DAT640 2023 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 Similarity (2x1.5 points)

$$\begin{aligned} x &= (1,0,0,1,1,0,1,1,0,1) \\ y &= (1,1,0,1,0,0,1,0,1,1) \end{aligned}$$

Calculate the similarity of the above two binary vectors.

- Jaccard similarity: 0.5
- Cosine similarity: 0.666

3 Classification (3 points)

Assume a multiclass classification problem with 5 categories. Using the one-against-one strategy, how many binary classifiers are needed in total?

Answer: [$\frac{5*4}{2} = \mathbf{10}$]

4 Indexing (3 points)

Select all statements that are correct regarding the payload of a posting in an inverted index.

- [X] The payload is not required in a posting
- [ ] Postings with payload require less memory than postings without payload
- [ ] Document IDs are stored in the payload
- [X] Postings with payload supports more ranking algorithms

Scoring:

- 3 points if all correct; otherwise, 1 point for each correct, -0.5 point for each incorrect answer
- For example: 1.5 points if 2 correct and 1 incorrect; 0.5 point if 1 correct and 1 incorrect
- Minimum points is 0

5 Coding (2 points; -1 if incorrect)

What would be the time complexity of the `score_collection` method performing term-at-a-time scoring assuming that we have n query terms, m documents, and k as the length of the average posting list?

- ( ) $\mathcal{O}(n \cdot m)$
- ( ) $\mathcal{O}(k \cdot m)$
- (X) $\mathcal{O}(n \cdot k)$
- ( ) $\mathcal{O}(n \cdot k \cdot m)$

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


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
```

6 Coding (2 points)

Assume you have an Elasticsearch index with three documents (without any analysis performed). Which of these document IDs will be returned in res['hits']['hits']?

- [X] 1
- [ ] 2
- [ ] 3

Scoring: 2 points for correct selection, otherwise 0.

7 Retrieval (5x2 points)

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

```
DOCS = {
    1: {"title": "All Along The Watchtower",
        "content": "There must be some way out of here Said the joker to the thief \
        There's too much confusion I can't get no relief"
        },
    2: {"title": "Land of Confusion",
        "content": "There's too many men, too many people Making too many problems \
        And not much love to go round Can't you see this is a land of confusion?"
        },
    3: {"title": "Nowhere Near",
        "content": "How easy I forget Just how you add to my confusion So I'm out of here \
        Cause I know I'm nowhere near What you want, What you want, What your lookin for"
        },
}

