DAT640 2022 autumn, Final Exam - Answer Key

General approach to grading the essay questions
Even if a solution contains the answer that should give a certain score based on the provided solution key, points will be deducted if the answer (1) has a mix of correct and incorrect statements (-1 point) or (2) is too long or has largely irrelevant content w.r.t. the question (-1 point).

2 Normalization (3x1 point)

Normalize the following vector of values using min-max normalization: $< 2, 12, 7 >$**.**

Answer: < [0], [1], [0.5] >

3 Classification (2 points, -1 if incorrect)

Assume a multiclass classification problem with 3 categories that is decomposed into a binary classification problem. The one-against-one and one-against-rest strategies in this case require the same number of binary classifiers.

- (X) True
- ( ) False

4 Clustering (3x1 point)

Points

	w	x	y	z
P1	3	1	0	4
P2	2	0	1.5	1
P3	1	3	4.5	5
P4	2.5	2.5	5	3.5
P5	6	3	2	0

Centroids

	w	x	y	z
C1	3	5	4	4.5
C2	1.5	2.5	1	2
C3	2	3.5	1	0

You are given five 4-dimensional data points and three initial cluster centroids.
Perform the first iteration of K-means clustering using the Euclidean distance

Which cluster the following data points get assigned to?
(In case a data point is of equal distance to more than one cluster, pick the cluster with the lowest index.)

- P1 is assigned to C2
- P3 is assigned to C1
- P5 is assigned to C3

5 Retrieval (5x2 points)

	doc1	doc2	doc3	doc4
term 1	1	1	2	1
term 2		2		1
term 3	2		1	
term 4	4		1	2
term 5	1	2	1	

A term-document matrix is given above. We use a Language Modeling retrieval method with Dirichlet smoothing and the smoothing parameter (mu) set to 6.

- What is the probability of term5 in the empirical language model of doc1? [0.125]
- What is the probability of term4 in the background language model? [0.318]
- What is the probability of term2 in the (smoothed) language model of doc3? [0.074]
- Which term has the lowest probability in the (smoothed) language model of doc2? [term3]
- Which is the top scoring document for the query "term1 term3 term5"? [doc3]

6 Retrieval evaluation (4x2 points)

Evaluate two retrieval systems in terms of DCG@5 and NDCG@10 on a given search query. The table above contains the rankings generated by the two systems as well as the ground truth. Documents are judged on a 4-point scale: non-relevant (0), poor (1), good (2), excellent (3). The DCG formula to be used is also shown for your reference.

System A ranking	10, 7, 9, 8, 2, 1, 3, 4, 5, 6
System B ranking	3, 2, 1, 4, 5, 7, 8, 10, 9, 6
Ground truth	excellent: 1, 7 good: 2 poor: 3 (the rest are non-relevant)

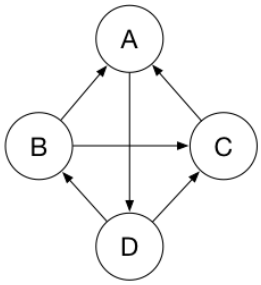
$$DCG@p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i}$$

- What is DCG@5 for System A? 3.861
- What is DCG@5 for System B? 4.893
- What is NDCG@10 for System A? 0.693
- What is NDCG@10 for System B? 0.78

7 Entity retrieval (3 points; -1 if incorrect)

Which of the following statements about predicate folding is *false*?

- ( ) It helps to deal with data sparsity.
- (X) It helps to preserve the semantics of RDF data.
- ( ) A single RDF triple may be mapped to multiple fields.
- ( ) It is not needed when there are only a handful of different predicates.



	Iteration 0	Iteration 1	Iteration 2
A	0.25	0.35	0.31
B	0.25	0.15	0.15
C	0.25	0.25	0.21
D	0.25	0.25	0.33

8 PageRank (8x1 point)

Compute the PageRank values for the following graph for two iterations.
The probability of a random jump (i.e., the parameter q) is 0.2.

