Goethe Universität
Institut für Informatik
Texttechnologie

SoSe 2023

NLP-gestützte Data Science

Übung 1

Manuel Stoeckel Prof. Dr. Alexander Mehler

Frist: 01. Juni 2023

Formalia

Die Übungen der Veranstaltung *NLP-gestützte Data Science* dienen der Vertiefung der in der Vorlesung behandelten Themen und einem praktischen Einblick in die diskutierten Probleme und Ihrer Lösungen. Es wird insgesamt **drei** Übungsblätter geben, mit denen Sie bis zu **150** Punkte erreichen können, welche Ihnen bei der Modulabschlussprüfung zu **einem Zehntel** als **Bonuspunkte** angerechnet werden. Ziehen Sie hierzu die entsprechenden Abschnitte Ihrer Prüfungsordnung zu rate.[1]

Allgemein gilt für Abgaben:

Die Abgabe erfolgt im **PDF Format** via OLAT. Bitte stellen Sie sicher, dass **Ihr Name und Ihre Matrikelnummer** auf jeder Abgabe vermerkt sind. Wir empfehlen Ihnen die Verwendung von LaTeX zur Erstellung Ihrer Abgabedokumente. Dazu liegt ein Template in OLAT bereit. Es gelten die gängigen Regeln für Plagiate: stellen Sie sicher, dass Sie Ihre Abgaben selbstständig erstellt haben und etwaige Fremdinhalte entsprechend gekennzeichnet und korrekt zitiert haben.

Für Programmieraufgaben gilt gesondert:

Neben der Abgabe Ihrer Ergebnisse im geforderten Format, ist zusätzlich der gesamte Quellcode ZIP-komprimiert einzureichen. Bitte achten Sie darauf, dass Sie **keine Binaries oder Bibliotheken** mit abgeben (z.B. `virtualenv`, Python Byte-Code, `git`-Repositories, etc.). Der Quellcode ist zu kommentieren. Fremdcode ist entsprechend zu kennzeichnen, auch hier gelten die gängigen Regeln für Plagiate. Wir empfehlen die Verwendung von `conda` zur Umgebungsverwaltung, z.B. mit `miniconda`. Bitte verwenden Sie eine Version von Python ≥ 3.10.

Bei Gruppenabgaben gilt gesondert:

Der individuelle Beitrag jedes Gruppenmitglieds muss aus der Abgabe ersichtlich sein.

[1] §36 Abs. 6 Satz 2 der Ordnung für den Bachelorstudiengang Informatik bzw. §35 Abs. 5 Satz 2 der Ordnung für den Masterstudiengang Informatik, jeweils in der Fassung vom 17. Juni 2019.

Übung 1: Word2Vec **max. 50 P**

In dieser Übung werden wir uns mit dem `word2vec`-Modell (Mikolov, Chen et al., 2013; Mikolov, Sutskever et al., 2013) beschäftigen. Das `word2vec`-Modell wird in zwei Artikeln beschrieben; für die Übung wird aber insbesondere das zweite Paper von Relevanz sein.

1.0 Einarbeitung

Allgemein

- Lesen Sie das Paper „Distributed Representations of Words and Phrases and their Compositionality“, Mikolov, Sutskever et al. (2013).
- Richten Sie Ihre Python Entwicklungsumgebung ein und installieren Sie `PyTorch`[2].
  - Wenn Sie `conda` oder `mamba` verwenden, können Sie die beigelegte `env.yml` Datei verwenden um eine Umgebung zu erstellen.
- Machen Sie sich mit `PyTorch` vertraut.

Code-Template & Daten

Machen Sie sich mit dem gegebenen Code und den gegebenen Daten vertraut:

- Es ist ein Python Modul `nlpds` gegeben, das eine Reihe von Interfaces in `nlpds.abc.ex1` enthält.
  - Diese **müssen** Sie implementieren und dürfen Sie **nicht** verändern.
- Ihre Lösung sollte in `nlpds.submission.ex1` zu finden sein.
- Ihr Code wird automatisch evaluiert und getestet. Ein kleiner Test dieser Art ist bereits in der Aufgabe enthalten.
- Zum Ausführen Ihres Codes sollten Sie eine separate `main.py` Datei im Wurzelverzeichnis anlegen (nicht Teil der Bewertung).

