

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Czotter Benedek

Klaszterezési stratégiák alkalmazása RAG rendszerek hatékonyságának növelésére

Tudományos Diákköri Konferencia Dolgozat

Konzulens

Dr. Szűcs Gábor

BUDAPEST, 2025

Tartalomjegyzék

[Abstract 4](#_Toc211371475)

[1 Bevezetés 5](#_Toc211371476)

[1.1 A kutatás célja 5](#_Toc211371477)

[1.2 RAG-rendszerek szerepe a modern NLP-ben 5](#_Toc211371478)

[1.3 Hatékonysági kérdések nagy dokumentumkorpuszok esetén 6](#_Toc211371479)

[2 Elméleti háttér 8](#_Toc211371480)

[2.1 A RAG (Retrieval-Augmented Generation) elmélete 8](#_Toc211371481)

[2.1.1 RAG architektúra felépítése 8](#_Toc211371482)

[2.1.2 Dokumentum-visszakeresés alapjai 9](#_Toc211371483)

[2.2 Embeddingek és vektorterek 10](#_Toc211371484)

[2.2.1 Sentence-transformerek működése és hatásuk a reprezentációra 10](#_Toc211371485)

[2.2.2 Távolságmértékek 10](#_Toc211371486)

[2.2.3 Vektorindexelési módszerek áttekintése 11](#_Toc211371487)

[2.3 Klaszterezés az információkinyerésben 11](#_Toc211371488)

[2.3.1 Klaszterezés célja 11](#_Toc211371489)

[2.3.2 Offline és online klaszterezés 11](#_Toc211371490)

[3 Adathalmaz és előfeldolgozás 13](#_Toc211371491)

[3.1 A SQuAD adathalmaz 13](#_Toc211371492)

[3.2 Szövegfeldolgozás és chunking 13](#_Toc211371493)

[3.3 Embedding generálás 14](#_Toc211371494)

[4 Klaszterezési módszerek implementációja és integrálása a retrieval folyamatba 16](#_Toc211371495)

[4.1 Centroid-alapú retrieval logika 17](#_Toc211371496)

[4.2 Online klaszterezés helye a retriveal folyamatban 18](#_Toc211371497)

[4.2.1 A pipeline technikai felépítése 18](#_Toc211371498)

[5 Eredmények és összehasonlítás 21](#_Toc211371499)

[6 Összegzés és kitekintés 22](#_Toc211371500)

[Utolsó simítások 23](#_Toc211371501)

[Irodalomjegyzék 24](#_Toc211371502)

[Függelék 25](#_Toc211371503)

Összefoglaló

A Retrieval-Augmented Generation (RAG) modellek az utóbbi években kulcsszerepet kaptak a nagynyelvi modellek (LLM-ek) tudásbővítésében és naprakész információkkal való ellátásában. E rendszerek lényege, hogy a nyelvi modell válaszadási folyamata előtt releváns dokumentumokat vagy dokumentumrészleteket („chunkokat”) keresnek ki egy nagyméretű tudásbázisból embedding-alapú hasonlóságmérés segítségével. A jelenlegi gyakorlatban a legtöbb RAG-megoldás a teljes embedding-halmazt közvetlenül használja a hasonlósági kereséshez, ami azonban pontossági, redundancia- és futásidőbeli korlátokat eredményezhet. A dolgozat célja annak vizsgálata, hogy klaszterezési technikák integrálásával miként növelhető a RAG-rendszerek hatékonysága, pontossága és robusztussága; hiszen hogyha a dokumentumok embeddingjeit először klaszterekbe rendezzük, majd a felhasználói lekérdezés embeddingjét először klaszterszinten vetjük össze velük, a keresés jelentősen gyorsabbá és relevánsabbá válhat. A dolgozat bemutatja a különböző klaszterezési algoritmusok teljesítményét és alkalmazhatóságát eltérő embedding-reprezentációkon, valamint azt, hogy hogyan lehet lehetővé tenni új dokumentumok folyamatos integrálását a tudásbázisba a teljes újraklaszterezés költsége nélkül. A kutatás eredménye egy klaszterezésen alapuló, adaptív retrieval-pipeline kialakítása, amely a hagyományos RAG-architektúrákhoz képest magasabb hatékonyságot, jobb pontosságot és fokozott robusztusságot biztosít. A kutatás eredményei nemcsak elméleti szempontból járulnak hozzá a RAG-rendszerek fejlődéséhez, hanem gyakorlati alkalmazásokban is közvetlenül hasznosíthatók lesznek.

