Hausarbeit

University of Applied Science - Online

Study-branch: Data Science

<THIS IS THE TITLE TO BE ADAPTED>

Georg Grunsky

Matrikelnummer: 010101001

Straße 10

realized in Tulln/Österreich

Advisor: <Advisor>

Realized with input of the Parameter Generator: <Signature-Hash> $\Omega \longrightarrow R^2$

Delivery date: 1.1.1970

Contents

I List of Figures

II List of Tables

1 Einleitung

1.1 Die Aufgabenstellung

Im Rahmen dieser Ausarbeitung soll ein Python-Programm umgesetzt werden, das "verwaschene" Trainingsdaten korrekt auf vier idealtypische Funktionen abbildet. Hierfür stehen 50 idealtypische Funktionen zur Auswahl. Maßgeblich für die Auswahl ist hierbei, gemäß Vorgabe, die kleinste Summe der quadratischen Abweichungen. Anschließend soll das gleiche Programm die Punkte eines Testdatensatzes, anhand eines definierten minimalen, statistischen Abstandswertes zu den gewählten idealtypischen Funktionen, klassifizieren und jeweils der entsprechenden Funktion zuordnen. Fokus der vorliegenden Aufgabenstellung, sowie dieser Arbeit war jedoch primär die Erlernung der Programmiersprache Python und nicht die ausschließliche Erfüllung der beschriebenen mathematischen Aspekte. Diese, isoliert betrachtet, hätte ebenso in Form eines einfachen Jupyter-Notebooks erarbeitet werden können.

1.2 Installationshinweise

Das Programm wurde vom Autor mit dem Namen "functionfinder" betitelt und steht auf GitHub unter dem Link

`https://github.com/GGProjects/DLMDWPMP01`

zum Download zur Verfügung. Für die Installation bietet es sich an, lokal eine virtuelle Pythonumgebung anzulegen. Anschließend kann nach Wechsel in das Modulverzeichnis, dieses mit dem Befehl

"pip install -e ."

installiert werden. Mit der Installation wird die Verfügbarkeit der benötigten Dateien überprüft und UnitTests zur Sicherstellung der Funktionalität durchgeführt. Benötigte Pakete werden gegebenenfalls in der virtuellen Umgebung nachinstalliert. Im Rahmen der Installation wird außerdem ein Logfile mit der Bezeichnung "setuplog..." im Verzeichnis output/logs angelegt. Nach erfolgreicher Installation, kann das Programm aus der Konsole mit dem einfachen Aufruf "ff" ausgeführt werden.

1.3 Konfigurationshinweise

Über Änderungen in der Datei functionfinder/config.py können, falls benötigt, Verzeichnispfade, Log-Einstellungen, Bewertungsfunktionen, sowie auch die Pfade zu den vorliegenden Datendateien angepasst werden.

Letzteres ermöglicht eine einfache Änderung der verwendeten Trainings-, Funktions- und Testdaten, um das Programm auch mit anderen als den, im Rahmen der Aufgabenstellung, bereitgestellten Daten auszuführen.

Eine Anpassung der Bewertungsfunktionen bietet des weiteren die Möglichkeit, die zur Bewertung der Trainingsdaten herangezogene Funktion (der Standardwert ist hierfür die kleinste Summe der quadratischen Abweichungen) sowie den Faktor für den errechneten Fehlerwert, zur Klassifikation der Testdaten (der Standardwert ist, gemäß Aufgabenstellung, die Wurzel aus Zwei) zu verändern.

1.4 Struktur der Hausarbeit

Im Hauptteil der vorliegenden Hausarbeit wird zuerst auf die bereitgestellten Datensätze eingegangen und wie diese, im Rahmen der Aufgabenstellung, verarbeitet wurden. Anschließend wird in einem weiteren Unterkapitel die Auswahl an idealtypischen Funktionen anhand der vorliegenden Trainingsdaten beschrieben und dargestellt. Das dritte Unterkapitel diskutiert die Klassifikation der Testdaten mithilfe der ausgewählten idealtypischen Funktionen. Hierbei wird auch auf die automatisierte Ausgabe zur Evaluierung nicht erfolgreich klassifizierter Testdaten eingegangen. In der Zusammenfassung werden, anhand der dargestellten Ergebnisse, mögliche Anpassungen des Programms zur Verarbeitung größerer Datenmengen angesprochen und als potentielles Thema für eine weitere Hausarbeit zur Diskussion gestellt.

2 Hauptteil

2.1 Bereitgestellte Datensätze

Im Zuge der Aufgabenstellung wurden, zur Bearbeitung dieser, drei Datensätze angefertigt und in Form unten aufgelisteter CSV-Dateien bereitgestellt.