# ...

query = {'match_phrase': {'content': "too much confusion"}}
res = es.search(index=INDEX_NAME, body={'query': query})
```

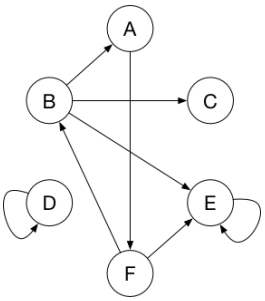
	doc1	doc2	doc3	doc4
term1	1	1	2	1
term2		2		1
term3	2		1	
term4	4		1	2
term5	1	2	1	

to 6.

Answer the following questions: (2p each)

- What is the probability of term2 in the empirical language model of doc2? [0.4]
- What is the probability of term5 in the background language model? [0.182]
- What is the probability of term1 in the (smoothed) language model of doc4? [0.24]
- Which term has the highest probability in the (smoothed) language model of doc2? [term5]
- Which is the top scoring document for the query "term5 term2"? [doc2]

8 PageRank (8x1 point)



Compute the PageRank values for the above graph for the first two iterations and provide the required values in the table below.
The probability of a random jump (i.e., the parameter q) is 0.2.

	Iteration 0	Iteration 1	Iteration 2
A	0.167	(0.1)	(0.079)
B	0.167	0.122	0.122
C	0.167	0.1	(0.079)
D	0.167	0.188	0.197
E	0.167	(0.3)	0.394
F	0.167	0.188	0.126

9 Retrieval (3 points)

In learning-to-rank, usually an initial retrieval round is performed to retrieve the top-N documents for the query using a baseline retrieval model (e.g., BM25). Then, those top-N documents are re-ranked using supervised learning. Why is this intermediate step necessary, i.e., why not use supervised learning directly on the entire document set?

Scoring all document in a collection is infeasible (as in computationally too expensive) at *retrieval time* (2p). Specifically, learning-to-rank involves computing features that are based on the query, i.e., cannot be pre-computed (1p).

It can be inferred from the wording of the question ("re-ranking") that it asks about LTR at ranking time, not at training time. Therefore, a point may be deducted if the answer talks about the training and not the ranking (inference) process. Further points may be deducted for incorrect/irrelevant reasons in the answer (such as filtering noise or labeling cost).

10 Relevance feedback (2 points; -1 if incorrect)

Which of the following statements is *false*?

- ( ) Implicit feedback is noisier than explicit feedback
- ( ) The Rocchio algorithm needs a set of annotated documents
- (X) Relevance feedback always improves recall

11 Retrieval Evaluation (5x2 points)

	Query 1	Query 2
Algorithm A	1, 2, 6, 5, 9, 10, 7, 4, 8, 3	1, 2, 4, 5, 7, 10, 8, 3, 9, 6
Algorithm B	10, 9, 8, 7, 5, 4, 6, 2, 1, 3	1, 3, 2, 4, 5, 6, 8, 7, 10, 9
Ground truth	1, 4, 5	3, 6

Table 1: Retrieval evaluation.

The table shows, for two queries, the document rankings produced by ranking two different algorithms along with the list of relevant documents according to the ground truth. We assume that relevance is binary.

Answer the questions below.

- What is P5 (precision at rank 5) of Algorithm A on Query 1? [0.4]
- What is the Average Precision of Algorithm A on Query 1? [0.625]