Abstract

Retrieval-Augmented Generation (RAG) models have become key components in recent years for enhancing the knowledge base of large language models (LLMs) and providing them with up-to-date information. The core idea behind these systems is that, before the language model generates a response, they retrieve relevant documents or document segments (“chunks”) from a large knowledge base using embedding-based similarity search. In current practice, most RAG solutions directly use the entire embedding set for similarity search, which can lead to limitations in accuracy, redundancy, and runtime performance. The aim of this thesis is to investigate how the integration of clustering techniques can improve the efficiency, accuracy, and robustness of RAG systems. By first organizing document embeddings into clusters and then comparing the query embedding at the cluster level, the retrieval process can become significantly faster and more relevant. The thesis explores the performance and applicability of various clustering algorithms on different embedding representations and examines how new documents can be continuously integrated into the knowledge base without the cost of full re-clustering. The outcome of the research is the design of a clustering-based adaptive retrieval pipeline that provides higher efficiency, better accuracy, and increased robustness compared to traditional RAG architectures. The results of this research contribute not only to the theoretical advancement of RAG systems but are also directly applicable in practical implementations.

# 1 Bevezetés

## A kutatás célja

A mesterséges intelligencia fejlődésének egyik legmeghatározóbb területe az utóbbi években a természetes nyelv feldolgozás (Natural Language Processing, NLP) volt. A nagyméretű nyelvi modellek (Large Language Models, LLM-ek) képesek emberi szintű szövegértésre és -generálásra, azonban működésük korlátozott, ha a feladatukhoz szükséges információ nem szerepel a modell paramétereiben. A Retrieval-Augmented Generation (RAG) megközelítés ezt a problémát kezeli azáltal, hogy a szövegalkotás előtt a modell külső dokumentumokból keres releváns kontextust, és azt integrálja a válasz generálásába.

Jelen kutatás célja a retrieval-lépés hatékonyságának növelése. Nagyméretű dokumentumkorpuszok (pl. Wikipédia, tudományos cikkgyűjtemények vagy QA-adathalmazok) esetén a releváns információ visszakeresése rendkívül számításigényes feladat, hiszen minden lekérdezéshez több tízezer, sőt akár milliónyi szövegrész (chunk) vektoros reprezentációját kell összehasonlítani. Ez a folyamat a RAG-rendszerek egyik legszűkebb keresztmetszete, amely korlátozza azok valós idejű vagy online alkalmazását.

A dolgozat célja ezért a RAG-rendszerek hatékonyabbá tétele online klaszterezési módszerek integrálásával, különösen a centroid-alapú keresés alkalmazásán keresztül. A kutatás során bemutatásra kerül, hogyan csökkenthető a visszakeresési idő anélkül, hogy a pontosság jelentősen romlana, és milyen kompromisszum érhető el a gyorsaság és információvesztés között. A munka kísérleti platformja a SQuAD (Stanford Question Answering Dataset) ‎[1] adathalmaz, amely jól mérhető alapot biztosít a retrieval-modulok összehasonlításához.

## RAG-rendszerek szerepe a modern NLP-ben

A modern NLP-ben a RAG-rendszerek kulcsszerepet töltenek be az adatvezérelt tudás-hozzáférés, a forráshivatkozással alátámasztott szövegalkotás, valamint a megbízhatóbb generatív mesterséges intelligencia kialakításában. Széles körben alkalmazzák őket keresés-alapú chatbotokban, tudásmenedzsment rendszerekben, valamint olyan feladatokban, ahol a pontosság és az aktualitás kiemelt fontosságú. A RAG tehát nemcsak technológiai újítás, hanem paradigmaváltás is, hiszen az intelligens nyelvi rendszerek egyre inkább a „tudásra épülő”, nem pedig kizárólag a „nyelvre tanított” modellek irányába fejlődnek.