9 Coding (4 points)

```python
from collections import Counter, defaultdict
from typing import Dict, List, Union

import nltk

nltk.download("stopwords")
STOPWORDS = set(nltk.corpus.stopwords.words("english"))


def preprocess(doc: str) -> List[str]:
    """Preprocesses a string of text.

    Args:
        doc: A string of text.

    Returns:
        List of strings.
    """
    ...


class InvertedIndex:
    def __init__(self):
        self.index = defaultdict(list)

    def add_posting(self, term: str, doc_id: str, freq: int) -> None:
        """Adds a document to the posting list of a term.

        Args:
            term: Term for which to add posting.
            doc_id: Document ID.
            freq: Number of times the term appears in the document.
        """
        self.index[term].append((doc_id, freq))

    def add_doc(self, doc: Dict[str, Union[str, int]]) -> None:
        """Preprocesses document and adds postings for terms.

        Args:
            doc: Document to index. Expected to be a dictionary with title
              and body. Title and body should be concatenated.
        """
        # TODO Complete this part
        # ...

```

Write the body of the `add_doc()` method that adds a document to an inverted index.

You can use the method `preprocess()` if necessary.

Example solution:

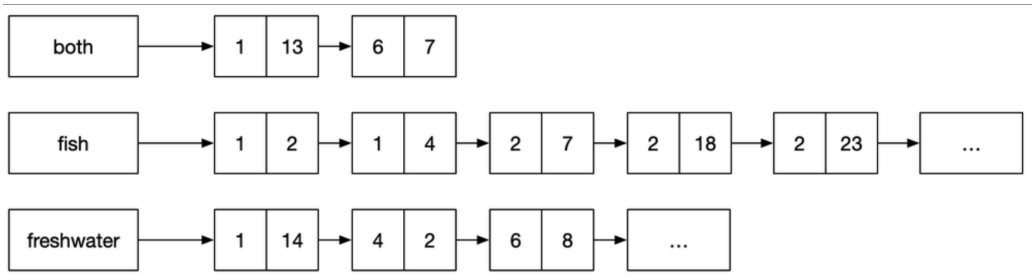
```python
def add_doc(self, doc: Dict[str, Union[str, int]]) -> None:
    """Preprocesses document and adds postings for terms.

    Args:
        doc: Document to index. Expected to be a dictionary with title
            and body. Title and body should be concatenated.
    """
    text = f'{doc["title"]} {doc["body"]}'
    for term, freq in Counter(preprocess(text)).items():
        self.add_posting(term, doc.get("id"), freq)
```

Scoring:

- 4 points if the implementation is correct; partial points may be awarded for iterating terms and frequencies (1p), using preprocessed text (1p), correctly determining term frequencies (1p), and adding postings to terms (1p).

10 Indexing (3 points)

You are given an excerpt from an inverted index. Select all statements that apply for this kind of index.



- [ ] The posting lists contain term frequencies and positions
- [X] The index is suitable for SDM retrieval
- [ ] Document IDs are stored in the payload
- [X] The length of a posting list will increase with the frequency of a term in any document

Scoring:

- 3 points if all correct
- 1.5 points if 2 correct and 1 incorrect
- 0.5 point if 1 correct and 1 incorrect
- 0 points otherwise

11 Coding (4 points)

```python
from typing import List, Tuple


def count_unordered_bigram_matches(
    bigram: Tuple[str, str], term_sequence: List[str], window: int = 4
):
    """Counts the number of unordered bigram matches in a given document field.

    Args:
        bigram: A sequence of two terms given as a tuple
        term_sequence: Sequence of terms in which to find bigram matches
        window: The maximum distance between the two query terms that still
          counts as a match

    Returns:
        Number of times the bigram can be found within a distance of `window`
        from each other in any order.
    """
    count = 0
    for i in range(len(term_sequence) - 1):
        if term_sequence[i] in bigram:
            other_term = (
                bigram[0] if term_sequence[i] == bigram[1] else bigram[1]
            )
            if other_term in term_sequence[i + 1 : i + window]:
                count += 1
    return count

```

Is the above implementation correct? If not, specify the problem line and give an input on which it fails. (bigram and text sequence)

Answer: No (1p), 25 and/or 26 ⇒ It will fail when there are multiple ‘other_term‘ terms inside the window (2p)
Example: (1p) count_unordered_bigram_matches((”t1”, ”t2”), [”t1”, ”t2”, ”t2”])

