|  | **Софийски университет „Св. Кл. Охридски”**  Факултет по математика и информатика |  |
| --- | --- | --- |

**Курсов Проект**

на тема: „Система за препоръки на игри”

Студент: **Биляна Емилова Инджева** Ф.Н. **4MI0800106**

Курс: „4 курс“, Учебна година: 2024/25

Преподаватели: **проф. Иван Койчев**

=================================

Декларация за липса плагиатство:

* Плагиатство е да използваш, идеи, мнение или работа на друг, като претендираш, че са твои. Това е форма на преписване.
* Тази курсова работа е моя, като всички изречения, илюстрации и програми от други хора са изрично цитирани.
* Тази курсова работа или нейна версия не са представени в друг университет или друга учебна институция.
* Разбирам, че ако се установи плагиатство в работата ми ще получа оценка “Слаб”.

02.02.25 г. Подпис на студента:

**Съдържание**

[**1**](#_gjdgxs) **УВОД 2**

[**2**](#_30j0zll) **ПРЕГЛЕД НА ОБЛАСТA *(ДА СЕ ЗАМЕНИ С КОНКРЕТНО ЗАГЛАВИЕ СПОРЕД ТЕМАТА)* 2**

[**3**](#_1fob9te) **ПРОЕКТИРАНЕ 2**

[**4**](#_3znysh7) **РЕАЛИЗАЦИЯ, ТЕСТВАНЕ/ЕКСПЕРИМЕНТИ 2**

[4.1](#_2et92p0) Използвани технологии, платформи и библиотеки 2

[4.2](#_tyjcwt) Реализация 2

[**5**](#_3dy6vkm) **ЗАКЛЮЧЕНИЕ 2**

[**6**](#_1t3h5sf) **ИЗПОЛЗВАНА ЛИТЕРАТУРА 2**

# **Увод**

Основната цел на този курсов проект е да се разработи система за препоръки на игри в платформата *Steam*. Съществуват милиони потребители и хиляди заглавия, което прави откриването на подходящи игри предизвикателство. Чрез подобна система се стремим да улесним потребителите в избора на игри, отговарящи на техните предпочитания.

Целите и задачите са:

* Събиране и почистване на данни от *Steam* (описания на игри, статистики за потребителска активност).
* Реализиране на няколко подхода за препоръки: *Content-based, K-Means, Autoencoder* (малък и голям).
* Оценка на всеки модел чрез метрики *Precision, Recall* и *F1*.

# Системите за препоръки

Системите за препоръки са ключови за множество платформи (*Netflix, Amazon, Steam*), тъй като значително подобряват потребителското изживяване. Съществуват няколко основни подхода:

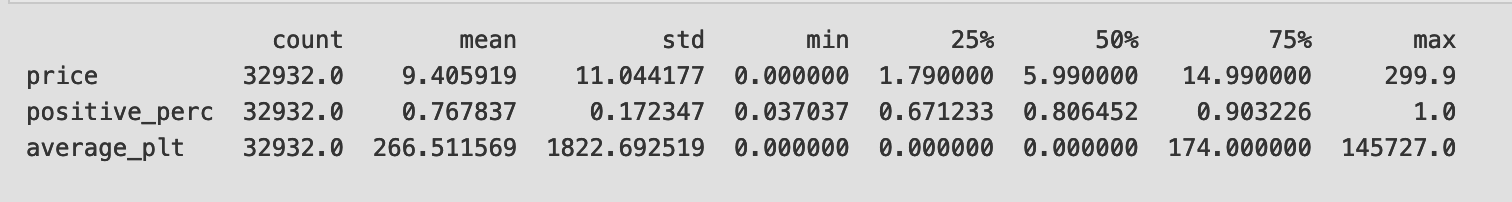
* ***Content-based Filtering***: препоръчва артикули, подобни на това, което потребителят вече е харесал, на база описателни характеристики.
* ***Collaborative Filtering***: използва сходството между потребители/артикули въз основа на рейтингите им (не разглеждаме в дълбочина тук, защото имаме ограничен достъп до реални оценки).
* **Хибридни методи**: комбинират информация за съдържание и поведение на потребителя.
* ***Neural Networks***: *Autoencoder* и *Deep Learning* подходи.

