Assignment 05 - Language Models for IR

Performance summary

✓ Assessed

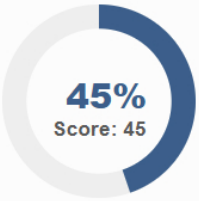
Success status	Score	Attempts
Undefined	45 of 100 points	1

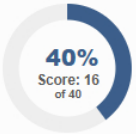
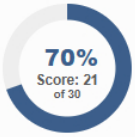
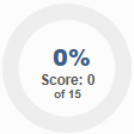
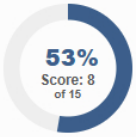
Results

Course	**Web Retrieval SoSe 2025** ID: 4853531344 / 109697254590984
Test	**Assignment 05: Underlying models (III) - Language Models for IR** ID: 4562682576

This are your test results

Duration	0h 36m 56s 5/24/2025, 1:44 PM - 5/24/2025, 2:21 PM
Answered	12 of 12 questions (100%)
Your score	45 of 100 points (45%)



1. Knowledge Check Tasks (40 points) 3	2. Practical Tasks (30 points) 2	3. Programming tasks (30 points) 1	Language Models (15 points) 6
			
go to section	go to section	go to section	go to section
✗ ✗ ✗	✗ ✗	✗	✓ ✗ ✓ ✗ ✓ ✗

1. Knowledge Check Tasks (40 points) 16 of 40 points (40%)

Language Model Concepts (20 points)

Status	Answered	
Your score	16 / 20	80%

Response

Which of the following sentences are true for language models (LM) in Information Retrieval?

Please note:

- Maximum Overall Score -> 20 points
- Minimum Overall Score -> 0 points
- Incorrect Answer -> -1 points
- Unanswered -> 0 points

Unanswered	Right	Wrong	
☐	☐	☑	A language model views queries as strings of text randomly sampled from documents.
☐	☑	☐	Unigram models consider the probability of words independent of their context.
☐	☐	☑	Bigram models are used to predict the probability of the next word based on two previous words.
☐	☐	☑	Trigram language models are frequently used in information retrieval due to their precision.

☐	☑	☐	The query likelihood model estimates the probability of a query being generated from a document's language model.
☐	☑	☐	Laplace smoothing is used to handle zero-frequency problems by assuming all unseen words are equally likely
☐	☑	☐	Dirichlet smoothing assigns fixed extra counts to unseen words based on their global probability.
☐	☐	☑	Language models are deterministic.
☐	☐	☑	The vector space model uses probabilistic intuitions like language models.
☐	☐	☑	In language models, term frequency does not influence the ranking of a document.
☐	☐	☑	Language models consider only the syntax of queries and not their semantics.
☐	☑	☐	A document that does not contain any of the query terms can still be ranked highly by language models if smoothing techniques are used.
☐	☑	☐	The use of a global language model in smoothing is to ensure that all words have a non-zero probability of occurring.
☐	☐	☑	In language models for IR, the relevance of a document increases with the number of unique terms it shares with a query.
☐	☑	☐	BERT inherently understands the context of terms as they appear in different scenarios.
☐	☐	☑	BERT-based query expansion only adds synonyms of query terms.
☐	☐	☑	BERT-based models can only process text in a single language.
☐	☐	☑	All modifications to BERT for query and document expansion have successfully eliminated the problem of query drift.
☐	☐	☑	Laplace smoothing is particularly effective for handling rare terms in large documents.
☐	☑	☐	Jelinek-Mercer smoothing uses a global language model to estimate the probability of unseen words.

▼ Solution

Which of the following sentences are true for language models (LM) in Information Retrieval?