Ordnerstruktur

```
src/
 ├── data/
 │   └── train_enwiki.txt.gz
 ├── nlpds/
 │   ├── __init__.py
 │   ├── abc/
 │   │   └── ex1.py
 │   └── submission/
 │       └── ex1.py
 ├── env.yml
 ├── main.py
 └── test_ex1.py
```

1.1 Textverarbeitung **20 P**

Um ein `word2vec`-Modell zu trainieren, müssen die Trainings-Korpora entsprechend vorverarbeitet werden. Dies umfasst die Tokenisierung sowie das Erstellen der Kontextfenster.

- Implementieren Sie die Klassen:

[2] https://pytorch.org/get-started/locally/

- `ContextWindow` und `TokenizedSentence`, sowie
- `Tokenizer` und `Dataset`.

Hinweise

- Im Gegensatz zu einer Mindestanzahl an Vorkommen im Trainings-Korpus für den Tokenizer, wie im originalen `word2vec`, sollen Sie einen Tokenizer mit einer maximalen Größe implementieren. Dabei werden die häufigsten Token gespeichert und weniger häufige Token verworfen.
- Der Tokenizer soll in der Vorverarbeitung **alle Satzzeichen**[3] von anderen Wörtern einzeln abtrennen.

1.2 Sprachmodellierung **25 P**

- Implementieren Sie die `onehot_vector()` Funktion.
- Implementieren Sie die Skip-Gram-Modelle:
  - `SkipGramSoftMax` und
  - `SkipGramNegativeSampling`.
- Achten Sie darauf auch das Subsampling von häufigen Wörtern zu implementieren.

Hinweis 1

Die in dem Paper im Kontext von *Negative Sampling* erwähnte „unigram distribution“ ist gegeben als:

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} f(w_j)^{3/4}}$$

Hinweis 2

Die in dem Paper angegebene Formel für das *Subsampling* stimmt nicht mit der im originalen `word2vec`-Code verwendeten Formel über ein. Bitte verwenden Sie die folgene Formel, wie sie auch im Code verwendet wird:

$$P(w_i) = \left(\sqrt{\frac{z(w_i)}{t}}\right) \cdot \frac{t}{z(w_i)}$$

Dabei ist t ein Parameter, üblicherweise 0.001, und $z(w_i)$ die *Frequenz* mit der das Wort im Trainingskorpus vorkommt, also die Anzahl seiner Vorkommen geteilt durch die Anzahl aller Wörter.

1.3 Ergebnisse **5 P**

Verwenden Sie Ihren Code und dokumentieren Sie Ihre Ergebnisse (in einer begleitenden PDF).

- Trainieren Sie einen Tokenizer an dem kleinen Beispiel-Korpus (`data/train_enwiki.txt.gz`). Was sind die häufigsten Token?
- Trainieren Sie die Modelle auf dem tokenisierten Korpus. Können Sie die Ergebnisse von Mikolov et al. reproduzieren?
  - Verwenden Sie Standard-Hyperparameter zum Training. Sie können die GPUs auf den Rechnern der RBI nutzen, um Ihr Training zu beschleunigen.
  - Falls das Training zu lange dauert, können Sie es auch frühzeitig unterbrechen.
  - Dokumentieren Sie auch Ihre Zwischenergebnisse.

[3] Siehe: https://docs.python.org/3/library/string.html

Bonus: CBOW **5 P**

In Aufgabe 1.2 war nur das Skip-Gram-Modell zu implementieren.

- Implementieren Sie nun das CBOW-Modell `CbowSoftMax`.

Bonus: Wang2Vec **5 P**

In dem Artikel „Two/too Simple Adaptations of Word2vec for Syntax Problems“ beschreiben Ling et al. (2015) eine Variante des `word2vec`-Modells, die besser für Syntaxprobleme geeignet sein soll.

- Lesen Sie das Paper.
- Implementieren Sie die Änderungen.
- Wiederholen Sie Ihre Experimente und interpretieren Sie etwaige Änderungen in den Ergebnissen.

Literatur

Ling, Wang et al. (2015). „Two/too Simple Adaptations of Word2vec for Syntax Problems“. In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 00155. Denver, Colorado: Association for Computational Linguistics, S. 1299–1304. DOI: 10.3115/v1/n15-1142.

Mikolov, Tomás, Kai Chen et al. (2013). „Efficient Estimation of Word Representations in Vector Space“. In: *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. Hrsg. von Yoshua Bengio und Yann LeCun. arXiv: 1301.3781.

Mikolov, Tomás, Ilya Sutskever et al. (2013). „Distributed Representations of Words and Phrases and their Compositionality“. In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. Hrsg. von Christopher J. C. Burges et al., S. 3111–3119. URL: https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html.