Machine Learning for IR

31 Dec, 2024

Outline

Machine Learning for IR [5 Hours]

6.1. Supervised learning for ranking (Learning to Rank)

6.2. Feature engineering for IR

6.3. Evaluation of machine learning models in IR

Ad Hoc Retrieval and Standing Queries

- **Ad Hoc Retrieval**:
  - This involves addressing transient (temporary) information needs through queries.
  - Example: Searching "latest smartphone trends" on a search engine when you need immediate information.

- **Ongoing Information Needs**:
  - Some users have persistent needs, such as tracking developments in a specific topic over time (e.g., multicore computer chips).
  - For such needs, ad hoc retrieval is inefficient since users would need to repeatedly run the same or similar queries manually.
- **Standing Queries**:

  - A **standing query** is a predefined query that is automatically executed periodically as new documents are added to the collection.
  - Example: A daily search for "multicore AND computer AND chip" in news articles.
  - These queries save effort by automating repetitive searches for ongoing information needs.

Challenges in Standing Queries

- **Vocabulary Mismatch**:
  - Standing queries may miss relevant documents if the vocabulary in the documents differs from the query terms.
    - Example: A query for "multicore AND chip" might miss documents that mention "multicore processors."

- **Solution: Refinement**:
  - To improve recall (finding all relevant documents), standing queries need refinement over time.
    - Example: Expanding the query to "(multicore OR multi-core) AND (chip OR processor OR microprocessor)" using Boolean logic and stemming.

Standing Queries as Classification

- **Classification Problem**:
  - Classification involves assigning objects (e.g., documents) to one or more predefined categories (or classes).

  - Example:
    - Classes: "Documents about multicore chips" and "Documents not about multicore chips."
    - Standing queries act as classifiers by dividing new documents into these two classes.

- This process is also called **routing** or **filtering** in IR.

Applications of Classification

Preprocessing in IR:

- **Document Encoding Detection**: Identifying if a document is in ASCII, UTF-8, etc.
- **Word Segmentation**: Determining if whitespace marks a word boundary.
- **Language Identification**: Detecting a document's language for proper processing.

Spam Detection:

- Classify and exclude spam pages or emails from search results or inboxes.

Content Filtering:

- Detect explicit content for filtering based on user settings (e.g., SafeSearch).

...

Sentiment Analysis:

- Classify reviews or comments as positive or negative.
- **Example**: A user reads negative reviews of a product to avoid undesirable features.

Email Sorting:

- Automatically sort emails into folders like "Bills," "Announcements," or "Spam."
- Helps users manage large inboxes efficiently.

Vertical Search Engines:

- Specialized search engines focus on a specific topic.
    - **Example**: A vertical search for "computer science" in China provides high-quality, relevant results about Chinese computer science departments, avoiding irrelevant results (e.g., fine china or porcelain).

Introduction

The most common use of machine learning in Information retrieval is Text Classification.

- Predict category and label for a string of text.
  - Is this email spam or not?
  - Does this blogger like or hate our product?
  - Will the Apple stock rise or fall after this news story?
- goal is to determine the class (or classes) a given document belongs to, based on its content.

...

Formal Definition

Let us formally define the text classification problem:

- **Input**:
  - A set of documents $D=\{d1,d2,\ldots,dn\}$.
  - A predefined set of classes $C=\{c1,c2,\ldots,cm\}$.
  - **Output**:
  - A function $f:D\rightarrow C$ that maps each document $d \in D$ to one or more classes $c \in C$.

Document Representation

Before applying a classification algorithm, textual data needs to be represented in a numerical form.

This is typically done using the **vector space model (VSM)**.

1. **Feature Space**:
   - Each document d is represented as a vector in a high-dimensional space:
     d=(w1,w2,…,wt) where ***wi*** is the weight of the ***i-th*** term in the document.
2. **Term Weighting**:
   - Common term-weighting schemes include:
     - **Binary Weighting**: w_i=1 if the term is present; otherwise wi=0.
     - **Term Frequency (TF)**: w_i is the count of term ***i*** in the document.
     - **TF-IDF**: Combines term frequency with inverse document frequency to balance the importance of terms:

$$w_i = \mathrm{TF}(t_i, d) \cdot \log\left(\frac{|D|}{\mathrm{DF}(t_i)}\right)$$

- $\mathrm{TF}(t_i, d)$: Frequency of term t_i in document d.
- $\mathrm{DF}(t_i)$: Number of documents containing t_i.
- $|D|$: Total number of documents.

