“Handschrift-Klassifikation mit 1NN”

Assignment 3

Informatik 2 für Biomedical Engineering

Technische Universität Graz

Abstract

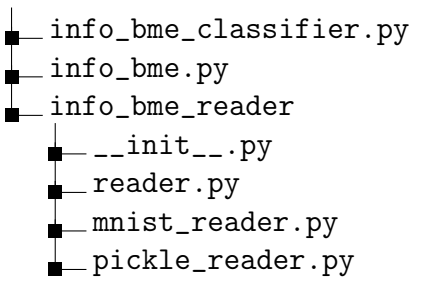
Nachdem nun alle Vorbereitungen abgeschlossen sind, widmen Sie sich in diesem Assignment der finalen Aufgabe, die handgeschriebenen Zahlen mithilfe eines simplen Algorithmus zu klassifizieren. Um diese Label Zuordnungen zu evaluieren, verwenden Sie etablierte Metriken. Schlussendlich kommen Sie in diesem Assignment auch mit Aspekten in Kontakt, die für teamorientierte Softwareentwicklung von hoher Bedeutung sind.

---

1. Tasks

1.1. Bisherige Abgaben

Der Code Ihrer letzten Abgaben soll im selben Verzeichnis verfügbar sind. Hier ist die Struktur Ihres Projektes:



Wichtig: Geben Sie Komponenten Ihrer letzten Abgaben nicht erneut mit ab!

1.2. Klasse InfoBmeClassifier

Erstellen Sie die Datei `info_bme_classifier.py` und implementieren Sie dort die folgende Klasse `InfoBmeClassifier`.

Wichtig: Packages wie *SciPy* oder *sklearn* sind in diesem Assignment NICHT zugelassen. Sie können deren Funktionalität lediglich verwenden, um Ihre Ergebnisse zu Testen und zu Vergleichen.

1.2.1. Constructor

Signatur `__init__(self, info_bme)`

Parameter `info_bme`**:** Eine Instanz der InfoBme Klasse aus Assignment 2. Verwenden Sie dies immer, wenn Sie die Samples oder Labels benötigen.

1.2.2. Methode classify (3pt)

Diese Methode soll die Klassifizierung durchführen. Erstellen Sie Test Sets (mithilfe der bereits implementierten Funktion `generate_strat_splits` aus Assignment 2) und rufen Sie für jedes Set den `nearest_neighbor` Algorithmus auf. Fügen Sie anschließend die Ergebnisse ihrer Klassifikation (Rückgabe des k-NN Algorithmus) wieder zusammen.

Die Rückgabe der Methode ist ein Tuple bestehend aus den zusammengefügten Labels ihrer Klassifikation (`y_pred`) und den tatsächlichen Labels

(`y_true`) (so wie sie bisher in `info_bme.y` gespeichert waren). **Wichtig:** Achten Sie darauf, dass (`y_pred`) und (`y_true`) die gleiche Reihenfolge haben!

Die Parameter `k` und `sample_length` können hier ignoriert (also einfach mit den hier angegebenen defaults an den Klassifizierungsalgorithmus weitergegeben) werden.

Wenn ein Plot gewünscht ist (durch Parameter definiert), erstellen Sie eine Heatmap mit den Klassen aus `y_pred` auf der x-Achse und `y_true` auf der y-Achse. Die Werte der Heatmap beschreiben dann zum Beispiel, wie die Samples der Klasse 1 (also Label 1 in `y_true`) klassifiziert wurden (also anhand der Labels in `y_pred`). **Hinweis:** Wenn Sie in Ihrer Heatmap keine Zahlen für jeden Eintrag haben, verwenden Sie log-scaling oder eliminieren Sie alle korrekt klassifizierten Samples im Plot. Ansonsten wird diese Achse sehr stark dominant sein, und Sie können keine fehlerhaft klassifizierten Samples erkennen!

Beispiel für eine solche Heatmap (auch bekannt als *Confusion Matrix*):

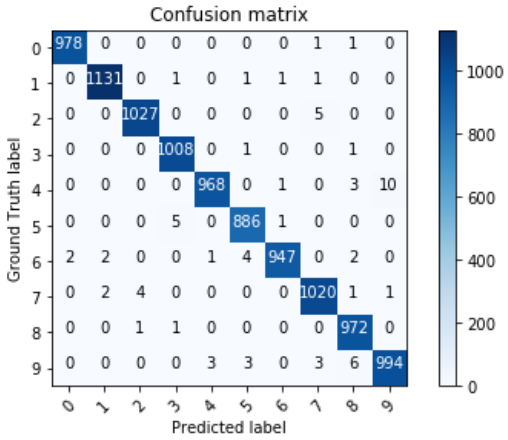


Image taken from Quora: https://www.quora.com/Classification-machine-learning-What-is-confusion-matrix

Signatur `classify(self, split_size=0.2, k=1, sample_length=1.0, plot_name=None)`

Parameter `split_size` (float): Die Größe der einzelnen Splits für ihr k-Folding

Parameter `k` (int): Parameter für den Aufruf von nearest_neighbor.

