LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

CENTRUM FÜR INFORMATIONS- UND SPRACHVERARBEITUNG
STUDIENGANG COMPUTERLINGUISTIK

KLAUSUR ZUR ÜBUNG BACHELORMODUL ,,SYMBOLISCHE PROGRAMMIERSPRACHE“ WS 2020/2021

FLORIAN FINK

KLAUSUR ZUR ÜBUNG AM 18.02.2021

VORNAME:	
NACHNAME:	
MATRIKELNUMMER:	
STUDIENGANG:	B.Sc. Computerlinguistik, B.Sc. Informatik, Magister

Die Klausur zur Übung besteht aus **7 Aufgaben** und umfasst **12 Seiten**. Die Punktzahl ist bei jeder Aufgabe angegeben. Die Bearbeitungsdauer beträgt **45 Minuten**. Bitte überprüfen Sie, ob Sie ein vollständiges Exemplar erhalten haben.

Aufgabe	mögliche Punkte	erreichte Punkte
1. Evaluierung von Klassifikatoren	4	
2. Quizz	4	
3. Git	6	
4. Dokumenten-Retrieval und Unit-Tests	5	
5. K-nearest neighbours	7	
6. WordNet	4	
7. POS Tagging	5	
Summe	35	
Note		

Einwilligungserklärung
Hiermit stimme ich einer Veröffentlichung meines Klausurergebnisses unter Verwendung meiner Matrikelnummer im Internet zu.

Datum: _____ Unterschrift: _____

NAME:

Aufgabe 1 *Evaluierung von Klassifikatoren* [4 Punkte]
Gegeben seien ein Klassifikator für die Klassen *A*, *B* und *C* und zwei Listen, die jeweils die vorhergesagten bzw. tatsächlichen (*A*, *B* und *C*) einer Testmenge enthalten.

(a) Implementieren Sie die Funktion unten, die die Accuracy berechnen soll.

```
# Beispielargumente
# example_y = ["A", "C", "B", "C", "A"]
# example_pred = ["B", "C", "A", "C", "B"]
def accuracy(y, pred)
```

(b) Implementieren Sie die Funktion unten, die die Precision für eine angegebene Klasse berechnen soll.

```
# Beispielargumente
# example_y = ["A", "C", "B", "C", "A"]
# example_pred = ["B", "C", "A", "C", "B"]
# example_c = "B"
def precision(y, pred, c)
```

(2+2 = 4 Punkte)

NAME:

NAME:

Aufgabe 2 *Quizz* [4 Punkte]

Beantworten Sie folgende Fragen.

(a)
```
import os
x = [p for p in os.listdir(".") if p.endswith(".txt")]
```
Was enthält die Variable x?

(b)
```
x = {a: b for a, b in zip(["a", "b"], [1, 2])}
```
Was enthält die Variable x?

(c)
```
import math
def x(y):
    return math.sqrt(sum(x*x for x in y))
print(x([1, 2, 3]))
```
Was wird in der Funktion x berechnet?

(d)
```
class Person:
    def __str__(self):
        return f"{self.fname} {self.name}"
p = Person()
print(p)
```
Warum scheitert obiger Code?

(1+1+1+1 = 4 Punkte)

NAME:

Aufgabe 3 *Git* [6 Punkte]

Sie arbeiten mit mehreren Teammitgliedern an einer Aufgabe und verwenden `git` (mit Gitlab remote) als Versionskontrolle.

(a) Erklären Sie kurz die Funktion der Befehle `git pull`, `git add`, `git push` und `git commit -m 'Lösung für die 2. Hausaufgabe'`.

(b) In welcher Reihenfolge wenden Sie die oben angegebenen Befehle (`git pull`, `git add`, `git push` und `git commit -m 'Lösung für die 2. Hausaufgabe'`) sinnvollerweise an, wenn Sie eigene Änderungen zum Gitlab remote hinzufügen wollen und eines Ihrer Teammitglieder möglicherweise auch Änderungen vorgenommen hat? Begründen Sie die gewählte Reihenfolge.

