Dokumentation zum Sprachverarbeitungs-Plugin «Lang-Main» für die Software «TOM»

erstellt durch:	d-opt GmbH Oberneumarker Str. 59 08496 Neumark (Sachsen)
erstellt von:	Florian Förster
Erstellungsdatum:	22.01.2025
geändert von:	Florian Förster
Änderungsdatum:	20.06.2025
Programmversion:	V0.1.3
Dokumentversion:	V1.1

Inhaltsverzeichnis

1 Pfadsystem und Verzeichnisstruktur

Die Anwendung ist ein Standalone-Paket, welches in einem Ordner gebündelt alle notwendigen Ressourcen für den Betrieb mitliefert. Programmintern wird zwischen einem *Wurzelverzeichnis* und einem *Bibliotheksverzeichnis* unterschieden. Das Wurzelverzeichnis ist der Ordner, welcher alle Programmdateien und -ordner beinhaltet. Dieses enthält die Ordner *„bin“* und *„data“*.
Das Verzeichnis *„bin“* enthält die Anwendung einschließlich der genutzten Sprachmodelle sowie der Konfiguration. Dieses stellt innerhalb der Anwendung das Bibliotheksverzeichnis dar. Alle Pfade der Konfigurationsdatei werden relativ zu diesem Verzeichnis angegeben. Weitere Informationen folgen im Kapitel zur Konfiguration.
Das Verzeichnis *„data“* enthält zwei Unterordner *„in“* und *„out“*. Der Ordner *„in“* dient der Ablage von Eingabedaten für Workflows, die dateibasiert ablaufen. Im Ordner *„out“* werden die Ergebnisse der Pipelines abgelegt. Dies schließt auch relevante Zwischenergebnisse ein. Bei einer Datensatzgröße von ca. 120.000 Einträgen erreichen die Ausgabedaten für einen Standardworkflow ca. 55 Megabyte. Ausgabeordner werden aktuell *nicht automatisch gelöscht*.

2 Systemanforderungen und Laufzeit

Mit Nutzung des Standardmodells, welches die Qualität des Outputs gegenüber der Geschwindigkeit vorzieht, benötigt das Modell für einen Datensatz mit ca. 120.000 Einträgen, wovon zahlreiche Duplikate eliminiert wurden, folgende Systemanforderungen:

- CPU: Es werden x86-64-CPUs benötigt, die die Befehlssatzerweiterung „AVX2“ unterstützen (zur Ausführung des optimierten Modells notwendig)
- RAM: 4,5 GB (Spitzenwert ca. 4,2 GB, Mittelwert bei 3,1 GB)

Bei der Verwendung eines kleineren und schnelleren, aber dafür etwas weniger genauen Modells ändert sich die RAM-Nutzung wie folgt:

- RAM: 3,5 GB (Peak lag bei etwa 3,2 GB, Mittelwert bei 2,8 GB)

Die Ausführungszeit für einen Rechenlauf kann nicht verallgemeinert werden, da dies vom verwendeten Prozessor (Taktrate und Anzahl der Kerne) und der Datensatzgröße abhängt. Reicht der Arbeitsspeicher nicht aus, steigt die Ausführungszeit durch Auslagerung des Systems erheblich.
Bei Verwendung eines kleineren Modells konnte auf derselben Hardware mit demselben Datensatz (120.000 Einträge) eine zeitliche Reduktion des Rechenlaufs von ca. 25 Prozent erzielt werden.

3 Konfiguration

Für die Konfiguration des Programms wird eine Datei im TOML-Format genutzt, die im Bibliotheksverzeichnis abliegt. Sie ist zentrales Element, um die wesentlichen Parameter der Datenverarbeitungsprozesse ohne Umschreiben des Quellcodes zu verändern.