- What is the Reciprocal Rank of Algorithm B on Query 2? [0.5]
- What is the Mean Reciprocal Rank of Algorithm B? [0.35]
- Which algorithm has higher Mean Average Precision? [A/B/the same]

12 Statistical significance testing (2 points)

Select all statements that are correct for Student's t-test:

- [ ] Any test statistic can be used
- [X] The systems compared follow a normal distribution
- [ ] The test statistic is recorded for several permutations of the systems' outputs

Scoring:

- 2 points if all correct
- 0.5 point if correct answer and one incorrect answer are selected
- 0 points otherwise

13 Retrieval evaluation (3 points)

Which of the following statements about creating assessment pools for retrieval systems is false?

- [ ] Only the top-k documents from each retrieval system (where k is much smaller than the number of documents in the collection) should be chosen
- [ ] The documents not included in the assessed pool are assumed to be non-relevant
- [X] The assessors are presented with documents in the order in which they are retrieved by the system
- [ ] Greater pool depth ensures that more of the relevant documents can be identified

Scoring:

- 3 points if only the correct answer is selected
- 1 point if the correct answer and one incorrect is selected
- 0 point otherwise

14 Conversational Search Systems (2 points)

Which of the following search tasks would be best addressed using a conversational user interface?

- [ ] Ad-hoc search
- [X] Searching for an item with rich attributes that can be individually specified, but are much simpler to provide piecewise
- [ ] Memoryless refinement where the user learns the right terms to describe their information need by iterating with a search system but each query is ad-hoc search
- [X] Planning a vacation where the results consist of a hotel, travel arrangements, restaurant plans, and places to see

Scoring:

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

15 Coding (4 points)

	qid	query	topic_number	turn_number
0	4_1	What was the neolithic revolution?	4	1
1	4_2	When did it start and end?	4	2
2	4_3	Why did it start?	4	3
3	4_4	What did the neolithic invent?	4	4
4	4_5	What tools were used?	4	5
...	...	...	...	...
248	105_5	Who named the movement?	105	5
249	105_6	What was the US reaction to it?	105	6
250	105_7	Tell me more about the movement of the police ...	105	7
251	105_8	Why were they killed?	105	8
252	105_9	What else motivates the Black Lives Matter mov...	105	9

```python
import pandas as pd
from transformers import T5ForConditionalGeneration, T5Tokenizer

SEPARATOR = "|||"
MODEL_NAME = "castorini/t5-base-canard"

# Load the model and tokenizer from HuggingFace
model = T5ForConditionalGeneration.from_pretrained(MODEL_NAME)
tokenizer = T5Tokenizer.from_pretrained(MODEL_NAME)

# Load topics
topics = pd.read_csv("topics.csv")


def create_query_rewrites(topics: pd.DataFrame) -> pd.DataFrame:
    """Create query rewrites.