Parameter `sample_length` (float): Parameter für den Aufruf von nearest_neighbor.

(KW) Parameter `plot_name` (str): Wenn angeben, definiert dieser Parameter den Pfad und Namen, unter dem der Plot gespeichert werden soll. Wird der Parameter nicht übergeben (oder ist `None`), soll auch kein Plot erstellt werden.

Return (tuple): y_pred (Labels Ihrer Klassifikation), y_true (tatsächliche Labels) (y_pred und y_true als numpy-Arrays)

1.2.3. Methode nearest_neighbor (3pt)

Diese Methode implementiert einen *k Nearest Neighbor* Algorithmus. Als Parameter werden die Indizes der zu klassifizierenden Samples übergeben. Für jedes Sample der Test Indizes suchen Sie sich das nächst ähnlichste Sample aus dem Rest (die *__nicht__* im Set der übergebenen Indizes sind) und speichern Sie das Label des ähnlichsten Samples. Verwenden Sie zum Berechnen der Ähnlichkeiten die Cosine Similarity, die Sie bereits in Assignment 2 implementiert haben. Die Parameter `k` und `sample_length` sind nur für die Bonus Tasks notwendig und können ignoriert werden.

Die Rückgabe der Methode ist ein Numpy Array mit den Labels aus Ihrer Prediction.

Signatur `nearest_neighbor(self, test_indices, k, sample_length)`

Parameter `test_indices` (list/np.ndarray): Die Indizes des aktuellen Test Sets.

Parameter `k` (int): Siehe Bonus Task 2.1.

Parameter `sample_length` (float): Siehe Bonus Task 2.2.

Return (np.ndarray): y_pred (Labels ihrer Klassifikation)

1.2.4. Methode precision (1pt)

Diese Methode berechnet den Precision Score einer Klassifikation. Dieser ist folgendermaßen definiert:

$$P = \frac{T_p}{T_p + F_p} \tag{1}$$

Für jede Klasse berechnen Sie folgendes:

- T_p: True Positives, Die Anzahl der Korrekt Klassifizierten Samples
- F_p: False Positives, Anzahl der Samples, die Sie fälschlicherweise mit diesem Label klassifiziert haben.

Um den globalen Precision Score zu ermitteln, berechnen Sie das gewichtete Mittel aus den Precision Scores der einzelnen Klassen. Die jeweiligen Gewichte ergeben Sich aus der Anzahl der Samples pro Klasse (basierend auf den tatsächlichen Labels).

Signatur `precision(self, y_true, y_pred)`

Parameter `y_true` **(list/np.ndarray):** Die tatsächlichen Labels.

Parameter `y_pred` **(list/np.ndarray):** Die Labels, die Ihre Klassifikation zurücklieferte.

Return (tuple): global_precision_score, Dict aus *Klasse: precision score der Klasse*

1.2.5. Methode recall (1pt)

Diese Methode berechnet den Recall Score einer Klassifikation. Dieser ist folgendermaßen definiert:

$$R = \frac{T_p}{T_p + F_n} \tag{2}$$

Für jede Klasse berechnen Sie folgendes:

- T_p: True Positives, Die Anzahl der Korrekt Klassifizierten Samples
- F_n: False Negatives, Anzahl der Samples, eigentlich zu dieser Klasse gehören, Sie aber als etwas anderes Klassifiziert haben.

Um den globalen Recall Score zu ermitteln, berechnen Sie das gewichtete Mittel aus den Recall Scores der einzelnen Klassen. Die jeweiligen Gewichte ergeben Sich aus der Anzahl der Samples pro Klasse (basierend auf den tatsächlichen Labels).

Signatur `recall(self, y_true, y_pred)`

Parameter `y_true` (list/np.ndarray): Die tatsächlichen Labels.

Parameter `y_pred` (list/np.ndarray): Die Labels, die Ihre Klassifikation zurückliefert.

Return (tuple): global_recall_score, Dict aus *Klasse: recall score der Klasse*

1.2.6. Methode f1_score (1pt)

Diese Methode berechnet den F1 Score einer Klassifikation. Dieser ist folgendermaßen definiert:

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{3}$$

Hinweis: Verwenden Sie Ihre Methoden für precision und recall. Dann brauchen Sie sich nicht mehr um die korrekten Gewichtungen kümmern.

Signatur `f1_score(self, y_true, y_pred)`

Parameter `y_true` (list/np.ndarray): Die tatsächlichen Labels.

Parameter `y_pred` (list/np.ndarray): Die Labels, die Ihre Klassifikation zurückliefert.

