Goethe Universität
Institut für Informatik
Texttechnologie

SoSe 2025

NLP-gestützte Data Science

Übung 1

Manuel Schaaf
Prof. Dr. Alexander Mehler

03.06.2025

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. 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 `uv` zur Umgebungsverwaltung. Bitte verwenden Sie eine Version von Python ≥ 3.13.

Gesonderter Hinweis zur Verwendung von KI-Assistenten

Die Verwendung von KI-Assistenten zur Bearbeitung der Übungsaufgaben auf diesem Blatt ist **nicht gestattet**. Gestattet ist jediglich die Verwendung von KI-Assistenten als Formulierungshilfe oder zur Überprüfung von Rechtschreibung und Grammatik – jedoch nur *mit einem entsprechenden Vermerk* am Ende der Abgabe-PDF.

[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: Lineare Klassifikatoren und Vektor-Repräsentationen

In dieser Übung werden wir uns mit dem linearen Modell zur Sprachklassifikation beschäftigen, dass in Kapitel 03 - Propädeutikum I der Vorlesung eingeführt wurde. Sie dient als Vorbereitung und Einführung in Machine & Deep Learning in Python mit `numpy`, `scikit-learn` und `PyTorch`.

Allgemein

- Richten Sie Ihre Python 3.13 Entwicklungsumgebung ein und installieren Sie `numpy`, `scikit-learn` und `PyTorch`[2].
  - Hierzu liegt eine entsprechende `pythonproject.toml` Datei zur Erstellung einer Entwicklungsumgebung mit `uv` bereit.
- Machen Sie sich mit `numpy`, `scikit-learn` und `PyTorch` vertraut.
  - Zu `scikit-learn`: Lineare Klassifikationsmodelle sind im `sklearn.linear_model`[3] Modul verortet.
  - Zu `PyTorch`: ziehen Sie (falls nötig) die Dokumentation[4] und Tutorials[5] 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 `ABC`s in `nlpds.abc.ex1` enthält.
  - Diese **müssen** Sie implementieren und dürfen Sie **nicht** verändern.
- Ihre Lösung sollte in `nlpds.submission.ex1` zu finden sein.
- **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.
- Zum Training eines Sprachklassifikationsmodells sind Trainingsdaten für Deutsch, Englisch und Französisch gegeben. Diese finden Sie im HuggingFace Hub: `Texttechnologylab/leipzig-corpora-collection`.
- Laden Sie zur Abgabe 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.

[2] https://pytorch.org/get-started/locally/
[3] https://scikit-learn.org/stable/api/sklearn.linear_model.html#linear-classifiers
[4] https://pytorch.org/docs/stable/
[5] https://pytorch.org/tutorials/
[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

Linear Language Classification **25 + 5 P**

In der Vorlesung wurde ein Linearer Klassifikator zur Erkennung von Textsprachen anhand von Buchstaben-Bi-Grammen vorgestellt (Kapitel 03, S. 18 ff.). In diesem Aufgabe werden wir einen solchen Klassifikator mit `sklearn` 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 Wert, der als Wahrscheinlichkeit für die beiden Klassen interpretiert werden kann.

1.1 Single-Class Classification **15 P**

- **Implementieren** Sie die Klassen: `BiGramFeaturizer` und `BiGramLanguageClassifier`.
  - Die Klassen erben jeweils von einer `AbstractBaseClass` – dort finden Sie ggf. einige Hinweise.
- **Schreiben** Sie ein Trainingsskript, **trainieren** Sie ein Sprachklassifikationsmodell mit *Logistic Regression* unter Verwendung des 28×28 Vokabulars von Goldberg (2017) und **evaluieren** Sie es.
- **Dokumentieren** und **interpretieren** Sie Ihre Ergebnisse in einer PDF-Datei.
  - Stellen Sie Ihre Ergebnisse (*accuracy*) in einer Tabelle und einer Konfusionsmatrix dar.
  - Interpretieren Sie Ihre Ergebnisse, ggf. mit einer Fehleranalyse anhand ausgesuchter Beispiele.

1.2 Vocabulary Optimization **5 P**

- Entwerfen Sie *optimiertes* Vokabular durch explizite Auswahl der Bi-Gramme.
  - Das Vokabular sollte kleiner als 28×28 sein und dabei nicht zu einer deutlich schlechteren Klassifikation führen.
- Entwerfen Sie einen Algorithmus zur Erstellung eines *minimalen* Vokabulars.
  - Das Vokabular sollte deutlich kleiner als 28×28 sein, aber dennoch eine akzeptable Klassifikationsleistung ermöglichen ($\geq$90%, relativ zu 1.1).
- Wiederholen Sie die Experimente aus 1.1 und vergleichen Sie die Ergebnisse.