    Args:
        topics: A dataframe containing the queries for each topic.

    Returns:
        Modified dataframe with the query rewrites.
    """
    rewrites = pd.DataFrame()
    for _, topic in topics.groupby("topic_number"):
        topic.reset_index(inplace=True, drop=True)
        for i, row in topic.iterrows():
            if i == 0:
                topic.at[i, "rewrite"] = row["query"]
                continue

            # TODO: Complete this function so the variable rewrite contains the
            # rewritten query.
            rewrite = ...
            topic.at[i, "rewrite"] = rewrite
        rewrites = pd.concat([rewrites, topic])
    return rewrites
```

Complete the part marked with # TODO in the `create_query_rewrites` method. This method creates query rewrites for each topic using T5. A query rewrite is based on the previous and current queries of the topic.

A part of the dataset containing the topics and their queries is displayed above. Hint: you should use the methods `encode` and `decode` of the T5 tokenizer and the method `generate` of the T5 model.

NB: You only need to provide the code that goes in place of the # TODO above.

Example solution:

```
previous_query = topic.loc[i - 1]["query"]
input_text = f"{previous_query} {SEPARATOR} {row['query']}"
input_ids = tokenizer.encode(input_text, return_tensors="pt")
outputs = model.generate(input_ids)
rewrite = tokenizer.decode(outputs[0], skip_special_tokens=True)
```

Scoring:

- 4 points if implementation is correct; partial points may be awarded for concatenating the previous and current queries with separator (1p), encoding the input (1p), generating the output (1p), and decoding the output correctly using the tokenizer (1p).

16 Coding (7 points)

```
import re
from typing import Dict, List


def create_entity_repr(docs: List[str], window_size: int) -> Dict[str, str]:
    """Creates entity representations from mention-annotated documents.

    Example input:

    docs = [
      "first document that mentions [[entity1|entity-one]] alone",
      "2nd document with [[entity2|ABC]] and [[entity1|entity-one]] together",
      "xxx yyy zzz [[entity2|ABC]] aaa bbb ccc ddd [[entity3|ZZZ]] eee fff",
    ]

    By calling this method with window_size=2, the generated output will be

    {
      "entity1": "that mentions entity one-alone ABC and entity-one together",
      "entity2": "document with ABC and entity-one yyy zzz ABC aaa bbb",
      "entity3": "ccc ddd ZZZ eee fff",
    }

    Args:
        documents: List of documents with mention-level entity annotations.
        window_size: Size of the context window (in terms).

    Returns:
        A dictionary indexed with entity IDs, holding the corresponding entity
        representations as values.
    """
    # TODO
```

Write a method that takes a collection of documents that contain mention-level entity annotations and creates term-based entity representations from them.

Entities are marked up using wiki-style piped links, i.e., are in [[entityID|text]] format. For simplicity, the text component of these links does not contain spaces. (You may assume that the input is syntactically correct with regards to these annotations.)

For each entity that is mentioned in the input documents, create an entity representation document by concatenating the terms within a given windows size around each of its mentions (i.e., take window_size terms before and window_size terms after its mention as well as the text annotated with the entity itself). Entity-annotated texts inside [[ ]] should be treated as a single term! You may assume that the input text has been preprocessed and that you can simply split to terms on spaces.
Above, you can see the signature of the method that you need to implement along with an example input and corresponding output. You may assume that the re package has been imported. The use of additional packages is not allowed.

NB: You only need to write the body of the method (i.e., that comes in place of # TODO) in the input area below.
You need to submit code that runs and produces output in the required format, as this exercise will be graded automatically based on how many of the tests it passes.

Example solution:

