Goethe Universität
Institut für Informatik
Texttechnologie

SoSe 2023

NLP-gestützte Data Science

Übung 1 - Primer[†]

Manuel Stoeckel Kevin Bönisch
Prof. Dr. Alexander Mehler

Frist: 21. Mai 2023, 16:00 Uhr

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] **Primer** geben je 10 Punkte, die auf das darauf folgende Übungsblatt angerechnet werden.

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. Python Byte-Code, `virtualenv`'s, `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`/`mamba` zur Umgebungsverwaltung, z.B. mit `miniforge`. Bitte verwenden Sie eine Version von Python ≥ 3.12.

Bei Gruppenabgaben gilt gesondert:

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

[†]Revision 1 mit Änderungen von 14.05.2024, 11:55.
[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 – Primer: Machine Learning for NLP in PyTorch

In der 1. Übung werden wir uns mit dem `word2vec`-Modell (Mikolov, Chen et al., 2013; Mikolov, Sutskever et al., 2013) beschäftigen, welches in `PyTorch` zu implementieren sein wird. Dieser Primer dient als Vorbereitung und Einführung in Machine & Deep Learning in Python mit `PyTorch`.

Allgemein

- Richten Sie Ihre Python 3.12 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.
  - Ziehen Sie hierzu die Dokumentation[3] und Tutorials[4] (falls nötig) zu Rate.
  - Literaturempfehlungen:
    - *Dive into Deep Learning*, Zhang et al. (2023)
    - *Programming PyTorch for Deep Learning*, Pointer (2019)

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.primer` zu finden sein.
- Ihr Code wird u.A. 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 `src/` anlegen.
- Zum Training eines Sprachklassifikationmodells sind Trainingsdaten für Deutsch und Englisch in drei Splits gegeben: `*_train.txt` & `*_dev.txt` für Training & Hyper-Parameter Tuning und `*_test.txt` für die Evaluation.

Ordnerstruktur

```
env.yml
data/
 └── ex1/
     └── primer/
         ├── deu_dev.txt
         ├── deu_test.txt
         ├── deu_train.txt
         ├── eng_dev.txt
         ├── eng_test.txt
         └── eng_train.txt
```

```
src/
 ├── main.py
 └── nlpds/
     ├── __init__.py
     ├── abc/
     │   └── ex1.py
     ├── submission/
     │   └── ex1.py
     └── tests/
         ├── abstract.py
         ├── unit.py
         └── data/
             ├── a.txt
             └── b.txt
```

[2] https://pytorch.org/get-started/locally/
[3] https://pytorch.org/docs/stable/
[4] https://pytorch.org/tutorials/

1.0 Language Classification — 10 P

In der Vorlesung wurde ein Linearer Klassifikator zur Klassifikation von Textsprachen anhand von Buchstaben-Bi-Grammen vorgestellt (Kapitel 03, S. 18 ff.). In diesem Primer werden wir einen solchen Klassifikator mit Pytorch implementieren und an einem kleinen Testdatensatz trainieren.

Das Modell soll anhand von Feature-Vektoren, welche die Frequenz von Bi-Grammen in einem Text abbilden, eine log-lineare, binäre Klassifikation durchführen. Ausgabe des Modells ist also ein Score, der als Wahrscheinlichkeit für die beiden Klassen interpretiert werden kann. Das Modell selbst soll dabei nur ein einzeles Hidden Layer mit Bias umfassen.

- **Implementieren** Sie die Klassen: `BinaryLanguageClassifier`, `LanguageClassificationDataset` und `BiGramGenerator`.
  - Die Klassen erben jeweils von einer `AbstractBaseClass`[5] – dort finden Sie ggf. einige Hinweise.
- **Dokumentieren** Sie Ihren Code.
  - Achten Sie hier vor allem auf *docstrings*[6] und *type hints*[7]!
  - Inline Kommentare sind nur für besonders komplizierte Code-Abschnitte notwendig.
- **Schreiben** Sie ein Trainingsskript, **trainieren** Sie ein Sprachklassifikationsmodell mit *Stochastic Gradient Descent* und **evaluieren** Sie es.
- **Dokumentieren** und **interpretieren** Sie Ihre Ergebnisse in einer separaten PDF-Datei.
  - Stellen Sie Ihre Ergebnisse (*accuracy*) in einer Tabelle und einer Konfusionsmatrix dar.
  - Verwenden und vergleichen Sie *mindestens* zwei unterschiedliche Vokabulare (= Menge der zu zählenden Bi-Gramme), z.B. zum einen die Kleinbuchstaben a-z und das Leerzeichen.
  - Interpretieren Sie Ihre Ergebnisse, ggf. mit einer Fehleranalyse, z.B. anhand von sinnvollen, fehlerhaft klassifizierten Beispielen aus dem Datensatz oder anhand selbst ausgesuchter Beispiele.
- Laden Sie Ihren Code und die PDF gezippt auf OLAT hoch.
  - Ihre Namen, Mat.-Nr. und studentische E-Mail-Adressen sollten im PDF-Dokument zu finden sein.