Míg korábban a modellek statikus tudásra támaszkodtak, a RAG megközelítés lehetővé teszi, hogy a rendszerek folyamatosan bővítsék és frissítsék tudásukat anélkül, hogy újratanítást igényelnének. Ez különösen fontos a gyorsan változó információs környezetekben, például a jogi, tudományos vagy üzleti szférában, ahol a relevancia és az aktualitás kulcstényezők. A modern NLP kutatásában a RAG így nemcsak egy hatékony technikai megoldás, hanem a mesterséges intelligencia átláthatóbbá és megbízhatóbbá tételének egyik legfontosabb irányvonala.

## Hatékonysági kérdések nagy dokumentumkorpuszok esetén

A RAG-rendszerek egyik legkritikusabb gyakorlati problémája a retrieval-lépés skálázhatósága. Nagy dokumentumkorpuszok esetén, ahol több millió szövegrész (chunk) kerül embedding formába, a keresés időigénye drámaian megnő. Egyetlen lekérdezés esetében akár több százezer koszinusz-távolság számítás is történhet, ami valós idejű alkalmazásokban (pl. kérdés–válasz rendszerekben) nem megengedhető. ‎[3]

A keresés gyorsítása több irányból közelíthető meg. Az egyik megoldás a vektorindexelés, például a FAISS, HNSW vagy ScaNN algoritmusok alkalmazásával, amelyek különböző approximációs módszerekkel (pl. kvantizáció, graf-alapú közelítés) csökkentik a szükséges összehasonlítások számát. Ezek az indexek azonban gyakran statikusak, a klasszikus FAISS indexek nem támogatják egyáltalán a dinamikus frissítést, emiatt, ha új dokumentumok kerülnek be, gyakran újra kell építeni az indexet, hogy megmaradjon az optimális teljesítmény. A HNSW egy dinamikus gráfstruktúra, amely hatékonyan támogatja új pontok hozzáadását, azonban a törlés és a tömeges frissítés továbbra is problémás. A ScaNN hasonlóan a FAISS-hoz szintén nem támogatja jól a folyamatos indexfrissítést.

Ezzel szemben a klaszterezésen alapuló megközelítések lehetőséget adnak a keresési tér intelligens felosztására. Az adathalmaz vektoraiból képzett klaszterek centroidjai egyfajta reprezentatív középpontot képeznek, amelyek alapján előszűrés végezhető, így a keresés először csak a centroidok között történik, majd a legközelebbi klaszter(ek)en belül folytatódik a részletesebb keresés. Ez a kétlépcsős folyamat jelentősen csökkenti az összehasonlítások számát, miközben a releváns találatok többsége továbbra is megtalálható marad.

A hatékonyság kulcsa a pontosság és sebesség közötti kompromisszum megtalálása. Ha túl kevés klasztert használunk, a keresés továbbra is lassú marad, ha viszont túl sokat, nő a hibás kizárások aránya. A jelen kutatás egyik fő célja ezért a paramétertér (klaszterszám, top-k klaszterek száma, top-n chunk visszaadása) grid search alapú optimalizálása, különböző embedding-méretek mellett.

További kérdést vet fel az online klaszterezés lehetősége. Míg az offline kmeans vagy MiniBatchKMeans csak teljes újratanítással tud reagálni az új adatokra, az online klaszterezési megoldások adaptív modelljei folyamatosan képesek frissíteni a centroidokat a beérkező minták alapján.‎[4]‎[5] Ez a megközelítés elméletileg lehetővé teszi a RAG-rendszerek folyamatosan tanuló retriever komponensének megvalósítását, olyat, amely idővel egyre jobban illeszkedik a beérkező kérdések és dokumentumok eloszlásához.