В нашия проект сме се фокусирали върху ***content-based*** и ***autoencoder*** подходи, както и базов clustering (*K-Means*).

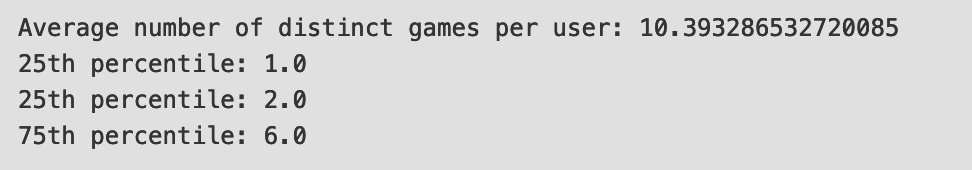
# **Проектиране**

**Данни:**

* Данни за игри: Съдържа данни за игрите, с които тренираме моделите. Колони: описание, цена, брой положителни / отрицателни ревюта, средно време на игра и други, които не използваме. Общо имаме данни за ~32к игри. Ето и анализ на цифровите колони, които тъкмо споменахме:



* Потребителски данни: Използваме само при оценяването на моделите. Колони: потребител, игра, тип поведение (купена / играна), прекарани часове. Като имаме ~12к уникални потребителя, и ~5к игри, след почистването на данните остават ~100к реда. Можем да видим данните за средния потребител и броя игри, които играре / притежава - интересно е, че явно малък брой от потребителите играя над 10 игри ( а те ще са ни по-полезни за оценяването на пропоръчителя):



И данни за най-популярните игри, интересно е колко бързо се смъкват броя играчи:  


**Архитектура на проекта:**

* **Предобработка**:
  + Почистване на дублирани записи, както и анализ на данните.
  + Премахване/заместване на стойности с липсваща информация - и двата сета с данни почти нямат липсваща информация. Имаше значително количество игри, които след “нормализиране” на имената - не съвпадаха с никоя игра от данните за всички игри и тях ги изтриваме.
  + Създаване на векторизирани колони за данните (*TF-IDF, one-hot* за цените и средно време на игране).
* **Препоръчители**:
  + *ContentBasedRecommender*: сравнява векторите на игрите чрез cosine similarity.
  + *KMeansRecommender*: групира игрите в клъстери и препоръчва от най-близкия клъстър.
  + *AutoencoderRecommender*: генерира латентно представяне, от което се извличат подобни игри. (Имаме множество, за тестване на хипер-параметри)
* **Оценител:**
  + Функция, която правилно да подава данните на потребителите, разделяйки ги на тестов и тренировачен сет и да ги подава на даден препоръчител.

# **Реализация, тестване/експерименти**

## ***Използвани технологии, платформи и библиотеки***

* ***Python*** – основен език за разработка, заради лесната интеграция с машинно самообучение и обработка на данни.
* ***Jupyter Notebook*** – за интерактивна разработка, експерименти и визуализации, особено при изследванне на данните.
* ***Pandas и NumPy*** – за четене на CSV файлове, трансформации на данни и ефективни операции върху масиви.
* ***Scikit-learn*** – за текстова векторизация (TF-IDF), K-Means клъстеризиране и други помощни методи.
* ***TensorFlow/Keras*** – за имплементация и трениране на Autoencoder модели.