3.1 Pfade

`[key: paths]`

In diesem Bereich können die Pfade für folgende Bereiche geändert werden:

- Eingabedaten `[key: inputs]`
- Ausgabedaten `[key: results]`
- Sprachmodelle `[key: models]`

Die Pfadangabe erfolgt relativ zum Bibliotheksverzeichnis *„bin“*.
Die Standardkonfiguration basiert auf der Ordnerstruktur, in der das Programm geliefert wird, und sollte deshalb nicht geändert werden.
Des Weiteren steht eine Option zur Festlegung des Dateinamens für den Export des Token-Graphs zur Verfügung:

- Export-Dateiname Token-Graph `[key: graph_export_filename]`

Der fertig generierte Token-Graph wird im Ausgabeordner (siehe auch Kapitel „Token-Graph: Workflow und Eigenschaften“) mit diesem Dateinamen und dem Suffix „.graphml“ abgelegt.

3.2 Logs

`[key: logging]`

Hier kann das Logging-Verhalten der Anwendung konfiguriert werden. Es stehen folgende Optionen zur Verfügung:

- aktvieren `[key: enabled]`: aktiviert oder deaktiviert das Logging der Anwendung gänzlich
- Konsole `[key: stderr]`: aktiviert oder deaktiviert das Logging der Anwendung zur Systemausgabe (Systemstandard für STDERR)
- Datei `[key: file]`: aktiviert oder deaktiviert das Logging der Anwendung in eine Log-Datei, die im Bibliotheksverzeichnis abgelegt wird

3.3 Ablaufkontrolle

`[key: control]`

Hier kann konfiguriert werden, welche ETL-Pipelines ausgeführt werden. Die Standardkonfiguration sieht alle Transformationen vor, die für die Erzeugung eines Graphen mit allen erforderlichen Kenngrößen benötigt werden.
Die Änderung dieser Konfiguration wird nicht empfohlen, da bestimmte Optionen zusätzliche Abhängigkeiten benötigen, die in der Standarddistribution nicht enthalten sind. Die Ergebnisse der deaktivierten Optionen sind zum aktuellen Stand nicht weiter verarbeitbar.

3.4 ETL-Pipelines

3.4.1 Vorverarbeitung

`[key: preprocess]`

Unter diesem Schlüssel werden alle relevanten Parameter zusammengefasst, die für die Datenvorverarbeitung notwendig sind.
Es stehen folgende Optionen zur Verfügung:

- Datumsspalten `[key: date_cols]`: definiert alle Eigenschaften im Datensatz, die Datumsangaben repräsentieren; vollständig und korrekt erforderlich; falsche Angaben führen zu Programmfehlern
- Ziel-Eigenschaft für Graph-Analyse `[key: target_feature]`: definiert die Eigenschaft, welche im Rahmen der Pipeline für die Graph-Analyse relevant ist und entsprechend aufbereitet wird; korrekt erforderlich; falsche Angaben führen zu Programmfehlern
- Schwellwert (inklusiv) für Zeichenanzahl von Einträgen `[key: threshold_amount_characters]`: gibt an, bis zu welcher Zeichenzahl ein Eintrag als ungültig erachtet wird; negative Werte werden auf 0 gesetzt
- Schwellwert (inklusiv) für die inhaltliche Ähnlichkeit von Einträgen zueinander `[key: threshold_similarity]`: gibt an, ab welcher Ähnlichkeit Einträge als ähnlich genug gelten, um zusammengefasst zu werden (inhaltliche Duplikate); muss zwischen 0 und 1 liegen, Werte außerhalb des Bereichs rufen einen gewollten Programmfehler hervor; der Standardwert sollte nicht verändert werden

3.4.2 Graph-Nachbearbeitung

`[key: graph_postprocessing]`

Hier werden alle Parameter zusammengefasst, die für die Nachbearbeitung des Graphen relevant sind und von außen verändert werden können.
Es stehen folgende Optionen zur Verfügung:

- Option zum Aktivieren/Deaktivieren der Normalisierung von Kantengewichten [`key: enable_edge_rescaling`]:
  - Bool'scher Wert – `true`: Aktivierung der Kantengewichtsnormalisierung (alle Kantengewichte liegen zwischen 0 und 1 als Gleitkommazahl)
  - Bool'scher Wert – `false`: Deaktivierung der Kantengewichts-normalisierung (alle Kantengewichte liegen als Ganzzahlen vor und geben an, wie häufig die Wortverbindungen im Datensatz erschienen)
- Maximalwert (inklusiv) für die Anzahl auszugebener Kanten `[key: max_edge_number]`: gibt an, wie viele Kanten der Graph schlussendlich maximal enthalten soll; Kanten werden nach ihrem Gewicht priorisiert; negative Werte bedeuten keine Filteranwendung

3.4.3 Zeitreihenanalyse

Diese Konfigurations-Parameter sind für die aktuelle Programmversion nicht relevant.

4 Anforderungen an den Datensatz

Für die aktuellen und zukünftigen Pipelines werden nachfolgende Eigenschaften zwingend benötigt. Die Namen basieren auf den zur Verfügung gestellten Testdatensätzen.
Notwendige Eigenschaften/Felder:

- VorgangsID
- ObjektID
- HObjektText
- VorgangsTypName
- VorgangsBeschreibung
- VorgangsArtText
- ErledigungsBeschreibung

- Datumsfelder:
  - VorgangsDatum
  - ErledigungsDatum
  - Arbeitsbeginn
  - ErstellungsDatum

5 Token-Graph: Workflow und Eigenschaften

5.1 Workflow

Ziel des Workflows ist die Erstellung eines Token-Graphs, der sogenannten „Wortwolke“.

5.1.1 Dateibasiert (CSV)

Eingabe
Zum gegenwärtigen Zeitpunkt implementiert das Plugin nur einen dateibasierten Workflow mithilfe von CSV-Dateien. Hierzu wird der CSV-Datensatz, dessen notwendigen Attribute im nachfolgenden Kapitel beschrieben werden, in den konfigurierten Ordner für Eingabedaten (*„data/in/“*) abgelegt. Der Dateiname ist für den Anstoß des Rechenlaufs notwendig.

Verarbeitung
Zur Identifikation des Rechenlaufs und eventuell auch des verwendeten Datensatzes wird noch ein Identifier benötigt, der durch den Anwender generiert wird. Dieser Identifier wird der entsprechenden Methode der Bibliothek zusammen mit dem Dateinamen mitgegeben.

Ausgabe
Im konfigurierten Ausgabeordner wird ein Unterordner mit dem Namen des Identifier angelegt, in welchen alle Ausgaben des Rechenlaufs abgelegt werden. Hierdurch wird es möglich, unterschiedliche Rechenläufe vorzuhalten (Cache) und gegebenenfalls erneut abzurufen, ohne einen neuen Rechenlauf starten zu müssen.

Wichtig:
Aktuell werden die Ordner nicht automatisch nach bestimmten Kriterien gelöscht und verbleiben somit langfristig auf der Festplatte. Werden immer neue Identifier genutzt, kommt es so zu einer Ansammlung von Daten, was in speicherbegrenzten Umgebungen unerwünscht sein kann. Es ist deshalb empfehlenswert, die Ordner regelmäßig zu löschen oder die Identifier rotierend wiederzuverwenden. Im letztgenannten Fall werden bestehende Dateien einfach überschrieben.

5.2 Ergebnis

5.2.1 Datei

Unabhängig von der konkreten Pipeline/Prozedur wird im Ausgabeordner (*„data/out/{Identifier}“*) eine Datei im GraphML-Format abgelegt. Sie trägt den Namen, der in der Konfiguration unter [`paths/graph_export_filename`] definiert wurde mit der Dateinamenerweiterung „.graphml“: *„{KONFIG-WERT}.graphml“*. Dies ist der Token-Graph mit allen abgeschlossenen Nachbearbeitungen und Transformationsschritten. GraphML ist ein standardisiertes XML-basiertes Austauschformat für Graph-Daten.