Return (tuple): global_f1_score, Dict aus *Klasse: F1 score der Klasse*

1.3. Dokumentation (4pt)

Dokumentieren Sie Ihren Code sauber. Orientieren Sie sich dabei am PEP 257 Standard[1]. Es muss für jede Funktion ein Docstring angelegt werden, der die Funktion, ihre Parameter und ihre Rückgabewerte beschreibt.

1.4. SVN (3pt)

Verwenden Sie zum Zusammenarbeiten mit Ihren Gruppenkollegen ein Subversion Project (SVN) im TugOnline. Fügen Sie Ihren Tutor/Ihre Tutorin als Reader zum Projekt hinzu. Eine Anleitung, wie Sie diese Projekte anlegen und verwalten, finden Sie auf der SVN Infoseite der TU Graz[2].

Wichtig: Achten Sie bei Ihren Commits auf saubere Commit Messages.

[1] https://www.python.org/dev/peps/pep-0257/
[2] https://svn.tugraz.at/

1.5. Report (3pt)

Beschreiben Sie die Ergebnisse Ihrer Klassifikation. Was hat funktioniert? Welche Zahlen wurden nicht erfolgreich klassifiziert? Versuchen Sie zu erklären, warum. Verwenden Sie hier die Plot-Funktionalitäten, die Sie in diesem und in den vergangenen Assignments implementiert hatten!

2. Bonustasks

2.1. KNN (4pt)

Erweitern Sie die Funktion `nearest_neighbor`. Anstatt nur das nächst ähnliche Sample als Basis für Ihre Klassifikation zu verwenden, betrachten Sie die k-ähnlichsten Samples (bei $k = 1$ ergibt sich also keine Änderung). Verwenden Sie dafür den bereits implementierten, entsprechenden Parameter (`k`). Danach gibt es unterschiedliche Strategien, das finale Label zu wählen. Zum Beispiel ein majority vote, bei dem Sie einfach die Klasse zuweisen, die in den k-ähnlichsten Samples am häufigsten ist. Oder Sie gewichten die Auswahl anhand des Rankings (also z.B.: das meist ähnliche Sample ist wichtiger als das 5. ähnlichste).

Experimentieren Sie mit unterschiedlichen Werten für k und beschreiben Sie Ihre Erkenntnisse im Report.

2.2. Reduzierte Features (6pt)

Anstatt die gesamten Features zu verwenden, beschränken Sie sich nur auf einen Bruchteil (definiert durch den Parameter `sample_length`). Zum Beispiel: Standardmäßig haben alle Samples 784 Features (Pixel). Mit $sample\_length = 0.5$ verwenden Sie nur die ersten 392.

Reduzieren Sie schrittweise die Anzahl der Features und beobachten Sie die Performance der Klassifikation (precision/recall/f1). Beschreiben Sie Ihre Erkenntnisse im Report.

3. 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.

4. 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.
##############################################################################
```

5. Coding Standard

Für diese Lehrveranstaltung orientieren Sie sich am offiziellen PEP 8 Standard[3]. 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.

[3] `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!

6. Automatisierte Tests

Ihr Programm wird mit dem folgenden Befehl ausgeführt:

```
>python assignment_3.py
```

Diese Datei (assignment_3.py) wird vom Testsystem eingebunden und stammt nicht aus Ihrer Abgabe. Nutzen Sie sie, um Ihren Code zu testen, oder um eventuelle Visualisierungen für Ihren Report zu erstellen. Vergewissern Sie sich, dass Ihre Abgabe allen Beschränkungen, die in dieser Angabe erwähnt sind, konform ist.

7. Abgabe

7.1. Deadline

8. Juni 2018 um 23:59:59.

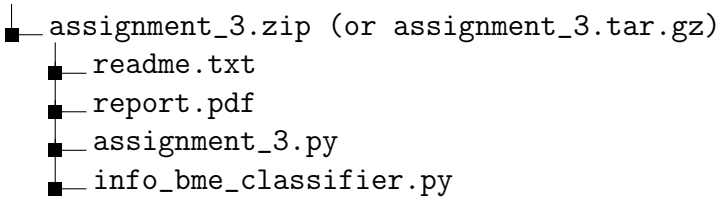
Eine Spätabgabe ist nicht vorgesehen. Ausnahme bilden hier Notfälle. Sollte das Abgabesystem nicht online sein, verlängert sich die Deadline automatisch um 24 Stunden. Sollten Sie, aus diversen Gründen, nicht in der Lage sein, Ihre Abgabe hochzuladen, kontaktieren Sie Ihren Tutor *VOR* der Deadline.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

7.2. 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

7.3. Struktur der Abgabe



Wichtig: Bitte geben Sie die Komponenten vergangener Assignments ***NICHT*** erneut mit ab!