A dolgozat későbbi fejezeteiben bemutatásra kerül, hogy az ilyen online klaszterezési módszerek hogyan illeszthetők a RAG-rendszerek retrieval folyamatába, és milyen teljesítményt nyújtanak a hagyományos, minden lekérdezéskor teljes vektortérben végzett kereséssel szemben. A cél egy olyan robusztus, adaptív és gyors retrieval-architektúra megalkotása, amely a valós idejű RAG-rendszerek egyik kulcskomponensévé válhat.

# 2 Elméleti háttér

A modern természetes nyelvfeldolgozás és információkinyerés területén az utóbbi évek egyik jelentős előrelépése a Retrieval-Augmented Generation modell alkalmazása. A RAG lényege, hogy a generatív modellek nem csupán a tanult mintákból próbálnak választ adni, hanem aktívan hivatkozásokat és információkat is keresnek a rendelkezésre álló dokumentumtárakban. Ez a megközelítés különösen hasznos olyan helyzetekben, ahol a pontos és friss információ kritikus, mivel a generált válaszok pontossága és relevanciája jelentősen növelhető a visszakeresett dokumentumok felhasználásával. A következő alfejezetekben részletesen bemutatom a RAG architektúráját, a dokumentum-visszakeresés elméleti alapjait, az embeddingek és vektorterek szerepét, valamint a klaszterezés lehetséges alkalmazását az információkinyerésben.

## 2.1 A RAG (Retrieval-Augmented Generation) elmélete

A RAG modell a mesterséges intelligencia két meghatározó komponensét, a dokumentum-visszakeresést (retrieval) és a szöveggenerálást (generation), egyesíti egy egységes keretrendszerben. A modell elsődleges célja, hogy a felhasználói lekérdezésekre adott válaszokat ne kizárólag a nyelvi modell előzetesen betanított paramétereiből származó tudás alapján állítsa elő, hanem azokat kiegészítse külső forrásokból, például dokumentumtárakból vagy tudásbázisokból származó, aktuális és releváns információkkal.

A RAG működése egy kétfázisú folyamatként értelmezhető: az első szakaszban a rendszer a bemeneti lekérdezés alapján releváns dokumentumokat azonosít a rendelkezésre álló adatforrásokban, majd a második szakaszban ezen visszakeresett szövegeket kontextuális inputként felhasználva generál egy koherens, tartalmilag megalapozott választ. Ennek eredményeként a modell képes egyesíteni a retrieval-alapú tudásbővítés előnyeit a generatív nyelvi modellek rugalmas válaszképességével.

### 2.1.1 RAG architektúra felépítése

A képen képernyőkép, diagram látható

Automatikusan generált leírás

1. ábra - RAG rendszer felépítése

A Retrieval-Augmented Generation rendszerek, amint az az 1. ábrán is szemléltetésre kerül, három alapvető komponensből épülnek fel. Az első elem a dokumentumtár, amely a modell számára releváns szöveges források gyűjteményét tartalmazza. Ezen források körébe tartozhatnak tudományos cikkek, termékdokumentációk, adatbázis-bejegyzések, valamint különféle webes tartalmak is.

A második fő komponens a visszakereső modul (retrieval module), amelynek feladata a felhasználói lekérdezés alapján a legrelevánsabb dokumentumok azonosítása. Ennek során a lekérdezést és a dokumentumokat vektoros reprezentációvá (embedding) alakítják, majd a vektortérben mért hasonlóság (például koszinusz-távolság vagy euklideszi távolság) alapján meghatározzák a legközelebbi, azaz leginkább releváns találatokat.

A harmadik komponens a generatív modul (generation module), amely a visszakeresett dokumentumokból származó kontextuális információ felhasználásával állítja elő a végső választ. Ezáltal a modell nem csupán a saját neurális súlyaiban kódolt, előzetesen tanult mintázatokra támaszkodik, hanem az aktuálisan elérhető és releváns tudáselemeket is integrálja a válaszgenerálás folyamatába. Ennek eredményeként a RAG rendszerek válaszai nagyobb pontosságot, kontextuális koherenciát és információs megbízhatóságot mutatnak a hagyományos, kizárólag generatív alapú nyelvi modellekhez képest.