Please note:

- Maximum Overall Score -> 20 points
- Minimum Overall Score -> 0 points
- Incorrect Answer -> -1 points
- Unanswered -> 0 points

Unanswered	Right	Wrong	
☐	☑	☐	A language model views queries as strings of text randomly sampled from documents.
☐	☑	☐	Unigram models consider the probability of words independent of their context.
☐	☐	☑	Bigram models are used to predict the probability of the next word based on two previous words.
☐	☐	☑	Trigram language models are frequently used in information retrieval due to their precision.
☐	☑	☐	The query likelihood model estimates the probability of a query being generated from a document's language model.
☐	☑	☐	Laplace smoothing is used to handle zero-frequency problems by assuming all unseen words are equally likely
☐	☐	☑	Dirichlet smoothing assigns fixed extra counts to unseen words based on their global probability.
☐	☐	☑	Language models are deterministic.
☐	☐	☑	The vector space model uses probabilistic intuitions like language models.
☐	☐	☑	In language models, term frequency does not influence the ranking of a document.
☐	☐	☑	Language models consider only the syntax of queries and not their semantics.
☐	☑	☐	A document that does not contain any of the query terms can still be ranked highly by language models if smoothing techniques are used.
☐	☑	☐	The use of a global language model in smoothing is to ensure that all words have a non-

☐	☑	☐	The use of a global language model in smoothing is to ensure that all words have a non-zero probability of occurring.
☐	☐	☑	In language models for IR, the relevance of a document increases with the number of unique terms it shares with a query.
☐	☑	☐	BERT inherently understands the context of terms as they appear in different scenarios.
☐	☐	☑	BERT-based query expansion only adds synonyms of query terms.
☐	☐	☑	BERT-based models can only process text in a single language.
☐	☐	☑	All modifications to BERT for query and document expansion have successfully eliminated the problem of query drift.
☐	☐	☑	Laplace smoothing is particularly effective for handling rare terms in large documents.
☐	☑	☐	Jelinek-Mercer smoothing uses a global language model to estimate the probability of unseen words.

BERT-Based Query Expansion (10 points)

Status	Answered	
Your score	0 / 10	0%

Response

Discuss the role of BERT in enhancing query expansion techniques for information retrieval. Analyze how BERT's understanding of language context improves the relevance and breadth of search results. Provide examples of how BERT-based query expansion might affect the retrieval process in practical search scenarios.

BERT (Bidirectional Encoder Representations from Transformers) provides deep understanding of language, which improves information retrieval technology's query expansion features. Most techniques to query expansion today are based on simplistic synonym listing procedures or co-occurrence statistic methods, which do not take deeper meaning into account and treat terms individually. Unlike BERT, which models language bidirectionally, considering the full context of a word, most models do not understand the meaning behind queries. This allows BERT to achieve much higher accuracy in understanding queries' semantic intent.

Deploying BERT in query expansion enables generation of expansions that are richer, context-aware, and incorporate concepts and phrases that are important to the user's actual intention. For instance, a query of "apple benefits" could be expanded to include "nutritional value," "health advantages," or "fruit vitamins" instead of unrelated interpretations involving the tech company. This type of expansion improves recall through identification of more relevant supporting documents while preserving precision through avoidance of irrelevant interpretations.

In practice, BERT-based expansions increase the relevance of search results retrieved in specialized domains containing complex language, like medical literature or legal documents, where terms have many meanings.The improvements enabled through incorporating BERT assist in comprehending nuanced search intents, allowing personalized or conversational search systems to do far more than simply respond to queries. Overall, BERT fills lexical gaps using semantically associative constructions, which enhances user satisfaction with search results.

229 words

▸ Solution

Smoothing Techniques (10 points)

Status	Answered	
Your score	0 / 10	0%

Response

Explain the purpose of smoothing techniques in language models used for information retrieval. Compare at least two different smoothing techniques and discuss how each technique affects the performance and reliability of language models in handling zero-frequency problems.

Smoothing strategies in language models for information retrieval remedy the zero-frequency issue, which occurs when certain query words do not appear in a document, resulting in the document being assigned a probability of zero by the model and consequently removed from retrieval. Smoothing aims to change probability estimates to ensure every word is present and has a non-zero probability value, which strengthens the model and enhances retrieval accuracy.

