Anleitung Dynamic-Pricing

# Übersicht

Das Tool nutzt im Wesentlichen Python zur Modellierung (PyMC3) und zum Ansprechen des Webservices (Flask-Framework) sowie MongoDB zum Speichern der Modelle. Die Subversionierung des Codes geschieht über GitLab. Als Entwicklungsumgebung wird PyCharm empfohlen

# Ansprechpartner

Chris-Gabriel Islam

# Zugänge

## Flask-Server starten/beenden/überwachen

1. Per SSH verbinden
   * Adresse:
   * Username:
2. In screen wechseln, um Server auch im Hintergrund laufen lassen zu können:
   * screen -d
3. Virtualenv aktivieren
   * source ~/envs/dynamic\_pricing/bin/activate
4. Server starten
   * cd ~/httpdocs
   * python api.py
5. Screen verlassen:
   * Strg A + D

## MongoDB

* Adresse:
* Authentifizierung via SSH-Tunnel:
  + Adresse:
  + Username:
  + Passwort:
* Als Oberflächen-Tool zum Verwalten und Einsehen am besten Robo3T verwenden

# Schnittstellen

Generell:

* Bei Listenargumenten einzelne Einträge in der Anfrage durch Komma trennen
* Preise mit Punkt als Dezimaltrennzeichen (= englische Schreibweise) angeben

## Bestelldaten vom Webshop zu MongoDB schicken

* POST-Befehl an …/api/transfer\_bestdata
* Daten, deren Index kleiner als 3000000 ist, werden von der MongoDB ignoriert.
* Nach Absetzens des Befehls wird ein Thread gestartet, der im Konsolenoutput des Flask-Servers betrachtet werden kann.
* Sollte der Thread abbrechen, startet ein erneuter POST-Befehl direkt an der zuletzt hinzugefügten Stelle
* Sollte es zu Problemen kommen, kann manuell die MongoDB-Collection gedroppt werden (Input->Bestelldaten)

## Modelle trainieren

* POST-Befehl an .../api/train\_models?aids=X,Y,Z&uvps=A,B,C
* Beispielaufruf: .../api/train\_models?aids=1025078,1021517&uvps=59.99,32.99
* Nach Absetzens des Befehls wird ein Thread gestartet, der im Konsolenoutput des Flask-Servers betrachtet werden kann.
* Für jeden Artikel wird für jede mögliche Preiskategorie ein Modell unter Main/Models angelegt
* Unter Main/Overview wird eine Übersicht abgelegt, die darüber informiert, wann der Artikel das letzte Mal trainiert wurde und ob genügend Daten zum Trainieren vorliegen
* Der Prozess dauert ca. 3 Minuten pro Artikel und benötigt 30-100 MB

## Optimaler Preis ausgeben

* GET-Befehl an .../api/optimal\_prices?aids=X,Y,Z&uvps=A,B,C
* Beispielaufruf: .../api/optimal\_prices?aids=1025078,1021517&uvps=59.99,32.99
* Optimaler Preis pro Artikel sowie Angaben zu erwartendem Profit sowie Güte der Modellierung
* Da es sich um eine probabilistische Modellierung handelt, ändert sich das Ergebnis von Aufruf zu Aufruf leicht
* Falls ein optimaler Preis für ein Artikel gesucht wird, auf den noch nicht trainiert wurde, wird kein Preis angegeben, sondern die Aufforderung zum Trainieren gestellt:

{

"1021517": {

"message": "Train model!"

}

}

* Beispielantwort:  
  {

"1021517": {

"best\_price": 27.49,

"exp\_costs\_per\_unit": 0,

"exp\_demand": 0,

"exp\_profit\_per\_unit": 0,

"model\_variance\_sufficient": false,

"test\_price": false

},

"1025078": {

"best\_price": 54.99,

"exp\_costs\_per\_unit": null,

"exp\_demand": null,

"exp\_profit\_per\_unit": null,

"model\_variance\_sufficient": false,

"test\_price": true

}

}

## Abfrage, welche Artikel (nochmals) trainiert werden können

* GET-Befehl an .../api/trainable\_articles
* Es wird eine JSON-Response zurückgegeben, die alle Artikel beinhaltet, die wieder trainiert werden könnten. Als Meta-Daten wird pro Artikel das letzte Trainingsdatum, die letzte Beobachtung, die in das Training eingeflossen ist und (TODO) die Anzahl der neuen Beobachtungen
* Beispielantwort:

# Theorie

Das Tool nutzt im Wesentlichen probabilistische Modellierung (MCMC). Weitere Infos unter: <https://blog.griddynamics.com/dynamic-pricing-algorithms/> (Scenario 4).

Einige Mindestvorgaben, wie z.B. die Mindestanzahl an Beobachtungen pro Preisklasse, sind hartkodiert in der Datei constants.py angegeben und können dort angepasst werden.

Die möglichen Preise basieren auf dem Psychopreiser und sind hartkodiert in “assets/possible\_prices.csv” enthalten.

# Todos

* Backup-Job für MongoDB kreieren
* Server-Logging aktivieren
* Unterschiedliche Preise pro Tag beachten (s. TODO im Code)
* Preisliste mit Psychopreiser abgleichen
* Anzahl der neuen Beobachtungen für api\_trainable\_articles
* Warning bei pymc3 entfernen:
  + FutureWarning: In v4.0, pm.sample will return an `arviz.InferenceData` object instead of a `MultiTrace` by default. You can pass return\_inferencedata=True or return\_inferencedata=False to be safe and silence this warning.
* Weitere Kovariate neben Preis mit einbeziehen
  + Markteintrittsdatum
  + Anzahl der Seitenaufrufe / Conversions
  + Kalendereffekte
  + Google-Analytics-Daten