```
def create_entity_repr(docs: List[str], window_size: int) -> Dict[str, str]:
    entity_repr = {}
    pattern = r'\[\[(.*?)\|(.*?)\]\]'

    for doc in docs:
        terms = doc.split()

        for i, term in enumerate(terms):
            match = re.match(pattern, term)
            if match:
                entity_id, text = match.groups()
                start = max(0, i - window_size)
                end = min(len(terms), i + window_size + 1)

                context = ' '.join(terms[start:i] + [text] + terms[i+1:end])

                # Need to remove other entity markup from context
                context = re.sub(pattern, r'\2', context)

                if entity_id not in entity_repr:
                    entity_repr[entity_id] = context
                else:
                    entity_repr[entity_id] += ' ' + context

    return entity_repr
```

Scoring based on tests:

- (1p) Empty input or no entity annotations at all
- (1p) Single document, single entity
- (1p) Multiple documents, single entity
- (1p) Multiple documents, multiple entities, but no other entity mentions present in context windows
- (3p) Multiple documents, multiple entities, with other entities mentions also present in context windows

17 Retrieval (3 points; -1 if incorrect)

Which of the following statements about the sequential dependence model (SDM) is *false*?

- ( ) The ranking function is a weighted combination of feature functions
- ( ) It is a particular Markov random field model
- ( ) It belongs to the class of linear feature-based models
- (X) The feature functions estimate term/bigram frequencies combined across multiple fields

18 Retrieval (3 points)

You are given a small collection of documents, $D = \{d_1, d_2, d_3\}$, and a query q, each consisting of a sequence of terms t_i:

$d_1 = \langle t_1\ t_2\ t_3\ t_4\ t_5\ t_6 \rangle$
$d_2 = \langle t_3\ t_4\ t_3\ t_1\ t_8\ t_2\ t_2\ t_7 \rangle$
$d_3 = \langle t_2\ t_9\ t_4\ t_1\ t_8\ t_2\ t_3\ t_1\ t_4 \rangle$
$q = \langle t_4\ t_3 \rangle$

The SDM scoring function:

$$score(d, q) = \lambda_T \sum_{i=1}^{n} f_T(q_i, d) + \lambda_O \sum_{i=1}^{n-1} f_O(q_i, q_{i+1}, d) + \lambda_U \sum_{i=1}^{n-1} f_U(q_i, q_{i+1}, d) \tag{1}$$

The weights are given as $\lambda_T = 0.85$, $\lambda_O = 0.1$, $\lambda_U = 0.05$.
The specific feature functions are:

Unigram matches:

$$f_T(q_i, d) = \log P(q_i|\theta_d) \tag{2}$$

Ordered bigram matches:

$$f_O(q_i, q_{i+1}, d) = \log \left(\frac{c_o(q_i, q_{i+1}, d) + \mu P_o(q_i, q_{i+1}|D)}{|d| + \mu} \right) \tag{3}$$

Unordered bigram matches:

$$f_U(q_i, q_{i+1}, d) = \log \left(\frac{c_w(q_i, q_{i+1}, d) + \mu P_w(q_i, q_{i+1}|D)}{|d| + \mu} \right) , \tag{4}$$

Note the use of the logarithm (base 2) and the use of the Dirichlet smoothing with parameter $\mu = 6$. Also $|d|$ is the length of a document. Use a window of $w = 4$ terms for the unordered bigrams.

What is the value of the unordered bigram feature function for "t4 t3" in document d3?

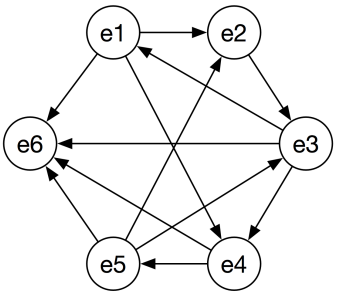
- -2.876
- -1.716
- 0.3043

19 Entity linking (2 points; -1 if incorrect)

Entity	count
Superman	1000
Superman (comic book)	120
Superman (1978 film)	50
Superman (film series)	27
Superman (1999 video game)	3

The table shows all the different entities and counts from a surface form dictionary for the entry (i.e., surface form) "superman". **Which entity has a commonness score of 0.1?**

- Superman
- Superman (comic book)
- Superman (1978 film)
- Superman (film series)
- Superman (1999 video game)
- None of them



20 Entity linking (3 points)

$$WLM(e, e') = 1 - \frac{\log\left(\max(|\mathcal{L}_e|, |\mathcal{L}_{e'}|)\right) - \log(|\mathcal{L}_e \cap \mathcal{L}_{e'}|)}{\log(|\mathcal{E}|) - \log\left(\min(|\mathcal{L}_e|, |\mathcal{L}_{e'}|)\right)} \tag{5}$$

- $\mathcal{L}_e$ is the set of entities that link to e
- $|\mathcal{E}|$ is the total number of entities

What is the relatedness (Wikipedia Link-based Similarity) score between entities 3 and 6, based on their incoming links?

(Use base 2 for log.)

WLM(e3, e6) = [-0.26]

21 Entity-oriented search (3 points, -1 if incorrect)

Which of the following considerations is *incorrect* for an evaluation measure designed specifically for the *target type identification* task?

- ( ) Not all types of mistakes are equally bad, near-misses should be graded
- ( ) Hierarchical relationships in the type taxonomy should be taken into account
- ( ) Multiple correct ground truth types should be supported
- (X) It should not matter whether a wrong answer is located on the same branch than the correct answer or on a different one

22 Entity linking (2 points)

Name two main differences between entity linking in queries and entity linking in documents.

- (1p) queries are much shorter than documents (limited context)
- (1p) because of the limited context, the ambiguity may not be possible to resolve in queries
- (1p) purpose: understanding info needs in queries vs. understanding document content
- (1p) entity linking in queries has to happen real time ("online"), while for documents it can be performed "offline"
- (1p) evaluation measures are different (single interpretation vs. multiple interpretations)
- (1p) queries often lack proper grammar

23 Coding (2 points)