Types of Classification Tasks

1. **Binary Classification**:
   - Classify a document into one of two classes.
   - Example: Spam detection (Spam or Not Spam).

2. **Multiclass Classification**:
   - Assign a document to one of multiple classes.
   - Example: News classification (Politics, Sports, Technology).

3. **Multilabel Classification**:
   - Assign a document to multiple classes.
   - Example: A news article tagged as Politics and Economy.

Probabilistic Classification

One common approach to text classification is **probabilistic modeling**,

- where we assign a document ***d*** to the class ***c*** that maximizes the probability P(c | d).

Bayes' Theorem:

- Using Bayes' theorem, $P(c|d)$ can be expressed as:

$$P(c|d) = \frac{P(d|c) \cdot P(c)}{P(d)}$$

- Since $P(d)$ is constant for all classes, we simplify to:

$$P(c|d) \propto P(d|c) \cdot P(c)$$

Class Prior ($P(c)$):

- The probability of a class c before observing any document.

Likelihood ($P(d|c)$):

- The probability of observing the document d given that it belongs to class c.

The **Naive Bayes classifier** simplifies $P(d|c)$ by assuming that terms in a document are conditionally independent given the class c.

1. **Conditional Independence**:
   - For a document d represented as terms $t_1, t_2, \ldots, t_n$:

$$P(d|c) = \prod_{i=1}^{n} P(t_i|c)$$

2. **Naive Bayes Classification Rule**:
   - Assign d to the class c^* that maximizes $P(c|d)$:

$$c^* = \arg\max_{c \in C} P(c) \prod_{i=1}^{n} P(t_i|c)$$

...

Text Classification: Converts textual documents into mathematical representations for computational processing.

Efficiency: Uses algorithms to systematically and efficiently assign categories to documents.

Evolution: Transitioned from manual classification methods to machine learning-driven approaches.

Scalability: Machine learning enables handling large volumes of data, overcoming the limitations of manual efforts.

Adaptability: Learns and adjusts to new data, improving performance over time.

Significance: A foundational component in modern information retrieval systems for organizing and retrieving information effectively.

Supervised learning for ranking (Learning to Rank)

- Learning to Rank (LTR) refers to training models to rank items/documents based on their relevance to a query.
    - Relevance as a supervised signal.
    - Input: Queries, documents, and relevance labels.
    - Output: Ordered list of documents.

The scoring model is a Machine Learning model that learns to predict a score s given an input x = (q, d) during a training phase where some sort of ranking loss is minimized.

Machine Learning Models for Learning to Rank

To build a Machine Learning model for ranking, we need to define **inputs**, **outputs** and **loss function**.

- **Input** – For a query ***q*** we have ***n*** documents $\boldsymbol{D = \{d_1, \ldots, d_n\}}$ to be ranked by relevance. The elements $\boldsymbol{x_i = (q, d_i)}$ are the inputs to our model.
- **Output** – For a query-document input $x_i = (q, d_i)$, we assume there exists a true **relevance score** $\boldsymbol{y_i}$. Our model outputs a **predicted score** $\boldsymbol{s_i = f(x_i)}$.

