“Analyse und Aufbereitung von Daten”

Assignment 2

Informatik 2 für Biomedical Engineering

Technische Universität Graz

Abstract

Bevor wir mit Daten arbeiten können, muss analysiert werden, wie diese Aufgebaut sind. Dadurch lässt sich vorhersagen, ob und wie gut diverse Algorithmen funktionieren werden und welche Vorbereitungsarbeiten (Preprocessing) notwendig sind. Im Rahmen dieses Assignments bauen wir auf unseren Reader auf und erstellen eine Klasse, welche eine Reihe von Kennwerten errechnet und visualisiert.

---

1. Tasks

Schreiben Sie die Klasse `InfoBme` in die Datei `info_bme.py`. Achten Sie dabei auf die folgenden Spezifikationen:

1.1. Klasse InfoBme(4pt)

1.1.1. Constructor

Der Constructor der Klasse erwartet eine korrekte Instanz eines `info_bme_reader` als Parameter. Diese ist einer der beiden Readern aus dem vergangenen Assignment. Überprüfen Sie, ob das übergebene Objekt tatsächlich dem entspricht. (**Hinweis:** `isinstance()` auf die Mutterklasse ist ausreichend). Speichern Sie den Reader als Member Variable. Legen Sie zusätzlich eine Variable namens `self.read_data` an und weisen Sie ihr als Wert die `read_data` Funktion des readers zu. Damit kann dann Ihr Objekt der InfoBme Klasse die Funktion `read_data()` des Readers verwenden.

1.1.2. Properties

`X:` Zeigt auf die Property `X` des Readers.

`y:` Zeigt auf die Property `y` des Readers.

1.2. (public) Funktionen

1.2.1. get_ class(label) - (1pt)

Diese Funktion liefert die Indizes aller Samples mit dem gewünschten Label.

Parameter `label`: Das Label der gewünschen Klasse. Überprüfen Sie, ob dies ein gültiges Label ist (in y mindestens einmal vorkommt).

Return (numpy array): Die indizes der Samples, die zu der gewünschten Klasse gehören.

1.2.2. calc_ balance(indices) - (1pt)

Diese Funktion berechnet, wie häufig jede Klasse(Label) in den gewünschten Indizes vorkommt.

Parameter `indices` (list, numpy array, str): Die Indizes der Samples für die die Balance berechnet werden soll. Wenn der String `'all'` übergeben wird, berechnen Sie dies über alle Samples.

Return (dict): Dict aus: key = Label/Klasse, value = Häufigkeit

1.2.3. calc_means(indices, plot=False, plot_name=None) - (1pt)

Diese Funktion berechnet die durchschnittlichen Feature-Werte der Samples (mit den gewünschten Indizes). Optional kann dies auch visualisiert werden. Durchscnittwerte sind dabei immer über alle gewünschen Samples zu Rechnen. Sie erstellen also ein Durchschnittsbild. Der erste Pixel dieses Bildes ist der durchschnittliche Werte aus 'erster Pixel aus Sample 1', erster Pixel aus Sample 2', 'erster Pixel aus Sample 3', usw.. Der Algorithmus dieser Funktion ist das arithmetische Mittel (mean). Zum plotten der Daten verwenden Sie wieder `imshow()`[1] (gleich wie die plot Funktion des Readers). **Hinweis:** Holen Sie sich zum Testen alle Inzides einer Klasse (via `get_class()` und berechnen Sie deren Durchschnitt.

Parameter `indices` (list, numpy array): Die Indizes der Samples aus welchen der Durschnitt berechnet werden soll.

(KW) Parameter `plot` (bool): Definiert, ob die Daten visualisiert werden sollen.

(KW) Parameter `plot_name` (str): Wenn `'plot is True'` dann definiert dieser Parameter den Pfad und Namen der zu schreibenden Datei. **Hinweis:** Raisen Sie eine Exception wenn der Benutzer die Plots haben will aber keinen Namen angibt.

Return (numpy array): Die Liste an Durchschnittswerten.

[1] https://matplotlib.org/api/_as_gen/matplotlib.pyplot.imshow.html

1.2.4. calc_hist(indices, plot=False, plot_name=None) - (1pt)

Berechnen Sie das Histogramm aus einen gewünschten Set von Samples. Zählen Sie dafür, wie oft jeder Wert in den Samples vorkommt. Dadurch erkennen Sie, ob die samples viel oder wenig Platz (am Bild) benötigen. Plotten Sie die Daten (sofern gewünscht) als Histogramm. Da wir konkrete Werte haben, bietet sich die Funktion `fill_between()`[2] mit `step=True` an.

Parameter `indices` (list,numpy array): Die Indizes der Samples.

(KW) Parameter `plot` (bool): Definiert, ob die Daten visualisiert werden sollen.

