Deggendorf Institute of Technology
Natural Language Processing Coursework

Professor Patrick GLAUNER

2022 Summer Term

1 Introduction

In parallel to the lectures, this coursework is offered. The aim of this coursework is to deepen and broaden topics taught in class. You will be able to elaborate your understanding by turning theoretical knowledge into code. You will also have the opportunity to become more familiar with the theory by solving assignments. Most of your solutions are not examined directly, but you are strongly encouraged to work on as many problems as time permits. **The written exam may refer to the problems of this coursework!** Questions and solutions can be discussed during the laboratory sessions.

2 Mandatory problem

Complete Problem 7.3 in order to pass the mandatory problem ("Leistungsnachweis"/"Übungsleistung"). In addition to the implementation, you have to write a report that consists of the following parts:

1. Introduction: e.g. goals of your project, why your problem is important to be solved, related work, etc.
2. Methodology: describe the mathematical and conceptual foundations of your model.
3. Experimental results and discussion: e.g. some details on your data set, report metrics (accuracy, precision, recall, F_1) of your model, discuss strengths and limitations of your model, report other interesting findings, etc. (Do not include any code. However, you can discuss in a few sentences some practical implementation issues you encountered.)
4. Conclusions and prospects: summary and what else you could do in order to further improve your model.

Your report should be about 4 pages. Groups of up to 4 students can make a joint submission. **Submit your report and code via iLearn until July 3, 2022.** Your submission must include the names and student IDs of all group members.

3 Exam bonus problem

Successful completion of Problem 8.2 carries a 10% bonus for the exam. In addition to the implementation, you have to write a report. The report structure should be somewhat similar to the one of the mandatory problem in Section 2. Your report should be about 4 pages. Groups of up to 4 students can make a joint submission. **Submit your report and code via iLearn until July 3, 2022.** Your submission must include the names and student IDs of all group members.

4 Remarks

You can choose any programming language to solve the problems. However, Python is the recommended language. We will use various Python packages for NLP and machine learning throughout the coursework.

5 Python primer

Problem 5.1

Familiarize yourself with Python using `intro-to-python.py`. Also familiarize yourself with a Python IDE such as PyCharm. Furthermore, try interactive programming using `python -i` or `ipython`.

6 Foundations

Problem 6.1

Provide 5 stopwords for the following languages: English, French and German.

Problem 6.2

Are your stopwords included in the lists used in `nltk`[1]?

Problem 6.3

Apply the different stemmers of `nltk`[2] to some words of your choice.

7 Text classification

Problem 7.1

Calculate on a piece of paper the probabilities of "CASINO", "CASINO MACAU" and "MAKE MONEY MACAU" being spam using naive Bayes on the following database.

	SPAM	HAM
1	FREE MONEY	WENT TO MACAU
2	FREE CASINO MONEY	WON MONEY MACAU CASINO
3	CASINO EVENT	GREETINGS FROM HOLIDAYS
4		MAKE PROFIT
5		MAKE SCRIPT

Problem 7.2

Familiarize yourself with what accuracy, precision, recall and F_1 mean. There are helpful explanations available online[3].

Problem 7.3

Implement a spam filter that is based on naive Bayes:

- Create your own data set of messages, extract messages from your email box or find a data set online.
- Implement naive Bayes, do not use a library such as `scikit-learn`.
- Report accuracy, precision, recall and F_1 on the test set.

[1]http://www.nltk.org/book/ch02.html
[2]http://www.nltk.org/api/nltk.stem.html
[3]http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html

Problem 7.4
Implement a language classifier for English and German using character 1-grams:

- You can compute the conditional probabilities $P(CHARACTER|LANGUAGE)$ (letter frequencies) or use pre-computed probabilities from an online resource[4].
- You can convert special characters to the characters a-z, e.g. "ä" to "a". However, make sure that $\sum_{i=1}^{26} P(c_1|LANGUAGE) \approx 1$ holds true for both languages.
- Think about how to define $P(English)$ and $P(German)$. Make sure that $P(English)+P(German) \approx 1$ holds true.
- The actual classification part can be implemented in a few lines of Python code without using a library.
- Report accuracy, precision, recall and F_1 on the test set.

Problem 7.5
Train a logistic regression classifier for spam classification. You may have to preprocess a message using one-hot-coding[5] or a count vectorizer[6]. Optimize the regularization parameter.

Problem 7.6
Try different methods, such as stemming or stopword filtering and report the change of accuracy, precision and recall.

Problem 7.7
Answer the following questions in your own words:

1. How does machine learning differ from traditional programming?
2. What is loss, and how do I measure it?
3. How does gradient descent work?
4. How do I determine whether my model is effective?
5. How do I represent my data so that a program can learn from it?

These questions are key to the Google Machine Learning Crash Course[7].

8 Spelling correction

Problem 8.1
Use the code of Peter Norvig's spelling corrector and test it on words of your choice. Does it behave as you expected?

Problem 8.2
Suggest and implement at least two enhancements of your choice. You can get some ideas from the initial post[8].

[4]`http://en.wikipedia.org/wiki/Letter_frequency`
[5]`http://scikit-learn.org/stable/modules/preprocessing.html#encoding-categorical-features`
[6]`http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html`
[7]`http://developers.google.com/machine-learning/crash-course/`
[8]`http://norvig.com/spell-correct.html`

9 Ranking search results

Problem 9.1

Familiarize yourself with how tf-idf works.

Problem 9.2

Run the cosine similarity implementation in Algorithm 1. Is the order of the similarities as you expected?

Algorithm 1 Cosine similarity

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

documents = (
'Interesting people live in Deggendorf for the last 500 years',
'Deggendorf has a river',
'30000 people live in Deggendorf and like Deggendorf',
'The man walked his dog near the river',
'The man walked his dog near the river in Deggendorf'
)

# Transform the documents into the tf-idf matrix
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(documents)
print('Matrix shape: {0}'.format(tfidf_matrix.shape))

# Search query
query = 'Live Deggendorf'
tfidf_search_vector = tfidf_vectorizer.transform([query])

# Compute similarities
print('Similarities: {0}'.format(cosine_similarity(tfidf_search_vector,
    tfidf_matrix)))
```

Problem 9.3

Add the calculation of the actual angles.