Jelen kutatás célja a RAG rendszerek visszakereső moduljának fejlesztése, különös tekintettel a klaszterezési technikák alkalmazására a dokumentum-visszakeresés pontosságának és hatékonyságának javítása érdekében. A kutatás további célja, hogy feltárja, miként használhatók fel nem felügyelt tanulási módszerek, különösen a dokumentum-embeddingek klaszterezése, a releváns információk strukturáltabb és gyorsabb előhívásának támogatására.

### 2.1.2 Dokumentum-visszakeresés alapjai

A dokumentum-visszakeresés célja, hogy egy adott lekérdezéshez a legrelevánsabb dokumentumokat biztosítsa. A folyamat több, egymásra épülő lépésből áll. Elsőként a dokumentumokat és a lekérdezést tokenizálják, normalizálják és chunkolják, azaz kisebb egységekre bontják. Ezt követően a szövegeket numerikus embeddingekké alakítják, amelyek lehetővé teszik a gépi hasonlóságmérést. A lekérdezés embeddingjét a dokumentumtár embeddingjeivel összehasonlítva meghatározzák a legrelevánsabb dokumentumokat, gyakran a koszinusz hasonlóság vagy az euklideszi távolság mérőszámai alapján. Végül a kiválasztott dokumentumokat relevancia szerint rangsorolják, biztosítva, hogy a generatív modell a lekérdezés szempontjából legértékesebb információkhoz férjen hozzá.

## 2.2 Embeddingek és vektorterek

### 2.2.1 Sentence-transformerek működése és hatásuk a reprezentációra

Az embeddingek a természetes nyelvi szövegek numerikus, vektoralapú reprezentációi, amelyek lehetővé teszik a gépi rendszerek számára a szövegek közötti hasonlóságok kvantitatív mérését és a hatékony vektoralapú keresést. Ezen ábrázolások nem csupán a szavak szerinti egyezést képesek megragadni, hanem a szövegek jelentésbeli és kontextuális kapcsolatait is képesek modellezni. Az embeddingek így olyan dimenziókat hoznak létre, amelyek a nyelvi mintázatokat, szemantikai összefüggéseket és a kontextuális információkat kódolják, lehetővé téve a gépek számára, hogy a jelentésbeli hasonlóságokat is figyelembe véve rangsorolják vagy csoportosítsák a dokumentumokat.

A sentence-transformerek kifejezetten mondat- és dokumentumszintű embeddingek előállítására szolgálnak. Ezek a modellek a bemeneti szöveget tokenizálják, majd a tokeneket belső reprezentációkká alakítják, amelyeket pooling műveletek segítségével egy fix hosszúságú vektorrá egyesítenek. Ennek eredményeként a jelentésükben hasonló mondatok embeddingjei a vektortérben közel helyezkednek el egymáshoz, ami elősegíti a dokumentum-visszakeresés, a klaszterezés és egyéb jelentésalapú elemzések pontosságát, valamint minimalizálja a releváns információk kihagyásának kockázatát. A sentence-transformerek így közvetlenül támogatják a gazdag, szemantikai információt hordozó reprezentációk létrehozását a természetes nyelvi adatok feldolgozásához.

### 2.2.2 Távolságmértékek

Az embedding vektorok közötti hasonlóság vagy távolság mérésére több módszer is létezik. A koszinusz hasonlóság (cosine similarity) a két vektor közötti szöget méri, ami különösen jól működik a nagyméretű, normalizált embeddingeknél. Az euklideszi távolság (euclidean distance) a vektorok geometriai távolságát adja meg, míg a Manhattan távolság (Manhattan distance) az egyes koordináták abszolút különbségeinek összegét használja. A skaláris szorzat (dot product) lineáris kapcsolatot mér, amelyet gyakran neurális hálózatokban alkalmaznak.