- train.csv
- ideal.csv
- test.csv

Erstere beinhaltet eine Tabelle von vier "verwaschenen" Funktionen in den Spalten "y1" bis "y4" mit jeweils 400 Datenpunkten. Die folgende Tabelle zeigt die exemplarische Struktur des Trainingsdatensatzes.

x	y1	y2	y3	y4
-20.0	100.216064	-19.757296	0.3461139	19.776287
-19.9	99.894684	-19.70282	0.61786354	19.789793
-19.8	99.397385	-19.564255	0.1743704	19.441765
-19.7	98.24446	-19.858267	0.7310719	19.869267
-19.6	97.926956	-19.825966	0.27302822	19.864285

Table 1: Exemplarischer Auszug der Datei train.csv

Eine Visualisierung der Trainingsdaten lässt bereits optische Rückschlüsse auf passende idealtypische Funktionen zu.

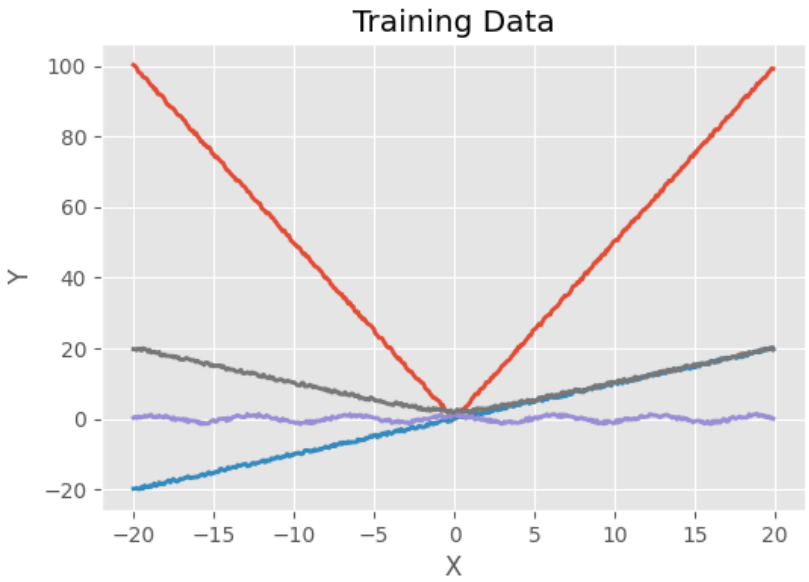

Figure 1: Darstellung der bereitgestellten Trainingsdatensätze **?**

Die Datei ideal.csv beinhaltet 50, nicht "verwaschene", idealtypische Funktionen in den Spalten "y1" bis "y50". Wie, im Detail im folgenden Unterkapitel beschrieben, sollen diese Funktionen durch das Programm mit dem

Trainingsdatensatz verglichen werden und daraus die passendsten vier Funktionen gewählt werden. Die Datei weist, bis auf die Anzahl der Spalten, eine ähnliche Struktur wie der Trainingsdatensatz auf und besteht ebenfalls je Spalte aus 400 Datenpunkten. Auf die Darstellung eines exemplarischen Auszuges wird daher an dieser Stelle verzichtet.

Im Gegenzug zu den oben dargestellten beiden Datensätzen besteht der Testdatensatz nur aus 100 einzelnen Datenpunkten. Diese werden in einer Spalte "y" der Datei test.csv zusammengefasst und sind nicht als zusammengehörige Funktion zu verstehen.

x	y
4.9	4.4963365
-4.7	34.25082
7.2	6.6985793
17.9	17.754583
-6.5	81.21402

Table 2: Exemplarischer Auszug der Datei test.csv

In der weiteren Bearbeitung der Aufgabenstellung sollen diese Datenpunkte mit den ausgewählten idealtypischen Funktionen verglichen und jeweils jener mit der geringsten Abweichung zugewiesen werden. Dieser Vorgang wird im Zuge dieser Arbeit als Klassifikation der Testdaten bezeichnet. Diese wird im Unterkapitel "Klassifikation der Testdaten" im Detail behandelt.

Gemäß der Vorgabe der Aufgabenstellung sollen sowohl die Trainings- (train.csv) als auch die Funktionsdaten (ideal.csv) in eine SQLite Datenbank eingelesen werden. Diese wird im Verzeichnis output/data bereitgestellt. Der Autor entschied sich hierbei für das Python-Modul sqlite anstelle des, in der Vorgabe vorgeschlagenen, Moduls sqlalchemy. Dieses bietet für die gegebene Anwendung eine vereinfachte Möglichkeit Daten mit einer SQLite Datenbank zu verarbeiten.