5.2.2 Relevante Eigenschaften des Graphen

Der Token-Graph enthält mehrere Attribute, die unter anderem für die grafische Darstellung des Graphen genutzt werden können:
Knoten:

- Betweenness Centrality [`attr: betweenness_centrality`]: gängige Graph-Metrik, die die Bedeutung von Knoten über Nachbarschaftsbeziehungen berechnet
- Gewichteter Grad [`attr: degree_weighted`]: gibt die Summe aller ein- und ausgehenden Kantengewichte an
- Wichtigkeit [`attr: importance`]: eigene Metrik, die die Betweenness Centrality mit dem gewichteten Grad kombiniert; kann gut zur grafischen Skalierung der Knotengröße genutzt werden

Kanten:

- Gewichtet [`attr: weight`]: Kantengewicht als Maß der Häufigkeit des Auftretens von zwei Wörtern zueinander; skaliert zwischen 0 und 1 zur verbesserten grafischen Darstellung

6 C#-Bibliothek

6.1 Allgemein

Das zur Verfügung gestellte Plugin wird in Form einer C#-Bibliothek nach dem .NET-Standard der Version 2.0 verteilt, welche in Form der DLL-Datei *„doptPlugin.dll“* im Anwendungsordner *„bin“* zu finden ist. Die gesamte Anwendung wird mit einer Python-Umgebung gebündelt, die alle relevanten Abhängigkeiten sowie die benötigten Pipelines enthält. Die C#-Bibliothek fungiert als dünner Wrapper um den eigentlichen Python-Kern. Über das Plugin und seine bereitgestellten Methoden wird im Hintergrund der entsprechende Workflow mit dem Python-Interpreter angestoßen.

6.2 Unterstützte Architektur

Die Bibliothek unterstützt ausschließlich 64-bit-Systeme. Dies ist notwendig, da die zugrundeliegende Python-Installation inklusiver diverser Extension-Modules ebenfalls für x64 kompiliert wurden und für Modellinferenzen entsprechender Arbeitsspeicher adressierbar sein muss.

6.3 Referenz

Die C#-Bibliothek enthält einen Namespace mit dem Namen *„doptPlugin“*, welcher lediglich eine Klasse *„Plugin“* beinhaltet. Diese Klasse stellt zum aktuellen Zeitpunkt einen Constructor sowie eine öffentliche Methode zur Verfügung, mit deren Hilfe CSV-Dateien eingelesen und verarbeitet werden können.
Die Zustandsverwaltung der Python-Runtime erfolgt intern.
Struktur:

- `Namespace: dopt.TOM`
  - `Klasse: Plugin`
    - `Constructor: Plugin`
    - `Methode: RunOnCSV`
    - `Methode: Dispose`

6.3.1 (Constructor) Plugin

```
public Plugin(string runtimePath = “”)
```

Parameter:

- `string runtimePath`: Pfad zur Python-Umgebung – erlaubt die Platzierung der Python-Umgebung auf einem Pfad unabhängig von der DLL; falls nicht angegeben (leerer String): Suche relativ zum Applikationsordner

6.3.2 RunOnCSV

```
public void RunOnCSV(string identifier, string filename)
```

Parameter:

- `string identifier`: eindeutiger Identifier zur Identifikation des Datensatzes; genutzt für Ordnererstellung im Ausgabeordner, in dem alle Ergebnisse gespeichert werden
- `string filename`: Dateiname der CSV-Datei, wie sie im konfigurierten Eingabeordner vorliegt

6.3.3 Dispose

```
public void Dispose()
```

Implementierung des IDisposable-Interface nach dem De-Facto-Standard in C#
Diese Methode sollte zum Beenden des Plugins verwendet werden. Die Python-Runtime wird damit ordnungsgemäß beendet und genutzte Ressourcen wieder freigegeben.
Parameter:

- keine