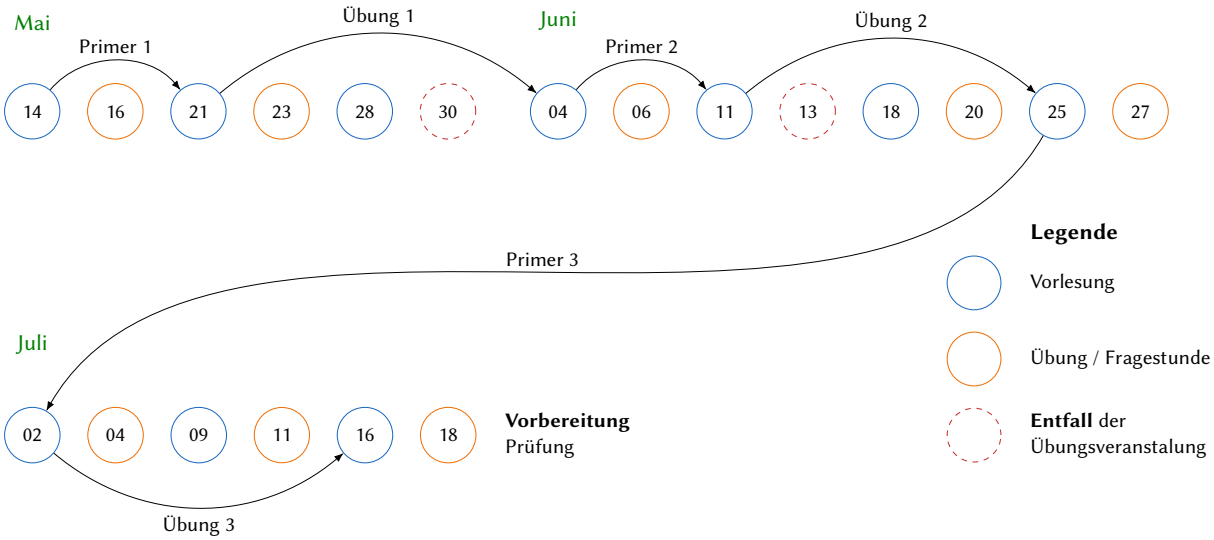
Hinweise

- Wie gehen Sie mit Großbuchstaben und Satzzeichen um?
- Achten Sie darauf, dass Sie keine „falschen" Bi-Gramme durch etwaige Vorverarbeitung hinzufügen! Alle extrahierten Bi-Gramme sollten tatsächlich so im Text vorhanden sein.
- Verwenden Sie in jedem Fall einen `DataLoader`[8], die *Sigmoid* Aktivierungsfunktion und *Binary Cross-Entropy Loss*. Wählen Sie sinnvolle Hyper-Parameter für den *Optimizer*, die *Batch Size* und die Anzahl der Trainingsepochen.
- Die Ausgabe der `.forward()` Funtion des Modells müssen **nicht normalisierte Logits** sein, dh. **ohne** Anwendung der Sigmoid-Funktion.
- Ihr Code wird sowohl mit Unit- als auch mit Integration-Tests evaluiert. Im Code-Template sind einige Unit-Tests enthalten. Sie können diese aus dem `src/` Ordner wie folgt ausführen:

```
python -m unittest nlpds/tests/*.py
```

[5] https://docs.python.org/3/library/abc.html
[6] https://peps.python.org/pep-0257/
[7] Insbesondere https://peps.python.org/pep-0484/, aber auch die neue Python 3.12 Syntax für *generics* und das `type` Statement https://docs.python.org/3/whatsnew/3.12.html#whatsnew312-pep695
[8] https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader

Übungsplan



Literatur

Goldhahn, Dirk, Thomas Eckart und Uwe Quasthoff (2012). „Building Large Monolingual Dictionaries at the Leipzig Corpora Collection: From 100 to 200 Languages“. In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation, LREC 2012, Istanbul, Turkey, May 23-25, 2012*. Hrsg. von Nicoletta Calzolari et al. *Die Trainings- und Testdaten für die Übungsaufgabe wurden dieser Korpussammlung entnommen.* European Language Resources Association (ELRA), S. 759–765. URL: http://www.lrec-conf.org/proceedings/lrec2012/summaries/327.html.

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.

Pointer, Ian (2019). *Programming PyTorch for Deep Learning: Creating and Deploying Deep Learning Applications*. O'Reilly Media. URL: https://ubffm.hds.hebis.de/Record/HEB459200410.

Zhang, Aston et al. (2023). *Dive into Deep Learning*. https://D2L.ai. Cambridge University Press.