Über die UnitTests, die im Rahmen der, in Kapitel 1.2 beschriebenen, Modulinstallation durchgeführt werden, wird das lokale System auf die Möglichkeit eines direkten SQL-Imports hin überprüft. Dies bedeutet, dass versucht wird, die Daten in die SQLite Datenbank zu schreiben, ohne diese zuerst über das Python-Programm einzulesen und erst in einem weiteren Verarbeitungsschritt an die Datenbank übergeben zu müssen. Das Ergebnis der Überprüfung wird bei Erfolg im Logfile des Setups angeführt.

Auch in der Ausführung des Programms wird vorerst noch einmal der direkte Import versucht und erst im Falle eines Fehlschlages der Umweg über die Funktionen des Python-Moduls pandas gewählt.

Obwohl, aufgrund der geringen Größe, der in dieser Arbeit vorliegenden Datendateien, die Möglichkeit des direkten Imports nur eine untergeordnete Rolle spielt, schont ein solcher im Falle von größeren Datenmengen die Systemressourcen und spart Zeit in der Programmausführung.

Die Testdaten (test.csv) werden, gemäß Vorgabe, in der Programmausführung zeilenweise eingelesen, mit den Daten der gewählten Funktionen verglichen und gemeinsam mit der getroffenen Klassifikation in die SQLite Datenbank geschrieben.

Für die interne Verarbeitung der Daten wurde, im Rahmen des Programms, eine eigene Objektklasse geschaffen, die spezifische Funktionen sowie die Parameter der Visualisierung bereitstellt. Diese Klasse und deren Unterklassen für die verschiedenen Datensätze sind in der Datei functionfinder/classes.py definiert.