## ***Реализация***

***Content-Based*:**

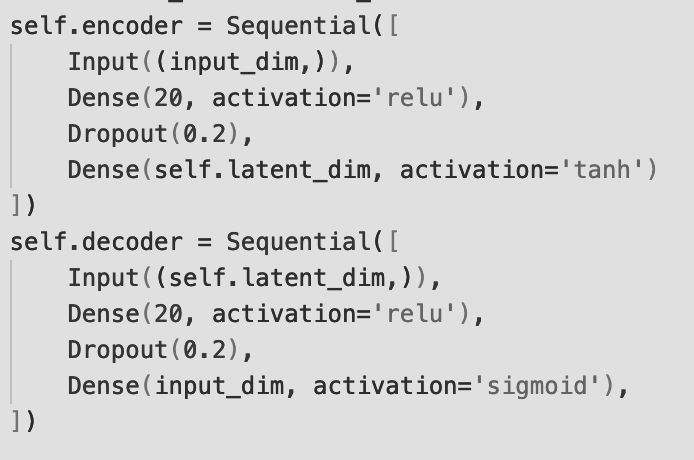
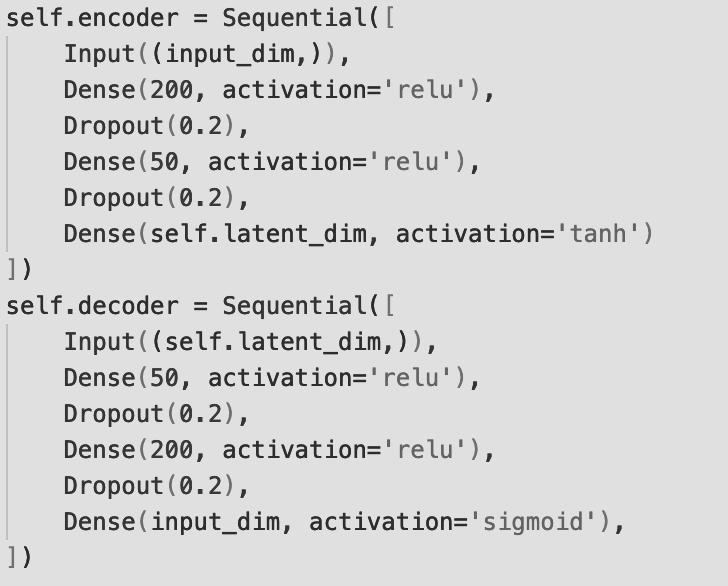
* Създаване на вектор (*TF-IDF* + *one-hot*) за всяка игра.
* Средно на векторите на игрите, които потребителят харесва и търсене на най-близки със *cosine similarity*.

***K-Means:***

* Клъстеризиране на всички игри в 20 клъстера според данните (може би ще открие жанрове).
* Средно на векторите на игрите, които потребителят харесва и намиране на най-близкия клъстър - препоръка на игри от същия клъстър с най-малка евклидова дистанция.

***Autoencoder:***

* Малък и Голям модел.
* Обучение върху игрите вектори от данни, използвайки *cosine\_similarity*.
* Използване на закодирания вектор от енкодера за препоръчване чрез *cosine similarity* (другите функции дадоха много лоши резултати).

Малък модел: Голям модел:

**Тестване:**

* Избират се случайни потребители от сета.
* За всеки потребител се разделят данните на тренировачен и тестващ сет.
* На подадения препоръчител се дава тренировачни сет за потребителя и той трябва да върне няколко препоръки. Проверяваме дали препоръките са част от тест сетът.
* За всеки потребител тези стъпки се повтарят няколко пъти и се взема средното.
* Накрая функцията връща Precision, *Recall* и *F1*.

## Резултати:

| Алгоритъм | *Precision* | *Recall* | *F1* |
| --- | --- | --- | --- |
| *Content-Based* | ~2.5% | ~4.9% | ~2.7% |
| *K-Means* | ~1.8% | ~0.76% | ~0.94% |
| *Autoencoder* (малък) | ~1.1% | ~3% | ~1.5% |
| *Autoencoder* (голям) | ~0.8% | ~4% | ~1.3% |

*( За Autoencoder-ите това са най-добрите резултати от всички изпробвани варианти за хиперпараметрите )*

# **Заключение**

Реализирахме система за препоръки на Steam игри, базирана на няколко подхода. Тестовете с данни от реални потребители показаха, че *content-based* подходът работи най-добре. *K-Means* и *Autoencoder* решенията не успяха да надминат простото сравнение на *TF-IDF* векторите, вероятно поради естеството на данните и сравнително малките набори потребителска активност.

**Идеи за бъдещо развитие**

* Намиране на повече данни за потребителите.
* Изпроване на *collaborative-filtering* подход.
* Тунинг на *autoencoder* мрежите (други загуби, архитектури и т.н.).
* Хибриден модел, комбиниращ *content-based* с *user-based* показатели.

# **Използвана литература**

* Уикипедия - https://en.wikipedia.org/wiki/Recommender\_system
* Онлайн ресурси (документация на TensorFlow, Scikit-learn, Pandas