(2 + 4 = 6 Punkte)

NAME:

Aufgabe 4 *Dokumenten-Retrieval und Unit-Tests* [5 Punkte]
Gegeben sei folgende Implementierung einer Dokumentenklasse.

```
import nltk
class Document:
    def __init__(self, text):
        self.text = text
        self.counts = nltk.FreqDist(nltk.word_tokenize(text))
```

(a) Die Funktion `dot` soll das Skalarprodukt zwischen zwei *Dokumenten* berechnen, indem das Skalarprodukt zwischen ihren Frequenzlisten berechnet wird.

```
# Beispielargument:
# example_d1=Document("first document")
# example_d2=Document("second document")
def dot(d1, d2)
```

(b) Implementieren Sie eine Unit-Testklasse, in der Sie die `dot` Funktion aus Aufgabe (a) testen.

(3+2 = 5 Punkte)

NAME:

NAME:

Aufgabe 5 *K-nearest neighbours* [7 Punkte]

Gegeben sei eine Liste mit den Dokumentkategorien und deren Kosinus-Ähnlichkeiten. Implementieren Sie folgende Funktionen.

(a) Die Funktion `order_nearest_to_farthest` soll die Liste nach der Kosinus-Ähnlichkeit sortieren (vom ähnlichsten Dokument zum am wenigsten ähnlichen).

```
# Beispielargument:
# example_distances=[(0.2,"news"),(0.5,"cars"),(0.7,"sport")]
def order_nearest_to_farthest(distances)
```

(b) Die Funktion `labels_k_closest` soll die Liste der `k` ähnlichsten Kategorien zurückliefern (Sie können davon ausgehen, dass die übergebene Liste bereits entsprechend sortiert ist).

```
# Beispielargument:
# example_distances=[(0.7,"sport"),(0.5,"cars"),(0.2,"news")]
# example_k=2
def label_k_closest(distances, k)
```

(c) Die Funktion `choose_label` soll die häufigste Kategorie aus einer Liste von Kategorien zurückliefern (Sie können davon ausgehen, dass die Kategorien bereits entsprechend sortiert sind). Bedenken Sie bei Ihrer Implementation, dass die häufigste Kategorie nicht immer eindeutig ist. Behandeln Sie auch solche Fälle.

```
# Beispielargument:
# example_labels=["sport","cars"]
def choose_label(labels)
```

(2+2+3 = 7 Punkte)

NAME:

NAME:

Aufgabe 6 *WordNet* [4 Punkte]

Zwei Synsets a und b seien *direkte Kohyponyme*, wenn sie beide ein gemeinsames Hyperonym h (ihr *direktes Hyperonym*) besitzen. So sind z.B die Synsets `green.n.01` und `blue.n.01` direkte Kohyponyme, da sie beide das gemeinsame direkte Hyperonym `chromatic_color.n.01` haben.

Implementieren Sie die folgende Funktion `direct_hypernym`. Die Funktion erwartet zwei Wörter (Strings) und liefert entweder ein gemeinsames (direkte) Hyperonym der beiden Wörter zurück, wenn irgendwelche Sysnsets der beiden Wörter direkte Kohyponyme voneinander sind oder `None`, falls keine Sysnsets der beiden Wörter direkte Kohyponyme voneinander sind.

Bedenken Sie bei der Implementierung, dass einzelne Wörter (Strings) mehrere Synsets haben können und dass auch einzelne Synsets mehrere Hyperonyme haben können. Falls zwei Wörter mehrere direkte Hyperonyme haben, reicht es aus nur ein Hyperonym zurückzuliefern. Vergessen Sie auch die entsprechenden `import`-Ausdrücke nicht.

```
# Example
# print(direct_hypernym("green", "blue"))
# >>> chromatic_color.n.01
# print(direct_hypernym("green", "car"))
# >>> None
def direct_hypernym(w1, w2):
```

NAME:

NAME:

Aufgabe 7 *POS Tagging* [5 Punkte]

Gegeben sei die Hypothese: *„Ein Satz endet niemals mit einer Präposition (preposition)"*. Verwenden Sie NLTK und implementieren Sie ein Python-Programm, mit dem Sie diese Hypothese auf dem Korpus `data/corpus.txt` überprüfen.

Sie können davon ausgehen, dass Sie auf eine Datei `data/corpus.txt` zugreifen können, die ein entsprechendes Korpus enthält. Vergessen Sie auch die entsprechenden `import`-Ausdrücke nicht.