(KW) Parameter `plot_name` (str): Wenn `'plot is True'` dann definiert dieser Parameter den Pfad und Namen der zu schreibenden Datei. **Hinweis:** Raisen Sie eine Exception wenn der Benutzer die Plots haben will aber keinen Namen angibt.

Return (dict): Dict aus: key = Wert, value = Häufigkeit

1.3. calc_bbox(indices, plot=False, plot_name=None) - (2pt)

Berechnet die Bounding Box der Samples. Diese ist definiert durch 2 Punkte (Zeilen und Spaltenindex). P_0, links und darüber sind alle Werte aller Samples 0, und P_0, rechts und darunter sind alle Werte aller Samples 0. Beachten Sie dabei, dass Sie hier die Properties des Readers `num_rows` und num_cols eine Rolle spielen.

Parameter `indices` (list, numpy array, str): Die Indizes der Samples für die die Bounding Box berechnet werden soll. Wenn der String `'all'` übergeben wird, berechnen Sie dies über alle Samples.

Return (tuple): `((Row` P_0 `, Col` P_0`), (Row` P_1`, Col` P_1`))`

[2] https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.fill_between.html

1.3.1. calc_cosine_similarity(a, b, plot=False, plot_name=None) - (2pt)

Berechnen Sie die Cosinus Ähnlichkeit zweier Samples (a,b). Diese ist folgendermaßen Definiert:

$$similarity = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}} \tag{1}$$

a_i enspricht dem Wert mit dem Index i aus der Liste der Werte von Sample a. Die Cosinus-Ähnlichkeit ist stets zwichen 0 und 1.

Wenn der Benutzer diesen Vergleich visualisiert haben will, erstellen Sie einen simplen Scatterplot[3]. (Verwenden Sie die Werte von `a` als Parameter `x` und die Werte von `b` als Parameter `y`). **Hinweis:** Berechnen Sie die Durchschnittswerte zweier Klassen und berechnen Sie davon die Ähnlichkeit.

Parameter `a` (numpy array) Sample a

Parameter `b` (numpy array) Sample b

(KW) Parameter `plot` (bool): Definiert, ob die Daten visualisiert werden sollen.

(KW) Parameter `plot_name` (str): Wenn `'plot is True'` dann definiert dieser Parameter den Pfad und Namen der zu schreibenden Datei. **Hinweis:** Raisen Sie eine Exception wenn der Benutzer die Plots haben will aber keinen Namen angibt.

Return (float): Die Cosine Similarity der beiden Samples.

[3]https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html

1.3.2. generate_strat_splits(size) - (3pt)

Um in Zukunft klassifizieren zu können, müssen wir die Samples in Teile aufteilen. Jeweils in *Training* und *Verification* sets. Dabei ist es wichtig, dass in jedem Set jede Klasse gleich oft vorkommt und kein Sample in zwei Splits vorkommt. Schreiben Sie einen Generator, der pro yield die Indizes eines Sets zurückgibt.

Beispiel: `size=`0.25
Daten (Labels): $[4, 2, 1, 3, 1, 4, 1, 2, 3, 4, 3, 4, 2, 1, 2, 3]$

- Indizes Split 1: $[0, 1, 2, 3]$
- Indizes Split 2: $[4, 5, 7, 8]$
- Indizes Split 3: $[6, 9, 10, 12]$
- Indizes Split 4: $[11, 13, 14, 15]$

Parameter `size` (float) *size∗num_samples* Anteil der Samples pro Split. Runden Sie dabei auf. **Hinweis:** Das letzte Set ist im Regelfall wegen den Rundungen kleiner.

Return (numpy array): Indizes der Samples aus dem aktuellen Split. **Hinweis:** Generatoren retournieren mit dem Befehlt `yield` anstatt von `return`.

2. Beschränkungen

- Fügen Sie keine nutzlosen Komponenten Ihrer Abgabe hinzu.
- Sofern möglich, verwenden Sie eingebaute Funktionen.
- Importieren Sie keine extra packages die Sie nicht verwenden.
- Alle Kommandozeilenparameter müssen optional sein.

3. Datei Header

All Ihre Quelldateien in Ihrer Abgabe müssen gleich zu Beginn einen Kommentar mit folgenden Informationen enthalten:

- Author: – Ihr Name
- MatNr: – Ihre Matrikelnummer
- Description: – Generelle Beschreibung der Datei
- Comments: – Kommentare, Erklärungen, usw

Bitte einfach den folgenden Code in Ihre Dateien kopieren und den Inhalt anpassen. Je nach PDF-Reader müssen Sie eventuell die Leerzeichen/Einrückungen per Hand anpassen.

