



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> $\Omega \rightarrow R^2$

Delivery date: 1.1.1970

Contents

I	List of Figures	III
II	List of Tables	1
1	Einleitung	2
1.1	Die Aufgabenstellung	2
1.2	Installationshinweise	2
1.3	Konfigurationshinweise	2
1.4	Struktur der Hausarbeit	3
2	Hauptteil	4
2.1	Bereitgestellte Datensätze	4
2.2	Auswahl idealtypischer Funktionen	6
2.3	Klassifikation der Testdaten	7
3	Zusammenfassung	9
3.1	Literature References	9
3.2	Pictures	9
3.3	Tables	9
3.4	Listes	9
3.5	Formulæ	10
3.6	Tools and Code	10
3.7	Citation examples	10

I List of Figures

1	Darstellung der bereitgestellten Trainingsdatensätze ?	4
2	Darstellung der ausgewählten idealtypischen Funktionen ?	7
3	Klassifikation der Testdatensätze ?	8
4	A spiral... smooth vector-based with a clean parametrisation! Nothing to do with ?	9

II List of Tables

1	Exemplarischer Auszug der Datei train.csv	4
2	Exemplarischer Auszug der Datei ideal.csv	5
3	Exemplarischer Auszug der Datei test.csv	5
4	Downgrade and upgrade of job denominations	9

1 Einleitung

1.1 Die Aufgabenstellung

Im Rahmen der Aufgabenstellung zum Kurs "DLMDWPMP01 – Programmieren mit Python" 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 (Least-Square). 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. Im Zuge der Entwicklung wurde das vorliegende Programm mit den Pythonversionen 3.9 und 3.11 unter Windows10 getestet.

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 potentiell 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.

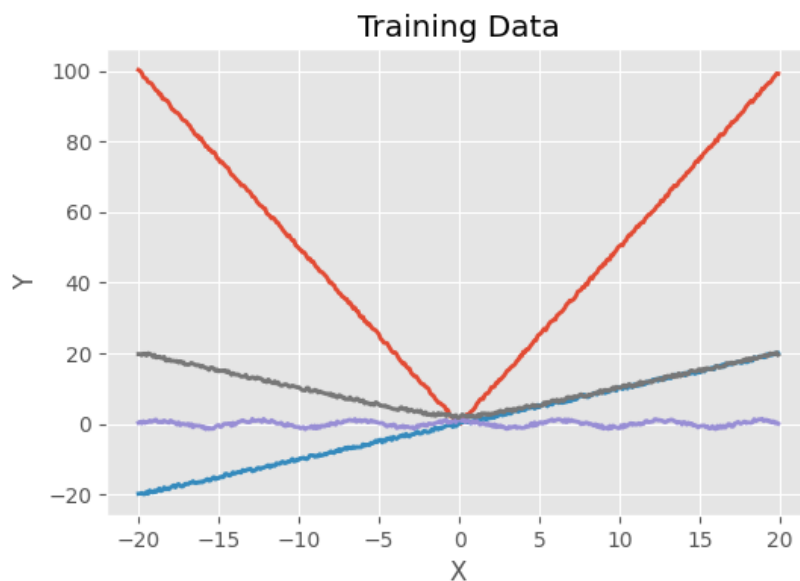


Figure 1: Darstellung der bereitgestellten Trainingsdatensätze ?

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.

x	y1	y2	y49	y50
-20.0	-0.9129453	0.40808207	0.9129453	0.3968496
-19.9	-0.8676441	0.4971858	0.8676441	0.47695395
-19.8	-0.81367373	0.58132184	0.81367373	0.5491291
-19.7	-0.75157344	0.65964943	0.75157344	0.6128399

Table 2: Exemplarischer Auszug der Datei ideal.csv

Im Gegensatz 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 3: 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 zu übergeben. 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, schon 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 geschaf-

fen, 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

Wie bereits im vorangegangenen Unterkapitel angeführt, steht zur Bearbeitung der Aufgabenstellung eine Auswahl an 50 idealtypischen Funktionen in einer CSV-Datei zur Verfügung. Aus diesen sollen für jede der vier Datenspalten der Trainingsdaten jeweils die passendste Funktion durch das Programm gewählt werden. Das Kriterium zur Selektion ist, gemäß Aufgabenstellung, die Minimierung der Summe aller quadratischen y-Abweichungen (Least-Square).