12 Coding (4 points)

```python
from typing import Dict, List

PREFIX = "##"
UNKNOWN = "<sep>"

VOCABULARY = {
    "s": 0,
    "sh": 1,
    "she": 2,
    "l": 3,
    "o": 4,
    "ck": 5,
    "##ck": 6,
    "##s": 7,
    "##sh": 8,
    "##she": 9,
    "##l": 10,
}


def tokenize(word: str, vocabulary: Dict[str, int]) -> List[str]:
    word = word.lower()
    tokens = []
    while word.lstrip(PREFIX):
        for i in range(len(word)):
            if word[: len(word) - i] in vocabulary:
                tokens.append(word[: len(word) - i])
                word = PREFIX + word[len(word) - i :]
                break
        else:
            return [UNKNOWN]
    return tokens

```

Describe in short what the above function does. What will be the output for `tokenize("shells", VOCABULARY)`? What will be the output for `tokenize("shellshock", VOCABULARY)`?

Answer: (1p) Tokenizes a single word based on the wordpiece vocabulary. (1p) Returns '<sep>' if it is not possible to fully tokenize the word.
Example 1: `["she", "##l", "##l", "##s"]` (1p)
Example 2: `["<sep>"]` (1p)

13 Coding (2x2 points)

```python
from typing import Dict, List

CollectionType = Dict[str, List]


class SimpleScorer:
    def __init__(
        self,
        index: CollectionType,
    ):
        """Interface for the scorer class.

        Args:
            index: Index to use for calculating scores.
        """
        self.index = index

        # Score accumulator for the query that is currently being scored.
        self.scores = None

    def get_postings(self, term: str) -> List:
        """Fetches the posting list for a given term.

        Args:
            term: Term for which to get postings.

        Returns:
            List of postings for the given term in the given field.
        """
        ...

    def score_collection(self, query_terms: List[str]):
        """Scores all documents in the collection using term-at-a-time query
        processing.

        Args:
            query_term: Sequence (list) of query terms.

        Returns:
            Dict with doc_ids as keys and retrieval scores as values.
            (It may be assumed that documents that are not present in this dict
            have a retrival score of 0.)
        """
        # TODO Complete this part
        ...

    def score_term(self, term: str, query_freq: int):
        """Scores one query term and updates the accumulated document retrieval
        scores (`self.scores`).

        Args:
            term: Query term.
            query_freq: Frequency (count) of the term in the query.
        """
        # TODO Complete this part
        ...

```

Implement the `score_collection()` and `score_term()` methods to perform term-at-a-time scoring using a simple retrieval function.

You can use the `get_postings()` method if necessary.

For scoring documents, employ the following simple retrieval function: $score(d, q) = \sum_{t \in q} w_{t,d} \times w_{t,q}$

Example solution:

```
def score_collection(self, query_terms: List[str]):
    """Scores all documents in the collection using term-at-a-time query
    processing.

    Args:
        query_term: Sequence (list) of query terms.

    Returns:
        Dict with doc_ids as keys and retrieval scores as values.
        (It may be assumed that documents that are not present in this dict
        have a retrival score of 0.)
    """
    self.scores = defaultdict(float)  # Reset scores.
    query_term_freqs = Counter(query_terms)

    for term, query_freq in query_term_freqs.items():
        self.score_term(term, query_freq)

    return self.scores

def score_term(self, term: str, query_freq: int):
    """Scores one query term and updates the accumulated document retrieval
    scores (`self.scores`).

    Args:
        term: Query term.
        query_freq: Frequency (count) of the term in the query.
    """
    postings = self.get_postings(term)
    for doc_id, payload in postings:
        self.scores[doc_id] += payload * query_freq
```

Scoring:

- 2 points for implementing `score_collection()` correctly; partial points may be awarded for initializing the scores (1p) and scoring each query term (1p).
- 2 points for implementing `score_term()` correctly; partial points may be awarded for getting term postings (1p) and updating the document score for each posting (1p).

14 Conversational Search Systems (3 points; -1 if incorrect)

Which of the following metrics take into consideration the turn depth and focus on deeper rounds to capture the ability of the system to understand the context of the whole conversation?

- ( ) NDCG@3 averaged for all turns in the given topic
- (X) Average NDCG@3 at each turn depth for all turns in all topics
- ( ) Mean Average Precision across all topics in the test dataset
- ( ) Recall@1000 across all topics in the test dataset

15 Fairness (2 points)

Select all statements that are correct about fairness in IR:

- [ ] Fairness has a unique definition in IR
- [X] Fairness does not have a unique definition in IR
- [X] Fairness can be studied at different levels

Scoring:

- 2 points if all correct
- 1.5 points if 2 correct and 1 incorrect
- 1 point if 1 correct
- 0.5 point if 1 correct and 1 incorrect
- 0 points otherwise

16 Conversational information access evaluation (2 points; -1 if incorrect)

Given the space of possible system responses and the conversation history up to that point, is it possible to create a reusable offline text collection to automatically evaluate system responses at a given conversation turn?