A képen Betűtípus, szöveg, fehér, diagram látható

Automatikusan generált leírás

1. egyenlet - Koszinusz hasonlóság számítási módja ‎‎[2]

A továbbiakban a munkám során az embedding vektorokat a koszinusz hasonlóság segítségével hasonlítottam össze. A teljesítmény növelése érdekében minden vektort L2-normalizáltam, vagyis úgy skáláztam, hogy a hosszúságuk 1 legyen. A koszinusz hasonlóság a vektorok közötti szöget méri, nem a nagyságukat; ha a vektorok különböző hosszúságúak, a nagyobb vektorok torzíthatják az eredményt. L2-normalizálás után a koszinusz hasonlóság két embedding között egyszerűen a skaláris szorzatra redukálódik, így a számítás is egyszerűsödik.

### 2.2.3 Vektorindexelési módszerek áttekintése

A nagy mennyiségű embedding hatékony és gyors keresése alapvető kihívást jelent a modern információ-visszakeresési rendszerekben. A hagyományos, ún. brute-force keresés során minden dokumentum embeddingjét összehasonlítják a lekérdezés embeddingjével, ami ugyan garantálja a pontos találatokat, de nagyméretű adatbázisok esetén rendkívül idő- és erőforrás-igényes megoldást jelent.

A gyakorlatban ezért gyakran alkalmaznak közelítő legközelebbi szomszéd (Approximate Nearest Neighbor, ANN) algoritmusokat, amelyek jelentősen felgyorsítják a keresési folyamatot, miközben a találatok pontossága közelítő módon megőrződik. Ezek közé tartozik többek között a FAISS (Facebook AI Similarity Search), a HNSW (Hierarchical Navigable Small World graphs) és az Annoy, melyek hatékony adattárolási és keresési struktúrákat biztosítanak a nagyméretű vektorterekben.

TODO: leírni pontosan, hogy a FAISS-t próbálom

## 2.3 Klaszterezés az információkinyerésben

### 2.3.1 Klaszterezés célja

A klaszterezés alapvető célja, hogy a dokumentumokat vagy azok embedding-reprezentációit tematikus, szemantikai szempontból koherens csoportokba rendezzük. A klaszterezés során olyan csoportokat alakítunk ki, amelyekben a belső hasonlóság maximális, míg a különböző klaszterek közötti eltérés jelentős. Ennek eredményeként a tartalmilag hasonló dokumentumok a vektortérben közel helyezkednek el egymáshoz, míg a különböző témájú csoportok jól elkülönülnek egymástól.

Ez a struktúra több szempontból is előnyös a dokumentum-visszakeresési rendszerek számára. Egyrészt lehetővé teszi, hogy a keresési folyamat a releváns klaszterekre koncentrálódjon, így jelentősen csökkentve a szükséges számítási időt és növelve a keresés hatékonyságát. Másrészt támogatja a relevancia szerinti rangsorolást, mivel a klasztereken belüli dokumentumok tartalmilag egységesebbek. Továbbá a klaszterezés hozzájárul a redundancia minimalizálásához is, mivel a hasonló információkat tartalmazó dokumentumok egy csoportba kerülnek, így a felhasználó számára a bemutatott találatok változatosabb és informatívabb képet adnak a keresett témáról.

### 2.3.2 Offline és online klaszterezés

A klaszterezési folyamatok két alapvető módon valósíthatók meg: offline és online módszerekkel. Az offline klaszterezés esetén a klaszterek a teljes rendelkezésre álló adatállomány alapján előre létrejönnek, és csak ritkán frissülnek. Ez a megközelítés hatékony, ha az adatok viszonylag statikusak, mivel lehetővé teszi a számításigényes algoritmusok alkalmazását és a klaszterek alapos optimalizálását. Hátránya azonban, hogy nem képes gyorsan reagálni az új, dinamikusan érkező adatokra, így a klaszterek idővel elavulhatnak, és a keresési relevancia csökkenhet.

Ezzel szemben az online klaszterezés folyamatosan alkalmazkodik az új adatokhoz, lehetővé téve a klaszterek dinamikus frissítését valós időben. Ez különösen fontos olyan környezetekben, ahol az adatforrások gyorsan változnak, például hírek, közösségi média bejegyzések, chat-adatok vagy valós idejű dokumentumáramlások esetén. Az online klaszterezés lehetővé teszi, hogy a vektoralapú embeddingeken alapuló reprezentációk folyamatosan tükrözzék az aktuális szemantikai szerkezetet, így a releváns dokumentumok könnyebben és gyorsabban azonosíthatók.