```python
from typing import List, Tuple

SkipGrams = Tuple[str, str]


def generate_positive_examples(sentence: str, l: int) -> List[SkipGrams]:
    """Generates positive examples for the given sentence.

    Args:
        sentence: Sentence.
        l: Window size.

    Returns:
        List of positive examples.
    """
    positive_examples = []
    tokens = sentence.split()
    for i, token in enumerate(tokens):
        for j in range(i, i + l + 1):
            if j < 0 or j >= len(tokens) or i == j:
                continue
            positive_examples.append((token, tokens[j]))
    return positive_examples
```

The code above is used to create positive examples for a given sentence in order to train a Word2Vec Skip-gram with negative sampling model.

Is there an error in the implementation of the method `generate_positive_examples`, and if yes, what is it?

The error is line 19, the window should comprise the words before and after the current word, not only the words after.

Scoring:

- 2 points if the error is correctly identified and explained; partial points may be awarded if the error is found at the correct place but is not sufficiently explained.
- Identifying line 19 but for wrong reasons, which are actually not errors (e.g., indexing out of bounds or range is off by one) gives 0 point.
- Points may be deducted if the answer proposes to change parts of the code that don't need fixing.

24 Word2Vec Skip-gram (2 points)

Sentence: "The curious cat explored the mysterious garden under the pale moonlight."

Lexicon: [the, curious, cat, explored, mysterious, garden, under, pale, moonlight]

Window size: 3

Given the sentence, lexicon, and window size above, select the examples that are not valid negative training examples for a Skip-gram model.

- [ ] (cat, moonlight)
- [ ] (garden, curious)
- [X] (moonlight, the)

- [ ] (curious, mysterious)

Scoring:

- 2 points if correct
- 0 point otherwise

25 Neural Retrieval (2p)

Which of the following statements about neural ranking is/are correct?

- [ ] The interaction matrix is created using a Learning-to-rank method.
- [X] Interaction-focused ranking uses Transformers to calculate relationships between queries and a documents.
- [X] An approximate nearest neighbor index substantially improves the speed of finding document embeddings closest to query embeddings.
- [ ] Approximate Nearest Neighbor algorithms provide exact matches of the nearest neighbors for any given query.

Scoring:

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

26 Neural IR (2 points)

Name two of the main differences between BERT-based word embeddings and word2vec word embeddings.

- (1p) contextual (multiple vector representations for each word) vs. static word embeddings (single vector representation for each word)
- (1p) model architecture: deep bidirectional transformer vs. shallow neural network
- (1p) pre-training objective (masked language modeling and next-sentence prediction vs. predicting words)
- (1p) utilized by further fine-tuning on downstream tasks vs. as features in ML models
- (1p) operates on subword units (WordPieces) vs. fixed vocabulary of words
- (1p) word order within a sentence is taken into account vs. not

27 Large Language Models (3 points)

Create a few-shot prompt to categorize animals into "domestic" and "wild" categories using ChatGPT or similar GPT-based model.

Example solution:
Categorize the following animals as either "domestic" or "wild".

horse ⇒ domestic
dog ⇒ domestic
deer ⇒ wild
bear ⇒ wild
[Animal Name] ⇒

Scoring:

- 3 points if a prompt with a few examples is given, it is straightforward to apply it on a new animal and ChatGPT correctly categorizes a new animal (e.g. elephant ⇒ ... returns "wild")
- 2 points if few-shot examples are provided, but the model, but the input animal(s) to categorize is not provided in the prompt.
- 1 point if the prompt does not contain few-shot examples (such as horse ⇒ "domestic"), but works as zero-shot.
- 0 point otherwise.

28 Cross-language Retrieval (3 points)

You would like to create information retrieval system for search in Norwegian documents using English queries. Describe very briefly three possible approaches that you could use.

Possibilities are following:

- Document translation
- Query translation (and applying retrieval on top of the translations)
- Multilingual embeddings
- Probabilistic structured queries (PSQ) as a form of n-best translation
- Pivot language
- Interlingua (language-independent representation)

Each of these possibilities is for one point (3 points maximum).

29 User Simulation (3 points; -1 if incorrect)

Information need

$$C_0 = \begin{bmatrix} type = hotel \\ location = central \\ price = cheap \end{bmatrix} \qquad R_0 = \begin{bmatrix} name = \\ website = \\ address = \end{bmatrix}$$

Agenda 1

$$\begin{bmatrix} inform(type = hotel) \\ request(name) \\ request(website) \\ request(address) \\ bye() \end{bmatrix}$$

Agenda 2

$$\begin{bmatrix} inform(name) \\ inform(website) \\ inform(address) \\ request(type = hotel) \\ request(location = central) \\ request(price = cheap) \\ bye() \end{bmatrix}$$

Agenda 3

$$\begin{bmatrix} inform(type = hotel) \\ inform(location = central) \\ inform(price = cheap) \\ request(name) \\ request(website) \\ request(address) \\ bye() \end{bmatrix}$$

Given the information need above, how would the agenda be initialized in an agenda-based user simulator?

- ( ) Agenda 1
- ( ) Agenda 2
- (X) Agenda 3

30 Conversational information access (3 points)

Suggest two different ways of exploiting item reviews by a conversational recommender system.

- (2p) Generating preference elicitation questions
- (2p) Improving item recommendations (e.g., matching user preferences against item aspects discussed in reviews)
- (2p) Generating explanations to accompany recommendations
- Answers related to the utilization of sentiment ("sentiment analysis") are only accepted if a reasonable approach is outlined (e.g., for summarizing specific positive and negative aspects mentioned in reviews as explanations). Ideas related to positive/negative sentiment aggregated on the item level (e.g., recommending items with positive reviews) are not accepted.
- The identification of key item features or attributes ("feature extraction") is given only 1p unless it is accompanied by a reasonably specific idea for utilization.
- Answers utilizing only the numerical rating but not the actual review text are not accepted.