Two of the most widely used techniques are Jelinek-Mercer smoothing and Dirichlet prior smoothing:
1. Jelinek-Mercer smoothing utilizes linear interpolation between the document model and the collection (corpus) model. It adds a parameter λ which controls the contribution of the document's observed term frequency and the corpus frequency. Backing off to corpus probabilities for missing terms makes this technique simple to implement and effective. Choosing λ, however, is critically important because poor selections may incur over-smoothing (loss of document-specificity) or under-smoothing (failure to resolve zero-frequency problems).

2. Dirichlet prior smoothing, on the other hand, combines Bayesian priors by considering term counts to be derived from a Dirichlet distribution, effectively supplying pseudo-counts for each term based on corpus statistics. Smoothing is controlled by a parameter μ, which determines the strength of the prior's influence over the document model. Dirichlet smoothing adjusts to the length of documents, usually performs better with documents of differing sizes by providing more smoothing for shorter documents and less for

longer ones.

Both strategies enhance the effectiveness of language models by eliminating the zero probabilities, but due to its adaptability, retrieval results are often better with Dirichlet smoothing.

258 words

▸ Solution

‹ go back to overview

2. Practical Tasks (30 points) 21 of 30 points (70%)

We want to implement a query auto-complete feature for the interface of a search engine, using the past query history, which is the following five queries:

- $q1$: the weather forecast in Koblenz
- $q2$: next week weather forecast
- $q3$: the weather now
- $q4$: weather forecast hourly
- $q5$: the weather now in

Suppose the user types “the weather”. What will be probability of the top recommended next word?

Tip: You need to compute the probabilities P(the weather *X*), for all possible terms *X* of the vocabulary: *the*, *weather*, *forecast*, etc, that is *P(the weather the), P(the weather weather), P(the weather forecast),* etc. Then, the auto-complete module will recommend to the user the words ranked according to the above (joint) probabilities. The top recommended word (X1) is considered to be the one with the highest (joint) probability, i.e. P(the weather *X1*) > P(the weather *X2*)> … P(the weather *Xk*)

••• Bigram model (15 points)

Status	Answered	
Your score	10.5 / 15	70%

Response

Using the bigram language model, the top recommended word has a probability score of: 0.67

(round up to four decimal digits, do not add additional spaces)

{the, weather, forecast, in, Koblenz, next, week, now, hourly}

$P_{bi}(the\ weather\ the)$= 0

$P_{bi}(the\ weather\ weather)$ = 0

$P_{bi}(the\ weather\ forecast)$ = 0.33

$P_{bi}(the\ weather\ in)$ = 0

$P_{bi}(the\ weather\ Koblenz)$= 0

$P_{bi}(the\ weather\ next)$ = 0

$P_{bi}(the\ weather\ week)$ = 0

$P_{bi}(the\ weather\ now)$ = 0.67

$P_{bi}(the\ weather\ hourly)$ = 0

▾ Solution

Using the bigram language model, the top recommended word has a probability score of: 0.0947

(round up to four decimal digits, do not add additional spaces)

{the, weather, forecast, in, Koblenz, next, week, now, hourly}

$P_{bi}(the\ weather\ the)$= 0

$P_{bi}(the\ weather\ weather)$ = 0

$P_{bi}(the\ weather\ forecast)$ = 0.0947

$P_{bi}(the\ weather\ in)$ = 0

$P_{bi}(the\ weather\ Koblenz)$= 0

$P_{bi}(the\ weather\ next)$ = 0

$P_{bi}(the\ weather\ week)$ = 0

$P_{bi}(the\ weather\ now)$ = 0.0632

$P_{bi}(the\ weather\ hourly)$ = 0

••• Trigram model (15 points)

Status	Answered	
Your score	10.5 / 15	70%

Response

Using the trigram language model, the top recommended word has a probability score of: 0.67

(round up to four decimal digits, do not add additional spaces)