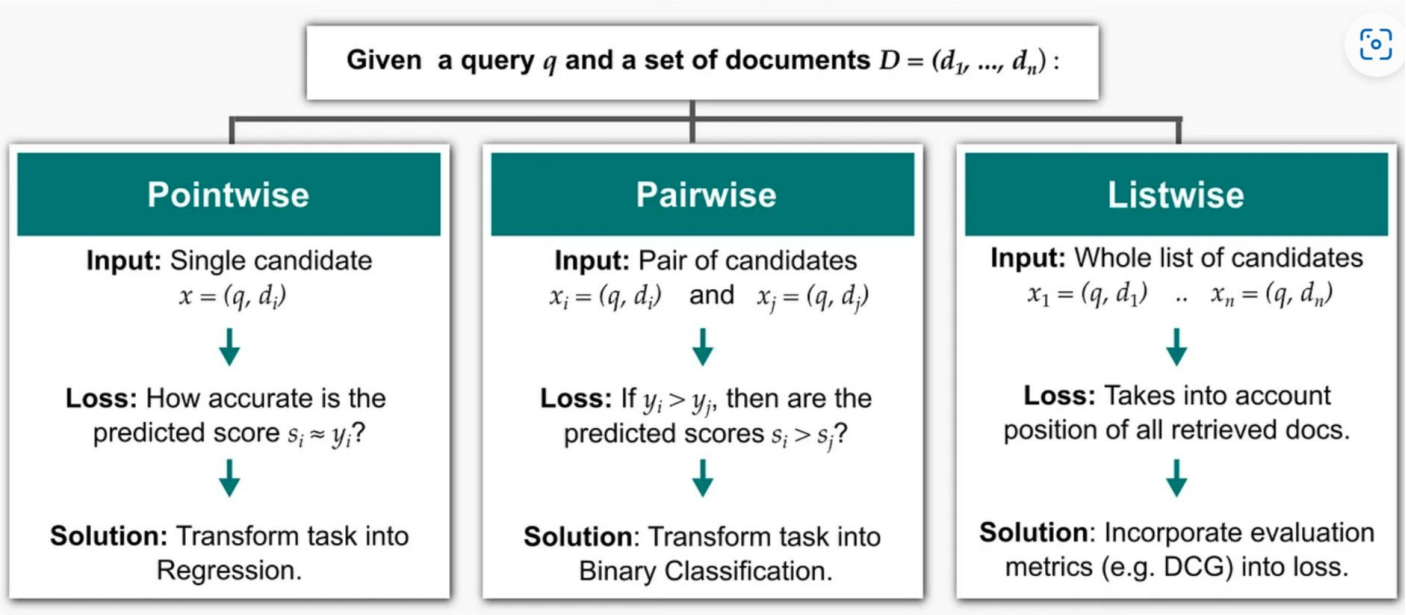
Loss Function

- Ranking models aim to minimize a **loss function** that measures the discrepancy between the predicted ranking and the ground truth.
- Example: **Cross-Entropy Loss** for pairwise ranking (will discuss later).

...

The choice of the **loss function** is the distinctive element for Learning to Rank models.

In general, we have **3 approaches**, depending on how the loss is computed.



Pointwise Methods

- The total loss is computed as the sum of loss terms defined on **each document d_i** (hence ***pointwise***) as the distance between the predicted score s_i and the ground truth y_i, for i=1…n. By doing this,
  - we transform our task into a **regression problem,** where we train a model to predict *y.*
- The pointwise approach is the simplest to implement, and it was the first one to be proposed for Learning to Rank tasks.
- The loss directly measures the distance between ground true score y_i and predicted s_i so we treat this task by effectively solving a regression problem.
  - As an example, **Subset Ranking** uses a Mean Square Error (MSE) loss.

$$L(\underline{s}, \underline{y}) = \sum_{i=1}^{n} (s_i - y_i)^2$$

...

Issue with Pointwise Models

- **Requirement for True Relevance Scores**: Pointwise models require labeled training data with explicit relevance scores (e.g., how relevant each document is on a numerical scale).

- **Limited Training Data**: In practice, explicit scores are often unavailable. Instead:
  - Only partial information is available, such as user clicks.

  - For example, we only know which document in a list of documents was chosen by a user (and therefore is *more relevant*), but we don't know exactly *how relevant* is any of these documents!

Pairwise Methods

- Pairwise models address the lack of explicit scores by predicting which document is more relevant between a pair.
- they work with **relative preference**: given two documents, we want to predict if the first is more relevant than the second.
- This way we solve a **binary classification task** where we only need the ground truth y_{ij} = (1 if $y_i > y_j$, 0 otherwise) and
  - we map from the model outputs to probabilities using a logistic function:
    - $s_{ij} = \sigma(s_i - s_j)$.
    - This approach was first used by **RankNet**, which used a Binary Cross Entropy (BCE) loss.

$$L(\underline{s}, \underline{y}) = -\sum_{i,j=1}^{n} y_{ij} \log(s_{ij}) + (1 - y_{ij}) \log(1 - s_{ij})$$

BCE loss for pairwise methods as in RankNet.

...

- RankNet is an improvement over pointwise methods, but all documents are still given the same importance during training,
  - while we would want to give **more importance to documents in higher ranks** (as the DCG metric does with the discount terms).

- Unfortunately, rank information is available only after sorting, and sorting is non differentiable.

- However, to run Gradient Descent optimization we don't need a loss function,
- we only need its gradient!
  - **LambdaRank** defines the gradients of an implicit loss function so that documents with high rank have much bigger gradients:

$$\lambda_j := \frac{\partial L}{\partial s_j} = \frac{1}{G_{max}} \sum_{i \neq j} \sigma(s_i - s_j) |G_i - G_j| |D_i - D_j|$$

Gradients of an implicit loss function as in LambdaRank.