Mathematisch wird daher der kleinste Wert der Formel

$$\sum_{i=1}^n (TrainingData_i - IdealFunction_i)^2 \quad (2.1)$$

gesucht. Wobei die Variable *TrainingData* durch alle vier vorliegenden Trainingsdatenspalten iteriert und die Variable *IdealFunction* durch sämtliche verfügbaren Datenspalten der idealtypischen Funktionen. *n* steht hierbei für die Anzahl an Datenpunkten je Datenspalte, die in beiden Datensätzen gleichgroß sein muss.

In der Ausführung des programmierten Moduls wurde für den mathematischen Teil dieser Aufgabe (gemäß Gleichung 2.1) von der Eigenschaft des pandas-Objekts "Series" Gebrauch gemacht, die, bei der Subtraktion von zwei gleichlangen Serien der Länge *n* (in diesem Fall mit *n* = 400) erneut eine Serie derselben Länge mit den jeweils subtrahierten Einzelwerten zurück gibt. Auch die mathematische Funktion des Quadrats wird auf jeden Einzelwert der Serie angewandt. Die Summenbildung der Serie fasst diese nun, dem vorgegebenen Kriterium entsprechend, zu einem Wert zusammen. Diese Eigenschaft ermöglichte es daher, jede Datenspalte des Trainingsdatensatzes durch die Datenspalten der verfügbaren idealtypischen Funktionen zu iterieren und die zurückgelieferten Ergebnisse direkt zu vergleichen. Das Ergebnis mit der kleinsten Summe wurde jeweils als das Passendste abgespeichert.

Im Programmdesign bot es sich daher an, hierfür zwei Methoden getrennt zu definieren:

1. Die Methode der Iteration und des Vergleiches zur Ermittlung des minimalen Fehlers wurde im Modul `functionfinder/datafunctions.py` definiert.
2. Die Methode zur Berechnung des Fehlers (gemäß Vorgabe, entsprechend der Gleichung 2.1) wurde in der Datei `functionfinder/config.py` definiert.

Dies erlaubt es, auf einfache Weise in der Konfigurationsdatei `functionfinder/config.py`, die Methode zur Fehlerberechnung zu verändern und gegebenenfalls andere Kriterien, wie zB die durchschnittliche mittlere Abweichung (Mean Squared Error - MSE) zur Bewertung heranzuziehen.

Als Rückgabewert der ersten genannten Methode, wurde vom Autor der Python-Datentyp dictionary gewählt. In diesem wurde zu jedem Key (die Datenspalten des Trainingsdatensatzes) der Value in Form eines Tupels abgelegt, welcher den Namen der gewählten Datenspalte des Datensatzes idealtypischer Funktionen, sowie die berechnete Abweichung beinhaltet.

Darstellung des Rückgabewertes der berechneten Übereinstimmungen:

```
{ 'y1': ( 'y36', 33.71178854422821 ),  
  'y2': ( 'y11', 32.62893138832121 ),  
  'y3': ( 'y2', 33.11847188090256 ),  
  'y4': ( 'y33', 31.75243110139478 ) }
```

Die oben angeführte Darstellung des Rückgabewertes macht ersichtlich, welche idealtypischen Funktionen für die jeweiligen Trainingsdatenspalten gewählt wurden. Die grafische Repräsentation lässt auch eine optische Bestätigung der getroffenen Auswahl zu.