$P_{tri}(the\ weather\ the)$= 0

$P_{tri}(the\ weather\ weather)$= 0

$P_{tri}(the\ weather\ forecast)$= 0.33

$P_{tri}(the\ weather\ in)$= 0

$P_{tri}(the\ weather\ Koblenz)$= 0

$P_{tri}(the\ weather\ next)$= 0

$P_{tri}(the\ weather\ week)$= 0

$P_{tri}(the\ weather\ now)$= 0.67

$P_{tri}(the\ weather\ hourly)$= 0

▾ Solution

Using the trigram language model, the top recommended word has a probability score of: 0.1053
(round up to four decimal digits, do not add additional spaces)

$P_{tri}(the\ weather\ the)$= 0

$P_{tri}(the\ weather\ weather)$= 0

$P_{tri}(the\ weather\ forecast)$= 0.0526

$P_{tri}(the\ weather\ in)$= 0

$P_{tri}(the\ weather\ Koblenz)$= 0

$P_{tri}(the\ weather\ next)$= 0

$P_{tri}(the\ weather\ week)$= 0

$P_{tri}(the\ weather\ now)$= 0.1053

$P_{tri}(the\ weather\ hourly)$= 0

go back to overview

3. Programming tasks (30 points) 0 of 15 points (0%)

In the query likelihood model, the likelihood of a document *d* generating a query *q* is estimated as the quantity P(q|Md). Suppose we have two simple models for this estimation:

- An un-smoothed, unigram model: Puni(q|Md) = Πt∈q PMd(t)
- A linear-interpolated, unigram model (0 ≤ λ ≤ 1): Pinterp-uni(q|Md) = Πt∈q [λ. PMd(t) + (1-λ). PMc(t)]

where,

PMd(t): the probability of term *t* occurring in the document

PMc(t): the probability of term *t* appearing in the whole corpus

The document corpus is:

d1: vaccine research corona virus research

d2: research research cancer vaccine vaccine

d3: virus virus corona vaccine lab

d4: cancer lab research lab

go back to overview

Language Models (15 points) 8 of 15 points (53%)

☑ P_Mc(t) values (2 points)

Status	Answered		
Your score	2 / 2		100%

Response

Which of following elements have equal value for the PMc(t)?

Please note:

- One or more options are correct.
- **Maximum Overall Score -> 2 points**
- **Minimum Overall Score -> 0 points**
- **Incorrect Answer -> - 1 points**
- **Unanswered -> 0 points**

☑ virus, lab

☐ vaccine, virus

☑ corona, cancer

☐ lab, research

☐ research, cancer

▸ Solution

••• PMd(t) values (3 points)

Status	Answered	
Your score	0 / 3	0%

Response

If Bag of Words (BoW) is arranged alphabetically, the PMd(t) values for the term "vaccine" for documents (d1, d2, d3, d4) are:
[0.2,0.4,0.2,0.0]

Maximum Overall Score -> 3 points

Please write results in this format (use brackets [] and do not add additional spaces!),e.g. [0.9, 0.95, 0.93, 0.25] or [0.90, 0.95, 0.93, 0.25]

▾ Solution

If Bag of Words (BoW) is arranged alphabetically, the PMd(t) values for the term "vaccine" for documents (d1, d2, d3, d4) are:
[0.20, 0.40, 0.20, 0.0]

Maximum Overall Score -> 3 points

Please write results in this format (use brackets [] and do not add additional spaces!),e.g. [0.9, 0.95, 0.93, 0.25] or [0.90, 0.95, 0.93, 0.25]

☑ Query 1; Model 1 (3 points)

Status	Answered	
Your score	3 / 3	100%

Response

For the query "vaccine vaccine research cancer", if we calculate the ranked result set (sorted in descending order) according to the unsmoothed model, which documents appear having equal rank?