Listwise Methods

- Optimizes the ranking of the entire list of documents.

- Unlike pointwise and pairwise methods, which convert the ranking problem into a classification or regression task, **listwise methods** directly optimize ranking metrics like **NDCG** (Normalized Discounted Cumulative Gain).

- Maximizing the ranking metric directly should yield the best results, as the entire ranking information is used in training.

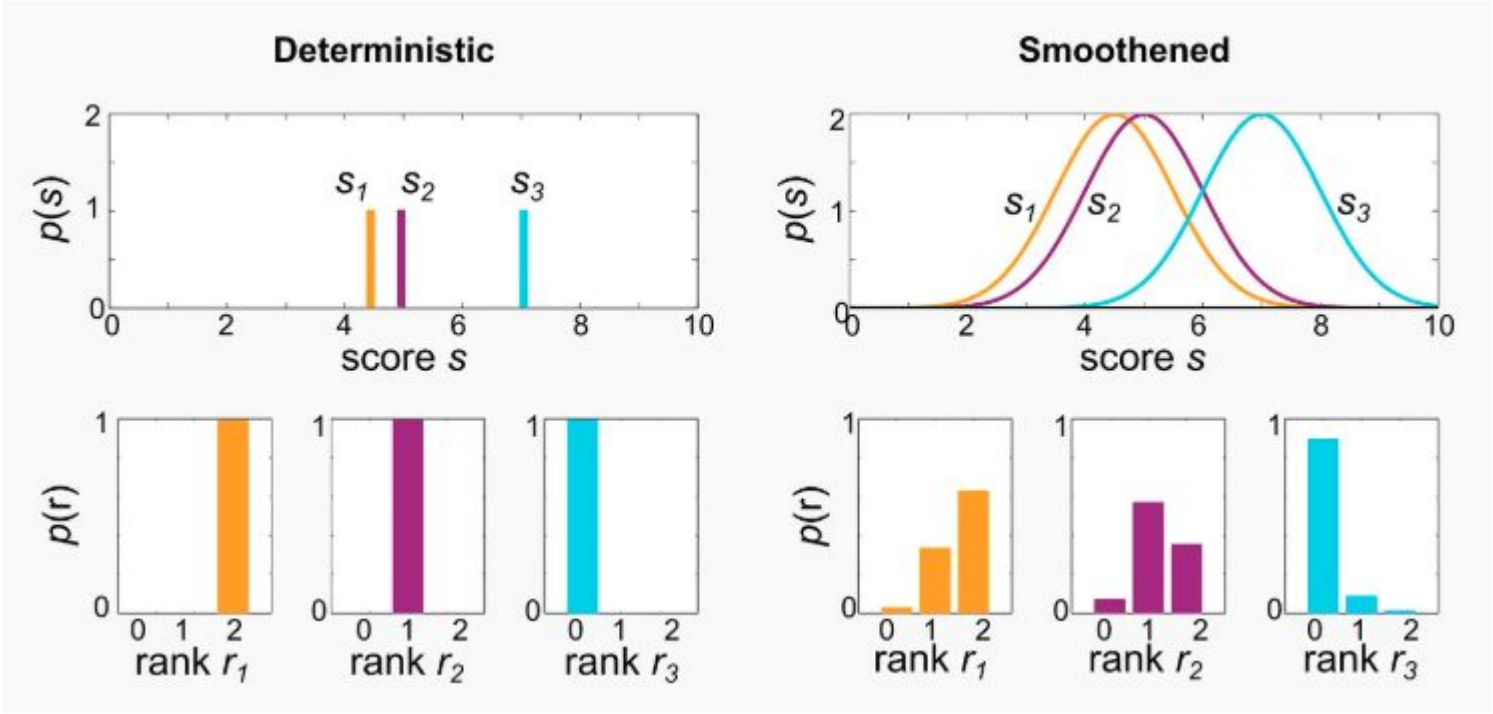
- Ranking metrics (like NDCG) require the ranks of documents to compute the discounted gains,
  - but sorting is non-differentiable, making it difficult to use traditional gradient-based optimization methods.

...

- A **first approach** is to use an iterative method where **ranking metrics are used to re-weight** instances at each iteration.
  - This is the approach used by **LambdaRank** and **LambdaMART**, which are indeed between the pairwise and the listwise approach.