- (X) True
- ( ) False

17 Conversational information access (5x1 point)

Connect the given statements with the dialogue system components. You are expected to know what the acronyms stand for.

Statement	NLU	ST	DP	NLG
Determines both the current state of the frame and the user's most recent dialogue act	[ ]	[X]	[ ]	[ ]
Responsible for sentence realization and in some cases content planning	[ ]	[ ]	[ ]	[X]
Responsible for determining domain and intent	[X]	[ ]	[ ]	[ ]
Does slot filling for the current utterance	[X]	[ ]	[ ]	[ ]
Decides what dialogue act to generate	[ ]	[ ]	[X]	[ ]

Scoring: 1p for each correct association.

18 Conversational information access (3 points)

Which of the following statements are true for conversational AI?

- [ ] Chatbots are task-based dialogue systems whose task is to entertain people
- [ ] Chatbot architectures fall into two classes: rule-based systems and template-based systems
- [X] A frame in frame-based dialogue systems represents the kinds of intentions the system can extract from user sentences
- [X] A frame in frame-based dialogue systems consists of a collection of slots, each of which can take a set of possible slot values

Scoring:

- 3 points if all correct
- 1.5 points if 2 correct and 1 incorrect
- 1 point if 2 correct and 2 incorrect
- 0.5 point if 1 correct and 1 incorrect
- 0 otherwise

19 Neural IR (2 points)

Which of the following statements about negative sampling used in dense retrieval are *false*?

- [ ] The top non-relevant documents retrieved by a retrieval system given a query are good negative examples for fine-tuning a model.
- [X] Negative samples are used to fine-tune a neural model for sparse retrieval task.
- [X] Random and in-batch negatives are very informative for a model because they are less relevant than the top match for a given query.
- [ ] The main goal of the negative sampling is helping the model with placing query and the relevant documents close to each other in the embedding space and far away from non-relevant documents.

Scoring:

- 2 points if all correct
- 1.5 point if 2 correct and 1 incorrect
- 1 point if 2 correct and 2 incorrect
- 0.5 point if 1 correct and 1 incorrect
- 0 points otherwise

20 Neural IR (3 points)

What are the main characteristics of an interaction-focused ranking system? Focus on the architecture, main components, and relevance estimation in your answer.

- (1p) two-stage pipeline
- (1p) unsupervised retrieval method for first stage
- (1p) neural re-ranker for second stage
- (1p) using fine-tuned language models for computing relevance score
- (1p) explicitly constructed interaction between a query and a document

21 Neural IR (2 points)

What are the main advantages of using a contextualized distributional text representation in retrieval compared with a discrete text representation?

- (1p) much lower dimension of the representation
- (1p) taking into consideration the ordering of words
- (1p) learning semantics and grammar information

22 Entity linking (3 points, -1 if incorrect)

Which of the following statements about entity disambiguation is *false*?

- ( ) Incorporating context and dependence between entities increases complexity
- (X) Individual local disambiguation is NP-hard
- ( ) Disambiguation approaches make simplifying assumptions for computational tractability
- ( ) Commonness in itself can be used as a disambiguation approach

23 Entity linking (4x1 point)

```
This is some text where [[entityA|A]] is first mentioned.
[[entityA|A]] often co-occurs with [[entityB|B]], who is known as [[entityB|The King]].

[[entityB|B]] also named his son [[entityD|B]], aka [[entityD|King Jr.]].

Others known as "The King" include [[entityC|C]] and [[entityD|D]].

The programming language [[entityCPP|C++]] is the successor of [[entityC_prog|C]].
```

You are given the above entity-annotated document collection with Wikipedia link syntax, i.e., [[Wikipedia page ID|link text]]. Assuming that all Wikipedia pages are to be treated as entities, **compute the following *commonness scores***:

- P(e=entityA|m="A") : [1.0]
- P(e=entityB|m="B") : [0.666]
- P(e=entityC|m="C") : [0.5]
- P(e=entityD|m="The King") : [0]

24 Statistical significance testing (2 points; -1 if incorrect)

In a randomization test, the null hypothesis H_0 is that two systems are identical with regards to a test statistic.

