarity} \angle$

For positive features (like tf-idf) $\angle \text{similarity} \angle$

Define **distance** = \textbf{1-similarity}

To normalize or not?

<table>
<thead>
<tr>
<th>1 0 0 5 3 0 0 1 0 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{Similarity} = 13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2 0 0 0 1 0 6 0 0 2 0 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{Similarity} = 52</td>
</tr>
</tbody>
</table>
In the normalized case

But not always desired...

Similarity = 13/24

Normalizing can make dissimilar objects appear more similar

Common compromise: Just cap maximum word counts
Other distance metrics

- Mahalanobis
- rank-based
- correlation-based
- Manhattan
- Jaccard
- Hamming
- ...

Combining distance metrics

Example of document features:

1. Text of document
   - Distance metric: Cosine similarity
2. # of reads of doc
   - Distance metric: Euclidean distance

Add together with user-specified weights
Locality sensitive hashing for approximate NN search

Complexity of brute-force search

Given a query point, scan through each point

- $O(N)$ distance computations per 1-NN query!
- $O(N \log k)$ per $k$-NN query!

What if $N$ is huge???
(and many queries)
Moving away from exact NN search

• Approximate neighbor finding...
  – Don’t find exact neighbor, but that’s okay for many applications
    > Out of millions of articles, do we need the closest article or just one that’s pretty similar?
    > Do we even fully trust our measure of similarity???

• Focus on methods that provide good probabilistic guarantees on approximation

Simple “binning” of data into 2 bins

\[ \text{Score}(x) = 1.0 \ \text{#awesome} - 1.5 \ \text{#awful} \]

Like a decision boundary in classification
Simple “binning” of data into 2 bins

<table>
<thead>
<tr>
<th>2D Data</th>
<th>Sign(Score)</th>
<th>Bin index</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 = [0, 5]$</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>$x_2 = [1, 3]$</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>$x_3 = [3, 0]$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Sign(Score($x$)) = -1

Sign(Score($x$)) = +1

Using bins for NN search

<table>
<thead>
<tr>
<th>2D Data</th>
<th>Sign(Score)</th>
<th>Bin index</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 = [0, 5]$</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>$x_2 = [1, 3]$</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>$x_3 = [3, 0]$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Candidate neighbors if Score($x$) < 0

Only search here for queries with Score($x$) < 0

Query point $x$

Only search here for queries with Score($x$) > 0
Using bins for NN search

Bin index

<table>
<thead>
<tr>
<th>2D Data</th>
<th>Sign(Score)</th>
<th>Bin index</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 = [0, 5] )</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>( x_2 = [1, 3] )</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>( x_3 = [3, 0] )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

List containing indices of datapoints:

- \{1, 2, 4, 7, ...\}
- \{3, 5, 6, 8, ...\}

candidate neighbors if \( \text{Score}(x) < 0 \)

search for NN amongst this set

Provides approximate NN

Nearest neighbor to query point found? NO
Three potential issues with simple approach

1. **Challenging to find good line**
2. **Poor quality solution:**
   - Points close together get split into separate bins
3. **Large computational cost:**
   - Bins might contain many points, so still searching over large set for each NN query
How to define the line?

Crazy idea:
Define line randomly!

How bad can a random line be?

Goal: If $x, y$ are close (according to cosine similarity), want binned values to be the same.
How bad can a random line be?

**Goal:** If \( x, y \) are close (according to cosine similarity), want binned values to be the same.

- **Both points in bin 1**
- **One point in bin 0 and other in bin 1**
How bad can a random line be?

**Goal:** If $x, y$ are close (according to cosine similarity), want binned values to be the same.

- **Bins are the same**
- **Bins are different**

If $\theta_{xy}$ is small ($x, y$ close), unlikely to be placed into separate bins.