Beispielheader:

```
##############################################################################
# Author:      Patrick Kasper
# MatNr:       0730294
# Description: The main file. Assignment only has 1 file...
# Comments:    This is the example comment. I just made it a bit
#              longer so it spans across multiple lines.
##############################################################################
```

4. Coding Standard

Für diese Lehrveranstaltung orientieren Sie sich am offiziellen PEP 8 Standard[4]. Dieser Beschreibt grundsätzliche Formalitäten im Bezug auf Ihren Code. Folgendes ist besonders zu Beachten:

Sprache. Code schreibt man in Englisch. Im internationalen Zeitalter ist es notwendig, dass auch jemand am anderen Ende der Welt verstehen kann, was Sie programmiert haben. Ihr gesamter Quellcode muss daher auf Englisch geschrieben sein. Dies betrifft sowohl die Kommentare also auch Variablennamen und Ähnliches.

[4] `https://www.python.org/dev/peps/pep-0008/`

Leerzeichen statt Tabulatoren. Python basiert auf Einrückungen, anstatt auf geschwungenen Klammern . Theoretisch gibt es die Möglichkeit, Leerzeichen (spaces) oder Tabulatoren (tabs) zu verwenden. PEP8 schreibt aber, 4 Leerzeichen als Einrückungen vor. Die meisten Python Programmierumgebungen werden automatisch 4 Leerzeichen einfügen, wenn Sie auf die Tabulator-Taste drücken.

Sprechende Namen. Verwenden Sie kurze, aber sprechende Namen für Ihre Variablen, Funktionen, (und Ähnliches). Es muss eindeutig aus dem Namen hervorgehen, was die Aufgabe des Elements ist. Für simple Iterationen kann ein einfaches i ausreichend sein, aber dies kann schnell zu Chaos führen. Sollten Sie Variablen haben, die keine Aufgabe haben und nicht verwendet werden, schreibt PEP 8 vor, einen einfachen Unterstrich (_) zu verwenden. Sollten Sie sich unsicher sein, dann beschreiben Sie ihre Variablen (und Namen) in einem Codekommentar.

120 **Zeichen Zeilenlänge.** PEP8 sieht Zeilenlängen von maximal 79 Zeichen vor. Ein anderes, etwas großzügigeres Limit, welches sich in der Szene etabliert hat, sieht eine Länge von 120 Zeichen vor. In dieser LV verwenden wir daher das erweiterte Limit. Bitte achten Sie darauf, dass keine Zeile in Ihrem Code diese Länge von 120 Zeichen überschreitet (gilt auch für Kommentare). *Hinweis:* Zeilenlänge inkludiert die Einrückungen mittels Leerzeichen!

5. Automatisierte Tests

Ihr Programm wird mit dem folgenden Befehl ausgeführt:

```
>python assignment_2.py
```

Diese Datei wird vom Testsystem eingebunden und stammt nicht aus Ihrer Abgabe. Vergewissern Sie sich, dass Ihre Abgabe allen Beschränkungen, die in dieser Angabe erwähnt sind, konform ist.

6. Abgabe

6.1. Deadline

4. Mai 2018 um 23:59:59.

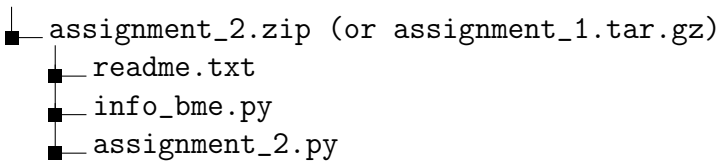
Eine Spätabgabe ist nicht vorgesehen. Ausnahme bilden hier Notfälle. Sollte das Abgabesystem nicht online sein, verlängert sich die Deadline automatisch um 24 Stunden. Sollten Sie, aus diversen Gründen, nicht in der Lage sein, Ihre Abgabe hochzuladen, kontaktieren Sie Ihren Tutor *VOR* der Deadline. (*Hinweis:* Urlaub oder Ähnliches wird nicht als Grund akzeptiert!)

6.2. Hochladen der Abgaben

Assignments werden stets als Archive abgegeben. Erlaubt sind hier die Formate *.zip*, und *.tar.gz*.
Zusätzlich zu ihren Quelldateien, soll Ihre Abgabe auf eine Datei namens *readme.txt* beinhalten. Das Vorhandensein der Datei ist Pflicht, ihr Inhalt aber optional. Sie soll folgenden Inhalt haben: (i) Die Zeit, die Sie benötigt haben, um die Aufgabenstellung zu absolvieren. (ii) Feedback, wo Sie Probleme hatten.
Abgaben erfolgen auf der Palme Website. Bitte prüfen Sie vor der Abgabe diese Kriterien:

- Datei- und Ordnerstruktur (siehe unten)
- Kommentarheader in jeder Quelldatei
- Coding Standard

6.3. Struktur der Abgabe



Wichtig: Geben Sie das Reader Package ***NICHT*** erneut mit ab!