1.3 Sprachanpassung **5 P**

- Passen Sie das Vokabular für die Unterscheidung zwischen Deutsch und Französisch an.
  - Welche Änderungen sind ggf. in der Vorverarbeitung notwendig oder sinnvoll?
- Wiederholen Sie die Experimente aus 1.1 und vergleichen Sie die Ergebnisse.

Extra: Multi-Class Classification **5 E**

- Erweitern Sie Ihren Sprachklassifikator auf **drei** Sprachen (Deutsch, Englisch & Französisch).
- Wiederholen Sie die Experimente aus 1.1 (mit drei Sprachen) und vergleichen Sie die Ergebnisse.

Hinweise

- Welches Vokabular wird in Goldberg (2017) verwendet? Wie können Sie mit Großbuchstaben umgehen?
- Welche sprachlichen Besonderheiten gibt es Deutschen, Englischen und Französischen, die Sie für die Vokabular-Bestimmung nutzen könnten?

Word Vector Representations **25 + 5 P**

In der Vorlesung lernen wir das `word2vec` Modell zur Einbettung von Wörtern in einen Vektorraum kennen. In dieser Aufgabe werden wir uns mit einer alternativen Methode zur Wort-Einbettung beschäftigen, die auch die Struktur der Wörter selbst miterfasst. Lesen Sie hierzu das Paper Bojanowski et al. (2017).

1.1 Skip-Gram Model with Negative Sampling **25 P**

`word2vec` und `fastText` haben jeweils zwei Varianten: **CBOW** und **Skip-Gram**. Hier werden wir uns mit der letzteren beschäftigen.

- **Implementieren** Sie die `AbstractNGramEmbedding` Klasse.
  - In der `forward(t,c,n)` Methode ist die *Objective Function* des *Negative Sampling Skip-Gram* Modells zu implementieren. Sie nimmt die drei Tensoren entgegen, welche das N-Gramm-Embedding des Zielworts t, die N-Gramm-Embeddings der Kontextwörter c und die N-Gramm-Embeddings der Negative Samples n enthalten sollen.
  - In den `embed_target(t, *, **)` und `embed_context(c, *, **)` Methoden ist das Embedding eines Tensors von N-Gramm-Indices zu implementieren. Das Embedding eines Wortes entspricht sodann dem *Durchschnitt* seiner N-Gramm-Embeddings.
- **Implementieren** Sie die Vorverarbeitung und das Negative Sampling.
  - Hier gibt es *keine* vorgegebenen Funktionen oder Klassen.
- **Trainieren** Sie ein Embedding-Modell auf Bi- und Tri-Grammen in einer Sprache Ihrer Wahl.
- **Evaluieren** Sie das trainierte Embedding-Modell qualitativ.
  - Haben Vektoren ähnlicher Wörter eine hohe Cosinus-Ähnlichkeit? Orientieren Sie sich hierbei an den Beispielen aus Bojanowski et al. (2017).
- **Dokumentieren** und **interpretieren** Sie Ihre Ergebnisse in einer PDF-Datei.

Extra: Größere N-Gramme? **5 E**

Wenn Sie größere N-Gramme (Quad-Gramme, etc.) verwenden, wächst die Größe Ihres Vokabulars exponentiell an – `fastText` wurde aber mit bis zu 6-Grammen trainiert!

- **Entwickeln** und **erläutern** Sie eine Änderung, mit der Sie auch mit größeren N-Grammen trainieren können.
- **Implementieren** Sie die Änderung in Ihrem Modell.
- **Trainieren** und **evaluieren** Sie die Ergebnisse.
- **Vergleichen** Sie die Ergebnisse mit dem Standardmodell.

Hinweise

- Können Sie mehrere Kontextwörter gleichzeitig einbetten, auch wenn diese unterschiedlich lang sind?
- Die N-Gramme können mit `torch.nn.Embedding` oder `torch.nn.EmbeddingBag` eingebettet werden – welches ist hier die bessere Wahl?
- Die beiden `embed_*` Methoden nehmen jeweils weitere *optionale* Argumente entgegen. Vielleicht können Sie diese sinnvoll verwenden.
- Die Paper definieren die Objective Function idR. als Summe. Können Sie eine einfachere und zugleich effizientere Lösung finden?

Literatur

Bojanowski, Piotr et al. (2017). „Enriching Word Vectors with Subword Information“. In: *Transactions of the Association for Computational Linguistics* 5. Hrsg. von Lillian Lee, Mark Johnson und Kristina Toutanova, S. 135–146. DOI: 10.1162/tacl_a_00051. URL: https://aclanthology.org/Q17-1010/.

Goldberg, Yoav (2017). *Neural network methods in natural language processing*. Morgan & Claypool Publishers.

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.

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.