Three potential issues with simple approach

1. **Challenging to find good line**
2. **Poor quality solution:**
   - Points close together get split into separate bins
3. **Large computational cost:**
   - Bins might contain many points, so still searching over large set for each NN query

<table>
<thead>
<tr>
<th>Bin</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>List containing indices of datapoints:</td>
<td>{1,2,4,7,...}</td>
<td>{3,5,6,8,...}</td>
</tr>
</tbody>
</table>
Improving efficiency: Reducing # points examined per query

Reducing search cost through more bins

Bin index: [0 0 0]
Bin index: [0 1 0]
Bin index: [1 1 0]
Bin index: [1 1 1]
Using score for NN search

2D Data

<table>
<thead>
<tr>
<th></th>
<th>Sign</th>
<th>Bin 1 index</th>
<th>Sign</th>
<th>Bin 2 index</th>
<th>Sign</th>
<th>Bin 3 index</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(Score)</td>
<td></td>
<td>(Score)</td>
<td></td>
<td>(Score)</td>
<td></td>
</tr>
<tr>
<td>$x_1 = [0, 5]$</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>$x_2 = [1, 3]$</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>$x_3 = [3, 0]$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Data indices:

- {1,2} -- {4,8,11} -- {7,9,10} -- {3,5,6}

search for NN amongst this set

Improving search quality by searching neighboring bins

<table>
<thead>
<tr>
<th></th>
<th>[0 0 0] = 0</th>
<th>[0 0 1] = 1</th>
<th>[0 1 0] = 2</th>
<th>[0 1 1] = 3</th>
<th>[1 0 0] = 4</th>
<th>[1 0 1] = 5</th>
<th>[1 1 0] = 6</th>
<th>[1 1 1] = 7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data indices:</td>
<td>{1,2}</td>
<td>--</td>
<td>{4,8,11}</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>(7,9,10)</td>
<td>{3,5,6}</td>
</tr>
</tbody>
</table>

Query point here, but is NN?

Not necessarily

Even worse than before...Each line can split pts.
Sacrificing accuracy for speed
Improving search quality by searching neighboring bins

<table>
<thead>
<tr>
<th>Bin</th>
<th>[0 0 0] = 0</th>
<th>[0 0 1] = 1</th>
<th>[0 1 0] = 2</th>
<th>[0 1 1] = 3</th>
<th>[1 0 0] = 4</th>
<th>[1 0 1] = 5</th>
<th>[1 1 0] = 6</th>
<th>[1 1 1] = 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data indices:</td>
<td>{1,2}</td>
<td>--</td>
<td>{4,8,11}</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>{7,9,10}</td>
<td>{3,5,6}</td>
</tr>
</tbody>
</table>

Next closest bins (flip 1 bit)

Further bin (flip 2 bits)
Improving search quality by searching neighboring bins

Quality of retrieved NN can only improve with searching more bins

**Algorithm:**
Continue searching until computational budget is reached or quality of NN good enough

**LSH recap**
- Draw $h$ random lines
- Compute “score” for each point under each line and translate to binary index
- Use $h$-bit binary vector per data point as bin index
- Create hash table
- For each query point $x$, search $\text{bin}(x)$, then neighboring bins until time limit
Moving to higher dimensions $d$

Draw random planes

$$\text{Score}(x) = v_1 \#\text{awesome} + v_2 \#\text{awful} + v_3 \#\text{great}$$
Cost of binning points in $d$-dim

$$\text{Score}(x) = v_1 \#\text{awesome} + v_2 \#\text{awful} + v_3 \#\text{great}$$

Per data point, need $d$ multiplies to determine bin index per plane

One-time cost offset if many queries of fixed dataset

Summary for retrieval using nearest neighbors and locality sensitive hashing
What you can do now...

- Implement nearest neighbor search for retrieval tasks
- Contrast document representations (e.g., raw word counts, tf-idf,...)
  - Emphasize important words using tf-idf
- Contrast methods for measuring similarity between two documents
  - Euclidean vs. weighted Euclidean
  - Cosine similarity vs. similarity via unnormalized inner product
- Describe complexity of brute force search
- Implement LSH for approximate nearest neighbor search