- A **second approach** is to approximate the objective to make it differentiable, which is the idea behind **SoftRank**.
  - Instead of predicting a deterministic score $s = f(x)$, we predict a **smoothened probabilistic score**
    - $s \sim \mathcal{N}(f(x), \sigma^2)$.
  - The **ranks *k*** are non-continuous functions of **predicted scores *s***, but
  - With the help of smoothening, we can compute **probability distributions for the ranks** of each document.
  - Finally, we optimize **SoftNDCG**, the expected NDCG over this rank distribution, which is a smooth function.

...


Deterministic
Smoothened
p(s)
score s
s_1
s_2
s_3
p(r)
rank r_1
rank r_2
rank r_3

. . .

Comparison:

- **Pointwise**: Simple but ignores relative order.
- **Pairwise**: Directly models relative order but does not optimize for ranking metrics.
- **Listwise**: Directly optimizes for ranking metrics but computationally expensive.

Challenges and Limitations

- **Feature Representation**: Designing effective features is critical.
- **Data Sparsity**: Limited labeled data for supervised training.
- **Computational Cost**: Listwise approaches can be expensive.

Feature	Descriptions
1	BM25
2	document length (dl) of body
3	dl of anchor
4	dl of title
5	dl of URL
6	HITS authority
7	HITS hub
8	HostRank (SIGIR feature)
9	Inverse document frequency (idf) of body
10	idf of anchor
11	idf of title
12	idf of URL
13	Sitemap based feature propagation (SIGIR feature)
14	PageRank
15	LMIR.ABS of anchor
16	BM25 of anchor
17	LMIR.DIR of anchor
18	LMIR.JM of anchor
19	LMIR.ABS of extracted title (SIGIR feature)
20	BM25 of extracted title (SIGIR feature)
21	LMIR.DIR of extracted title (SIGIR feature)
22	LMIR.JM of extracted title (SIGIR feature)
23	LMIR.ABS of title

Feature	Descriptions
24	BM25 of title
25	LMIR.DIR of title
26	LMIR.JM of title
27	Sitemap based feature propagation (SIGIR feature)
28	tf of body
29	tf of anchor
30	tf of title
31	tf of URL
32	tf*idf of body
33	tf*idf of anchor
34	tf*idf of title
35	tf*idf of URL
36	Topical PageRank (SIGIR feature)
37	Topical HITS authority (SIGIR feature)
38	Topical HITS hub (SIGIR feature)
39	Hyperlink base score propagation: weighted in-link (SIGIR feature)
40	Hyperlink base score propagation: weighted out-link (SIGIR feature)
41	Hyperlink base score propagation: uniform out-link (SIGIR feature)
42	Hyperlink base feature propagation: weighted in-link (SIGIR feature)
43	Hyperlink base feature propagation: weighted out-link (SIGIR feature)
44	Hyperlink base feature propagation: uniform out-link (SIGIR feature)

LETOR Features

Feature engineering for IR

1. Types of Features in IR

1. Text Preprocessing Features

- **Tokenization:** Splitting text into words or subwords.
- **Stopword Removal:** Eliminating frequent but non-informative words (e.g., "the", "is").
- **Stemming/Lemmatization:** Reducing words to their root form (e.g., "running" → "run").
- **Case Normalization:** Converting text to lowercase to avoid duplication.

2. Lexical Features

- **Term Frequency (TF):** Number of times a term appears in a document.
- **Inverse Document Frequency (IDF):** Measures term importance across documents.
- **TF-IDF:** Combined measure of term relevance.
- **BM25 Score:** A probabilistic ranking function improving TF-IDF.

. . .

3. Semantic Features

- **Word Embeddings (Word2Vec, GloVe, FastText):** Represent words in a dense vector space.
- **BERT Embeddings:** Context-aware embeddings using transformer models.
- **Synonyms & Named Entity Recognition (NER):** Expanding queries using knowledge bases.

4. Query-Document Matching Features

- **Exact Match:** Checks whether a query term appears in the document.
- **N-grams & Skip-grams:** Measures phrase similarities between query and document.
- **Jaccard Similarity:** Measures overlap of words in query and document.

5. Structural Features

- **Title Match Score:** Importance of query terms in document titles.
- **URL Length:** Shorter URLs might indicate high-ranking documents.
- **Click-through Rate (CTR):** Measures how often users click a document after searching.

. . .