2.2 Auswahl idealtypischer Funktionen

2.3 Klassifikation der Testdaten

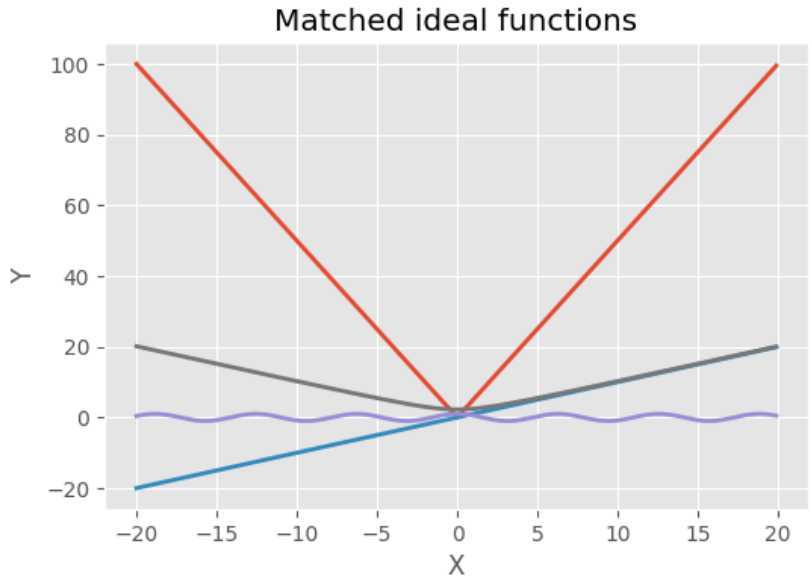

Figure 2: Darstellung der ausgewählten idealtypischen Funktionen **?**

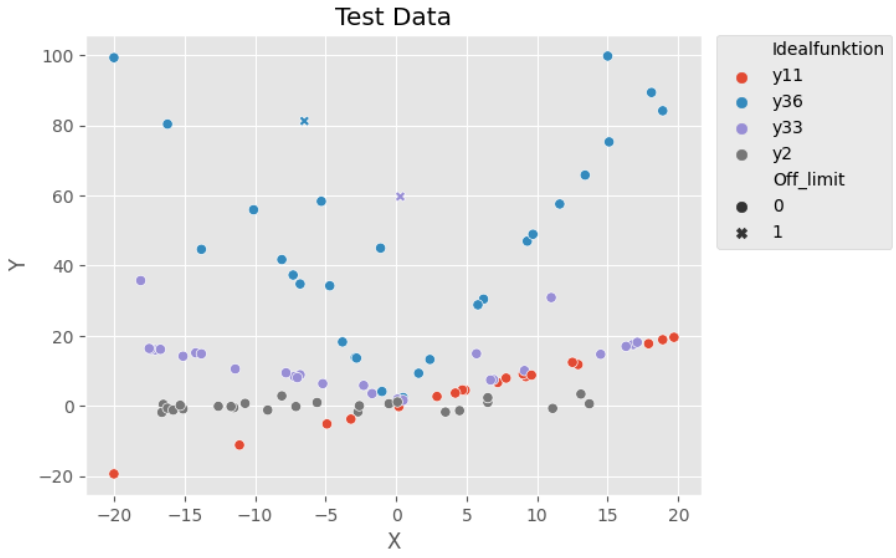

Figure 3: Klassifikation der Testdatensätze **?**

3 Zusammenfassung

3.1 Literature References

Here is an example of a reference with a page-number: (**?**, S. 6)

3.2 Pictures

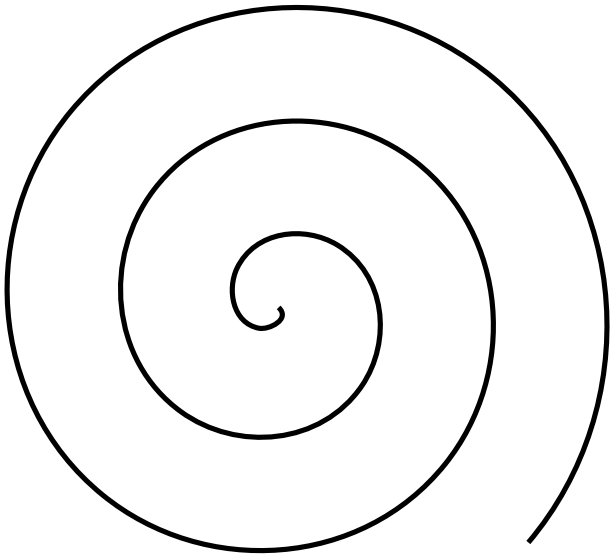

Figure 4: A spiral... smooth vector-based with a clean parametrisation!
Nothing to do with **?**

3.3 Tables

“ Industrial era ”	“Jobs ”	“ Wanted: Upgrade”’
Parts exchanger	Fitter	mecatronics specialist
eShop	reseller	“ Client-suggester”
“ Coding-guru”’	Softwaredesign	Whole-life designer
JA! Gut & Günstig	brand-names	“ Life-Style Feeling”
Internetbanking	Bank clerk	Customer adviser
Robots	Specialist	Machine supervisor
Bush	Gardener	Nature-sculptor
Painting	Painter	Interior Design

Table 3: Downgrade and Upgrade of job denominations
?

3.4 Listes

- one

- twoi
- threei

1. first
2. second
3. third

3.5 Formulæ

A formula can be written inline, e.g. as $\frac{d}{dx}\text{arctg}(x) = \frac{1}{1+x^2}$ or, in centered math:

$$\frac{d}{dx}\text{arctg}(x) = \frac{1}{1 + x^2} \tag{3.1}$$

Notice that formulæ that are centered start bigger (technically, they start in `\displaystyle`) than they start inline (technically, they start in `\textstyle` all subsequents reductions, e.g. an exponent, goes to `\scriptstyle` then `\scriptscriptstyle`). Indeed a best effort is made so that inline formulæ do not change the line height which would bother the eye of a reader.

Formulæ can be given a number and a label. Numbering happens automatically with `\begin{equation}` and `\end{equation}` and can be avoided if enclosing the formula betwee `\[` and `\]`. If using the `\label` macro inside, you can refer automatically to this equation using `\ref{label}`. E.g. Thanks to equation 3.1 one dare say that:

$$\int_0^t \frac{1}{1 + x^2} dx = \arctan(t) \tag{3.2}$$

3.6 Tools and Code

Many users of this template will want to include some code.

The simplest way to do so is to use the `\verb` macro which is followed by a sign, then some code, then the sign again to close. This is the inline version which works as in:

```
As we could calculate with \cite{Wolfram_alpha} using
\verb_integrate 1 / pi e ^ (t/pi) from zero to infinity_.
```

which yields:

As we could calculate with **?** using `integrate 1 / pi e ^ (t/pi)` from zero to infinity.

The multiline version of this is called `\begin{verbatim}` and finishes with `\end{verbatim}`.

3.7 Citation examples

Monography (**?**, S. 22)

Collection (**?**)

Article (**?**)

INTERNATIONAL
UNIVERSITY OF
APPLIED SCIENCES

Eidesstattliche Erklärung

I hereby certify...

.....	.....
Place, date	Signature