A Retrieval-Augmented Generation rendszerekben az online klaszterezés különösen hasznos lehet azokban az esetekben, ahol a dokumentumállomány folyamatosan bővül vagy változik. Dinamikus adatforrások, például hírek, közösségi média bejegyzések vagy valós idejű chat-adatok esetén a klaszterek hagyományos, előre definiált struktúrái gyorsan elavulhatnak, és nem tükrözik megfelelően a friss tartalmak jelentését. Az online klaszterezés lehetővé teszi, hogy a klaszterek folyamatosan frissüljenek az új dokumentumokkal, így a hasonló tartalmak mindig egy csoportban jelenjenek meg, és a keresési folyamat a releváns klaszterekre koncentrálódhasson. Ennek eredményeként a rendszerek képesek valós időben alkalmazkodni a változó dokumentumhalmazokhoz, fenntartva a tartalmilag koherens és jól strukturált klasztereket.

# 3 Adathalmaz és előfeldolgozás

## 3.1 A SQuAD adathalmaz

A jelenlegi munkában a SQuAD (Stanford Question Answering Dataset) adathalmazt alkalmazzuk a RAG rendszer fejlesztéséhez és kiértékeléséhez. A SQuAD egy széles körben használt, nyílt forráskódú kérdés-válasz adathalmaz, amely tartalmazza a Wikipedia szövegekből származó kontextusokat és a hozzájuk tartozó kérdéseket, valamint a helyes válaszokat. Az adathalmaz két fő részből áll: a train és a validation szettből, amelyek segítségével a modell betanítható és validálható. A train és validációs adatokat a datasets könyvtár segítségével töltöttük be, majd Pandas DataFrame-be konvertáltuk a könnyebb feldolgozás érdekében. A kontextusokhoz egyedi context\_id azonosítót rendeltünk, ezzel biztosítva, hogy a kérdések és a hozzájuk tartozó szövegek könnyen összekapcsolhatók legyenek. A válaszokat a SQuAD struktúrából kinyertük, külön mezőkbe helyeztük (answer\_text, answer\_start), ezzel megkönnyítve a későbbi összehasonlítást és kiértékelést.

A SQuAD adathalmaz elsődlegesen arra szolgál, hogy a modellek képesek legyenek a kontextusból pontosan kiválasztani a kérdésre adott választ. A kontextusok hosszúsága változó, és gyakran több bekezdést tartalmaznak, ami kihívást jelent a nagy nyelvi modellek számára, különösen akkor, ha az egész dokumentumot egyszerre kellene feldolgozni. Emiatt a chunking és a szövegfeldolgozás kulcsfontosságú előfeldolgozási lépések, amelyek lehetővé teszik a nagyobb szövegek kezelhetőségét és a releváns információk kiemelését.

## 3.2 Szövegfeldolgozás és chunking

A szövegfeldolgozás első lépéseként a SQuAD adathalmazban található kontextusokat megtisztítottuk a nem kívánt karakterektől és a felesleges szóközöktől. Ez a lépés alapvető fontosságú, mivel az adathalmazban gyakran előfordulnak HTML-entitásokból származó szimbólumok, speciális karakterek, illetve többszörös sortörések és tabulátorok, amelyek zavarhatják a későbbi nyelvi modellek működését. A tisztításhoz reguláris kifejezéseket (regex) alkalmaztunk, amelyek segítségével kiszűrtük a nem alfanumerikus karaktereket, valamint egységesítettük a whitespace karakterek használatát. A szövegeket először megtisztítottuk az idegen szimbólumoktól, majd egyetlen szóközre redukáltuk az egymás után következő üres karaktereket, végül eltávolítottuk a szöveg elején és végén található felesleges szóközöket.