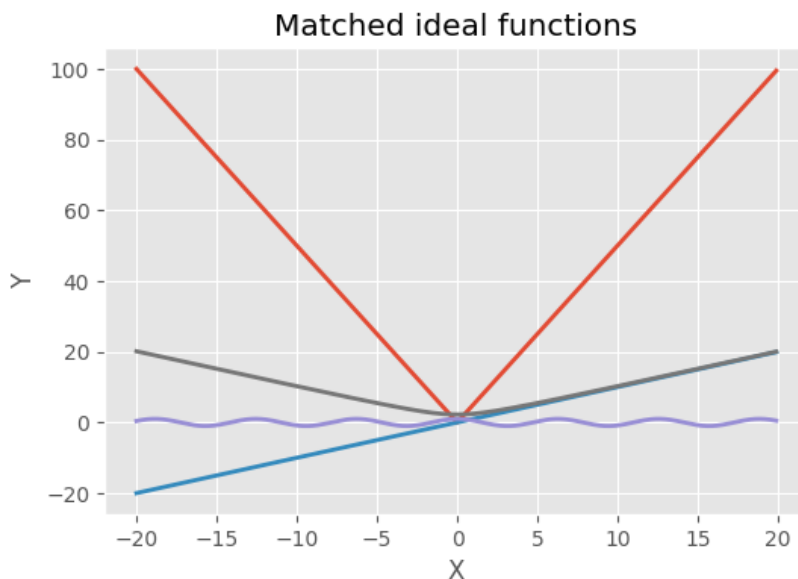


Figure 2: Darstellung der ausgewählten idealtypischen Funktionen ?

2.3 Klassifikation der Testdaten

In diesem Unterkapitel wird die Klassifikation der Testdaten mithilfe der ausgewählten idealtypischen Funktionen sowie den jeweiligen berechneten Abweichungen (siehe vorangegangenes Unterkapitel) diskutiert.

Entsprechend der Vorgabe zur vorliegenden Aufgabenstellung sollen die, in der Datei `test.csv`, abgebildeten Datenpunkte, Zeile für Zeile eingelesen und mit dem maximalen Abstand von $a * \sqrt{2}$ einer der vier gewählten Funktionen zugeordnet. Wobei a den berechneten Fehlerwert des Trainingsdatensatzes der jeweiligen Funktion repräsentiert.

In der Umsetzung wurde oben angeführte Berechnungsformel wiederum in zwei Teile zerlegt. Der Wert a stammt aus einer vorangegangenen Berechnung und ist an dieser Stelle bereits bestimmt. Der zweite Teil betrifft den Multiplikationsfaktor $\sqrt{2}$, welcher durch die Vorgabe zur Aufgabenstellung festgelegt wurde. Um diesen gegebenenfalls anpassen zu können wurde dazu die Variable *factor* in der Konfigurationsdatei `functionfinder/config.py` definiert. Dadurch wird wiederum eine flexible Herangehensweise an die Bewertung der Testdaten gewährleistet.

Für die Bewertung selbst wurde in der Datei `functionfinder/datafunctions.py` die Methode `calculate_best_ideal` definiert. Diese nimmt im Wesentlichen als Parameter die aktuell eingelesene Zeile der Datei `test.csv`, die Auswahl der idealtypischen Funktionen inklusive der Fehlerberechnungen, den Datensatz der idealtypischen Funktionen, sowie einen Funktionsparameter für die Filterung der Ergebnisse entgegen.

Letzterer ist für die Berechnung der passenden idealtypischen Funktion auf `min` gesetzt. Das bedeutet, dass aus den Abweichungen zu allen vier gewählten Funktionen, diejenige mit der kleinsten Abweichung gewählt wird.

Die Aufgabenstellung sieht vor, dass die Testdaten um die Werte der Abweichung (DeltaY), sowie die Nummer der gewählten Funktion (Idealfunktion) erweitert werden und in eine eigene Tabelle der SQLite Datenbank geschrieben werden. Der Autor entschied sich an dieser Stelle, noch einen Wert (`Off_limit`) hinzuzufügen. Dieser Wert, vom Datentyp *Boolean*, zeigt an, wenn dieser Datenpunkt den gesetzten Maximalabstand ($a * \sqrt{2}$) zu

keiner der gewählten Funktionen einhalten konnte und wird in diesem Fall auf *True* gesetzt. Dem Datenpunkt wird aber dennoch die Funktion mit dem geringsten Abstand zugewiesen.

Für eine Evaluierung der Ergebnisse wurde in der Objektklasse *testdata* die Methode *off_limit* definiert, die aus der SQLite Datenbank jene Werte abfragt, bei denen dieser Wert auf *True* gesetzt ist. Diese Datenpunkte werden nochmals an die Methode *calculate_best_ideal* übergeben, wobei diesmal der Funktionsparameter für die Filterung auf *raw* gesetzt wird. Dadurch werden die Ergebnisse des Vergleichs zu jeder der vier gewählten idealtypischen Funktionen ausgegeben und können über das Logfile manuell evaluiert werden.