6. Link-based Features

- **PageRank:** Measures the importance of a webpage based on links.
- **HITS Algorithm:** Differentiates between authoritative and hub pages.

7. Behavioral Features

- **Dwell Time:** Measures how long a user spends on a page after clicking.
- **Bounce Rate:** The percentage of users who leave a page quickly.
- **Query Reformulation:** Tracks if users modify their queries after retrieving results.

Evaluation of Machine Learning Models in IR

- Evaluating ML models in IR involves measuring how effectively they rank and retrieve relevant documents for given queries.
- Relevance-Based Metrics
  - Precision
  - Recall
  - F1-score
- Rank-Based Metrics
  - MAP
  - NDCG’s
- Click-Based Metrics
  - Click-Through Rate (CTR): Percentage of users clicking on retrieved documents.
  - Bounce Rate: Percentage of users leaving after clicking a search result.
- A/B Testing for Model Comparison
  - Deploy two models and compare user interactions.
  - Measure CTR, dwell time, and user engagement.

Click-Based Metrics in Information Retrieval (IR)

Click-based metrics evaluate user interactions with search results to assess the effectiveness of an IR system.

These metrics provide insights into how users engage with retrieved documents, helping to refine search algorithms and improve relevance.

Click-Through Rate (CTR)

- Click-Through Rate (CTR) measures how often users click on search results after performing a query. It indicates how attractive or relevant search results are to users.

$$\text{CTR} = \left(\frac{\text{Number of Clicks}}{\text{Number of Impressions}} \right) \times 100$$

- **Number of Clicks:** How many times users click on a result.
- **Number of Impressions:** How many times a result is displayed in the search engine.

. . .

Example

- Suppose a search result appears 1,000 times in response to a query.
- Out of those 1,000 impressions, 250 users click on the result.
- CTR = 250 / 1000×100=25%.

Interpretation

- **High CTR:** The search result is relevant and engaging.
- **Low CTR:** The result is not appealing (poor ranking, misleading title, or irrelevant content).

Improving CTR

- Optimizing **title and meta descriptions** to be more engaging.
- Using **rich snippets** (e.g., ratings, images).
- Improving **query-document relevance** using better ranking algorithms.

Bounce Rate

Definition

Bounce Rate measures the percentage of users who click on a search result but leave without interacting further (e.g., clicking another link or spending time on the page).

$$\text{Bounce Rate} = \left(\frac{\text{Number of Single Page Visits}}{\text{Total Clicks}} \right) \times 100$$

Number of Single Page Visits: Users who exit immediately after clicking a search result.

Total Clicks: Total users who clicked on a search result.

Example

- Suppose 200 users click on a search result, but 80 leave immediately without interaction.
- Bounce Rate = 80 / 200×100=40%

Interpretation

- **High Bounce Rate:**
  - The page content is **not relevant** to the user's query.
  - The page has a **bad user experience** (e.g., slow loading, poor design).
  - Users found the **answer directly** on the page (e.g., quick definitions).
- **Low Bounce Rate:**
  - Users **explore more content**, indicating engagement.
  - The page **meets user expectations** and provides useful information.

Reducing Bounce Rate

- Improving **content quality** to match search intent.
- Enhancing **website design** and navigation.
- Increasing **page speed** and mobile-friendliness.

A/B Testing for Model Comparison

- A/B testing is a controlled experiment where two models (A and B) are deployed to compare their effectiveness in retrieving relevant documents based on user interactions.

How It Works

1. **Split Traffic**
   - **Group A** sees search results from **Model A**.
   - **Group B** sees search results from **Model B**.
2. **Collect Metrics**
   - Compare **CTR**, **Bounce Rate**, **Dwell Time**, and **User Engagement** for both models.
3. **Analyze Results**
   - If **Model B shows better engagement (higher CTR, lower bounce rate, higher dwell time)**, it might be **better at ranking relevant documents**.

...

Example

- **Model A**: Uses **TF-IDF** for ranking.
- **Model B**: Uses **BERT embeddings** for ranking.
- A/B testing reveals **Model B has a 30% higher CTR** and **20% lower bounce rate**, indicating better relevance.

Key Metrics in A/B Testing

1. **CTR Comparison:** Higher CTR indicates better search results.
2. **Bounce Rate:** Lower bounce rate means users are engaging more.
3. **Dwell Time:** How long users stay on the clicked page.
4. **Conversion Rate:** If applicable, measures how many users take a desired action (e.g., making a purchase).

Queries???