Please note:

- One or more options are correct.
- **Maximum Overall Score -> 3 points**
- **Minimum Overall Score -> 0 points**
- **Incorrect Answer -> - 1 point**
- **Unanswered -> 0 points**

☑ d1

☑ d3

☐ d2

☑ d4

▸ Solution

••• Query 1; Model 2 (2 points)

Status	Answered	
Your score	0 / 2	0%

Response

For the query "vaccine vaccine research cancer", we consider a linear-interpolated uni model with λ = 0.5. If we calculate the ranked result set (sorted in descending order), the Pinterp-uni values for documents d1,d2,d3 and d4 (round up to four decimal digits) are
[0.0007,0.0048,0.0003,0.00

Maximum Overall Score -> 2 points

Please write results in this format (use brackets [] and do not add additional spaces!)

[0.9999, 0.9995, 0.9990, 0.9025] or [0.9999, 0.9995, 0.999, 0.9025]

▾ Solution

For the query "vaccine vaccine research cancer", we consider a linear-interpolated uni model with λ = 0.5. If we calculate the ranked result set (sorted in descending order), the Pinterp-uni values for documents d1,d2,d3 and d4 (round up to four decimal digits) are
[0.0047, 0.0007, 0.0005, 0.(

Maximum Overall Score -> 2 points

Please write results in this format (use brackets [] and do not add additional spaces!)

[0.9999, 0.9995, 0.9990, 0.9025] or [0.9999, 0.9995, 0.999, 0.9025]

☑ Query 2; Model 1 (3 points)

Status	Answered		
Your score	3 / 3		100%

Response

For the query "vaccine research", if we calculate the ranked result set (sorted in descending order) according to the unsmoothed uni model, which document appears on the first rank?

Please note:

- One or more options are correct.
- **Maximum Overall Score -> 3 points**
- **Minimum Overall Score -> 0 points**
- **Incorrect Answer -> - 1 points**
- **Unanswered -> 0 points**

- [x] d2
- [ ] d4
- [ ] d3
- [ ] d1

▶ Solution

••• Query 2; Model 2 (2 points)

Status	Answered		
Your score	0 / 2		0%

Response

For the query "vaccine research", we consider a linear-interpolated uni model with lambda = 0.5. If we calculate the ranked result set (sorted in ascending order), the Pinterp-uni values (round up to four decimal digits) are: [0.0270,0.0270,0.0681,0.10

Maximum Overall Score -> 2 points

Please write results in this format (use brackets [] and do not add additional spaces)

[0.9025, 0.9990, 0.9995, 0.9999] OR [0.9025, 0.999, 0.9995, 0.9999]

▼ Solution

For the query "vaccine research", we consider a linear-interpolated uni model with lambda = 0.5. If we calculate the ranked result set (sorted in ascending order), the Pinterp-uni values (round up to four decimal digits) are: [0.027, 0.027, 0.0681, 0.10

Maximum Overall Score -> 2 points

Please write results in this format (use brackets [] and do not add additional spaces)

[0.9025, 0.9990, 0.9995, 0.9999] OR [0.9025, 0.999, 0.9995, 0.9999]

Document Retrieval with Vector Space Model (15 points)

Status	Answered		
Your score	0 / 15		0%

Response

Develop a Python program to implement a Vector Space Model (VSM) that retrieves documents based on the cosine similarity between a user-provided query and documents in a corpus.

```
# Corpus
documents = [
    "The sky is blue and beautiful.",
    "Love this blue and beautiful sky!",
    "The quick brown fox jumps over the lazy dog.",
    "A king's breakfast has sausages, ham, bacon, eggs, toast, and beans",
    "I love green eggs, ham, sausages and bacon!",
    "The brown fox is quick and the blue dog is lazy!",
    "The sky is very blue and the sky is very beautiful today",
    "The dog is lazy but the brown fox is quick!"
]
```

```
# Query
query = ["quick brown fox"]
```

1. Preprocess the documents and the query to tokenize and vectorize the text.
2. Calculate the TF-IDF weights for the corpus and the query.
3. Compute cosine similarity between the query and each document.
4. Display the documents ordered from the most relevant to the least relevant based on their similarity scores.

Hint: Scikit-learn provides utilities for text preprocessing, feature extraction, and similarity calculations, that you may use in your implementation.