Bei der Bearbeitung der zur Aufgabe übergebenen Datensätze waren hierbei zwei Datenpunkte des Testdatensatzes betroffen. Der tabellarisch aufbereitete Auszug aus dem entsprechenden Logfile zeigt die unten angeführten Ergebnisse:

2 entries had a deviation above the set limit:

x	y	DeltaY	Idealfunktion	Off_limit
-6.5	81.21402	48.714020	y36	True
-6.5	81.21402	87.714020	y11	True
-6.5	81.21402	80.237432	y2	True
-6.5	81.21402	74.340156	y33	True

x	y	DeltaY	Idealfunktion	Off_limit
0.3	59.68033	58.180330	y36	True
0.3	59.68033	59.380330	y11	True
0.3	59.68033	58.724993	y2	True
0.3	59.68033	57.424227	y33	True

In den oben angeführten Tabellen wird die Distanz der beiden betroffenen Datenpunkte zu den vier gewählten Funktionen ersichtlich. In der Zuweisung wurden dennoch jene Funktionen mit dem geringsten Abstand gewählt (grafisch hervorgehoben).

In der grafischen Repräsentation der Ergebnisse ist die Annäherung der einzelnen Datenpunkte des Testdatensatzes an die gewählten idealtypischen Funktionen gut erkennbar. Jene beiden Datenpunkte, die im Vergleich den definierten Maximalabstand überschritten haben werden durch das Symbol *x* dargestellt. Wobei hier die optische Beurteilung des zweiten Punktes eher eine Tendenz zur Funktion *y36* vermuten ließe, als zur Funktion *y33*. In der oben angeführten Tabelle zu diesem Datenpunkt wird aber auch deutlich, dass die Berechneten Abstände tatsächlich sehr nahe beieinander liegen.

Zusätzlich kann in der Visualisierung beobachtet werden, dass auch an den Schnittpunkten der Funktionen die Zuordnung der Datenpunkte ausreichend gut zu funktionieren scheint und der Verlauf der idealtypischen Funktionen optisch erkennbar nachverfolgt wird.



Figure 3: Klassifikation der Testdatensätze ?

3 Zusammenfassung

Zeitersparnis: Tabellenweise verarbeitung der Testdaten Zuweisung nicht aufgrund des kleinsten Abstandes sondern aufgrund des kleinsten Abstandes in Bezug auf a (den berechneten Fehlerwert der Trainingsdaten)

3.1 Literature References

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

3.2 Pictures

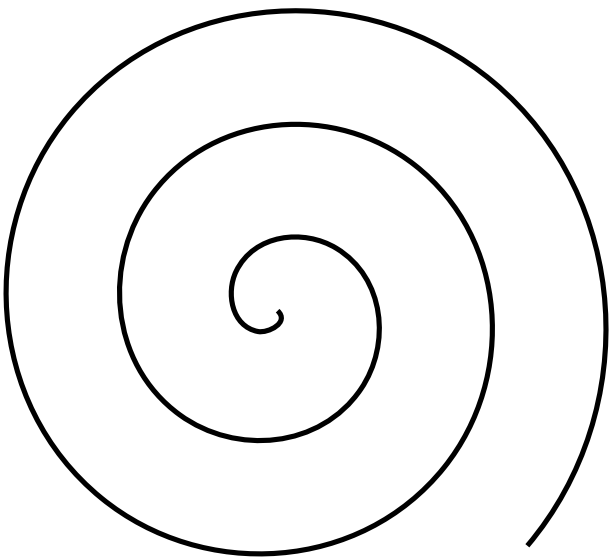


Figure 4: A spiral... smooth vector-based with a clean parametrisation!
Nothing to do with ?

3.3 Tables

“ Industrial era ”	“Jobs ”	“ Wanted: Upgrade”
Parts exchanger	Fitter	mecatronics special-ist
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 4: 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 $\frac{d}{dx}\arctg(x) = \frac{1}{1+x^2}$ or, in centered math:

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

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:

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

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 (`?`)

Eidesstattliche Erklärung

I hereby certify...

.....

Place, date

.....

Signature