- (X) True
- ( ) False

25 Retrieval evaluation (4 points)

You are given ranked lists of n documents from m different retrieval system. You wish to obtain the relevance judgements for the retrieved documents but your budget only allows for annotating a fraction of documents (i.e., number of documents to annotate is $<< n \times m$). **Describe how to choose the subset of documents to judge? What about the remaining documents?**

- (1p) Select top-k documents from each of the m retrieval system.
- (1p) Combine selected documents into a single pool and prune duplicate documents.
- (1p) The documents in the pool are judged.
- (1p) The remaining unjudged documents are assumed to be non-relevant.

26 Coding (3 points)

```python
def query(
    query_terms: List[str],
    doc_id: str,
    field: str,
    es: Elasticsearch,
    index: str,
) -> Dict[str, float]:
    """Extracts features
    .
    Args:
        query_terms: List of analyzed query terms.
        doc_id: Document identifier of indexed document.
        field: The name of the field in the index.
        es: Elasticsearch object instance.
        index: Name of relevant index on the running Elasticsearch service.

    Returns:
        Dictionary with keys 'feature_1', 'feature_2'.
    """
    features = {}

    features["feature_1"] = 0
    query_terms_tf = []

    # Gets the term frequencies of a field of an indexed document
    term_freqs = get_doc_term_freqs(es, doc_id, field, index=index)

    if term_freqs is None:
        term_freqs = {}
    for term in query_terms:
        query_terms_tf.append(term_freqs.get(term, 0))
    for term in set(query_terms):
        if term in term_freqs:
            features["feature_1"] += 1

    features["feature_2"] = (
        sum(query_terms_tf) if len(query_terms_tf) > 0 else 0
    )

    return features
```

Which features are extracted in this piece of code?

- [ ] Average term frequency in of each query term
- [X] Count of uniques terms present in the document field
- [ ] Count of tokens in the document field
- [X] Sum over the term frequencies of each query term
- [ ] Sum of TF scores of query terms in the document field

Scoring:

- 3 points for all correct
- 2.5 points if 2 correct and 1 incorrect
- 0.5 points if 1 correct and 1 incorrect
- 0 otherwise

27 Relevance feedback (2 points)

Which of the following statements about relevance feedback are *true*?

- [X] Pseudo releance feedback adjusts a query relative to the documents that initially appear to match the query
- [X] Relevance feedback usually extends the original query with additional terms
- [ ] The weights in the expanded query are adjusted in such a way that they are always higher for the original query terms than for expansion terms
- [X] In case of blind feedback the user is not directly involved in selecting the relevant documents

Scoring:

- 2 points if 3 correct
- 1 point if 2 correct
- 0.5 point if 2 correct and 1 incorrect
- 0.5 point if 1 correct
- 0 points otherwise

28 Retrieval (2x2 points)

	title	body	anchors
Doc 1	t1	t1 t2 t3 t1 t3	t2 t2
Doc 2	t4 t5	t1 t3 t4 t4 t4 t5	t5 t3
Doc 3	t1 t3 t5	t1 t1 t5 t3 t5 t3 t3	t1 t1 t5

- Fielded BM25 is defined as:

$$BM25F(d,q) = \sum_{t \in q} \frac{\tilde{c}_{t,d}}{k_1 + \tilde{c}_{t,d}} \times idf_t \tag{1}$$

$$\tilde{c}_{t,d} = \sum_i w_i \times \frac{c_{t,d_i}}{B_i} \tag{2}$$

  where

  - i corresponds to the field index
  - w_i is the field weight (assume $[w_1 = 0.1, w_2 = 0.7, w_3 = 0.2]$)
  - B_i is soft normalization for field i: $B_i = (1 - b_i + b_i \frac{|d_i|}{avgdl_i})$ (assume $b_1 = b_2 = b_3 = 0.75$)

  IDF values should be computed based on the body field using natural-base logarithm.

Given the document-term matrix corresponding to the document collection, and formulae shown above, answer the following:

- What is the value of BM25F scoring function for query "t3" and document 1? 0
- What is the value of BM25F scoring function for query "t2 t4" and document 2? 0.7108