- TfidfVectorizer: Converts a collection of raw documents to a matrix of TF-IDF features.
- cosine_similarity: Computes similarity between vectors, essential for ranking documents in VSM.

```
from collections import Counter
```

```
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def compute_term_probabilities(documents):
    corpus_terms = []
    doc_term_freqs = []
    doc_lengths = []

    for doc in documents:
        terms = doc.split()
        corpus_terms.extend(terms)
        counter = Counter(terms)
        doc_term_freqs.append(counter)
        doc_lengths.append(len(terms))

    corpus_counter = Counter(corpus_terms)
    corpus_length = len(corpus_terms)

    doc_term_probs = []
    for i, counter in enumerate(doc_term_freqs):
        length = doc_lengths[i]
        prob_dict = {term: freq/length for term, freq in counter.items()}
        doc_term_probs.append(prob_dict)

    corpus_term_probs = {term: freq/corpus_length for term, freq in corpus_counter.items()}

    return doc_term_probs, corpus_term_probs

def query_likelihood_unigram(query_terms, doc_term_probs, doc_lengths, doc_index):
    prob = 1.0
    doc_prob_dict = doc_term_probs[doc_index]
    length = doc_lengths[doc_index]
    for t in query_terms:
        pt = doc_prob_dict.get(t, 0)
        prob *= pt
    return prob

def query_likelihood_interpolated(query_terms, doc_term_probs, corpus_term_probs, doc_lengths, doc_index,
lam=0.5):
    prob = 1.0
    doc_prob_dict = doc_term_probs[doc_index]
    length = doc_lengths[doc_index]
    for t in query_terms:
        p_doc = doc_prob_dict.get(t, 0)
        p_corp = corpus_term_probs.get(t, 0)
        p = lam * p_doc + (1 - lam) * p_corp
        prob *= p
    return prob

corpus = [
    "vaccine research corona virus research",
    "research research cancer vaccine vaccine",
    "virus virus corona vaccine lab",
    "cancer lab research lab"
]

doc_term_probs, corpus_term_probs = compute_term_probabilities(corpus)
doc_lengths = [len(doc.split()) for doc in corpus]

query = "research vaccine".split()

print("Query Likelihood Unsmoothed Model Probabilities:")
for i in range(len(corpus)):
    prob = query_likelihood_unigram(query, doc_term_probs, doc_lengths, i)
    print(f"d{i+1}: {prob}")

print("\nQuery Likelihood Interpolated Model Probabilities (λ=0.7):")
lam = 0.7
for i in range(len(corpus)):
    prob = query_likelihood_interpolated(query, doc_term_probs, corpus_term_probs, doc_lengths, i, lam)
    print(f"d{i+1}: {prob}")

documents = [
    "The sky is blue and beautiful.",
    "Love this blue and beautiful sky!",
    "The quick brown fox jumps over the lazy dog.",
    "A king's breakfast has sausages, ham, bacon, eggs, toast, and beans",
    "I love green eggs, ham, sausages and bacon!",
    "The brown fox is quick and the blue dog is lazy!",
    "The sky is very blue and the sky is very beautiful today",
    "The dog is lazy but the brown fox is quick!"
]

query = ["quick brown fox"]

vectorizer = TfidfVectorizer()

tfidf_docs = vectorizer.fit_transform(documents)

tfidf_query = vectorizer.transform(query)

cos_similarities = cosine_similarity(tfidf_query, tfidf_docs).flatten()
```

```
ranked_doc_indices = np.argsort(-cos_similarities)

print("\nDocuments ranked by relevance to query:")
for rank, idx in enumerate(ranked_doc_indices, 1):
    print(f"{rank}. Doc{idx+1} (score: {cos_similarities[idx]:.4f}): {documents[idx]}")
```

333 words

▸ Solution

go back to overview

Test execution

Information

Availability: Expired at 5/29/2025, 1:59 PM

Max. attempts: Unlimited

Results of this test are visible to administrators and tutors of this course.

Start test

▸ Change log

Go to top