A tisztított szövegeket ezt követően chunkolási eljárásoknak vetettük alá, amelyek célja a hosszú kontextusok kisebb, jól kezelhető egységekre bontása volt. A chunkolás azért szükséges, mert a nyelvi modellek, különösen a transformer alapú architektúrák, csak korlátozott hosszúságú bemenetet képesek feldolgozni, jellemzően 512–1024 token között. Ezért a dokumentumokat több, egymást részben átfedő szövegszegmensre bontottuk, hogy minden releváns információ elérhető maradjon a későbbi visszakeresés során.

A chunkolási eljárásnak kétféle megközelítését alkalmaztuk ‎[9]: csúszó ablakos (sliding window) és szemantikus (semantic) chunkolást. Az adathalmaz összesen 18891 különálló szövegből áll, melyek hossza 150 és 3700 karakter között mozog. A csúszó ablakos módszer esetében a szöveget előre meghatározott hosszúságú szegmensekre osztottuk, fix átfedéssel. A teljes szöveget TODO karakteres blokkokra bontotta, TODO karakteres lépésekben haladva, így biztosítva, hogy az egyes chunkok között TODO karakteres átfedés maradjon. Ez az átfedés lehetővé teszi, hogy a határokon átnyúló információk se vesszenek el, és a modell ne veszítse el a kontextus folytonosságát. A függvény ciklikusan haladt a szövegben, minden lépésben létrehozva egy új chunkot, amelyhez a kezdő és végpozíciót is eltárolta.

A szemantikus chunkolás ezzel szemben nem fix hosszúságú ablakokra épült, hanem a mondatok közötti hasonlóság alapján választotta el az egységeket. Ehhez a szöveget először mondatokra bontottuk az NLTK (Natural Language Toolkit) könyvtár *sent\_tokenize()* függvényével, majd minden mondatot vektorra képeztünk egy SentenceTransformer modell segítségével, konkrétan az „all-MiniLM-L6-v2” ‎[6]‎[7] architektúrával. A modell a mondatokat nagy dimenziós vektortérbe ágyazza, ahol a hasonló jelentésű mondatok közel kerülnek egymáshoz. Ezt kihasználva kiszámítottuk az egymást követő mondatok közötti koszinusz-hasonlóságot, és ha az érték meghaladta a 0,6-os küszöböt, a mondatokat egy közös chunkba sorolta. Amikor a hasonlóság ez alá a küszöbérték alá esett, új chunk kezdődött. A módszer így képes volt dinamikusan, a tartalmi folytonosság alapján meghatározni az optimális szeleteket, miközben figyelembe vette a maximális TODO karakteres hosszkorlátot is.

Mindkét chunkolási eljárás kimenetét egy egységes struktúrában tároltuk. Iteratívan végighaladtunk az adathalmaz kontextusain, és minden kontextusból több chunkot generáltunk. Minden chunkhoz hozzárendeltük a forrás kontextus azonosítóját, a chunk pozícióját, valamint egyedi azonosítóját.

A chunkolás eredményeként egy strukturált, egységesített és tisztított szöveges adathalmaz jött létre, amelyben minden kontextus több kisebb egységre bomlott. Ez a feldolgozott forma már közvetlenül alkalmas volt a mondatszintű embeddingek előállítására, illetve a dokumentum-visszakeresési modul betanítására.

## 3.3 Embedding generálás

A chunkok előállítása után a következő lépést az embeddingek (szövegbeágyazások) generálása jelentette, amelynek célja, hogy a szöveges tartalmakat numerikus vektortérbeli reprezentációkká alakítsuk. Ez a lépés alapvető fontosságú a későbbi szemantikus visszakeresési és klaszterezési folyamatokhoz, mivel a nyers szövegek nem kezelhetők közvetlenül gépi tanulási modellek vagy hasonlóságmérési eljárások számára. Az embeddingek segítségével a szövegek közötti szemantikus hasonlóság távolságmértékek (pl. koszinusz-hasonlóság) segítségével számszerűsíthetővé válik.

A beágyazások előállításához a SentenceTransformer keretrendszert alkalmaztuk, amely kifejezetten alkalmas természetes nyelvű szövegek kompakt és informatív vektorreprezentációinak előállítására. Annak érdekében, hogy a modellünk robosztusságát és általánosíthatóságát is vizsgálni tudjuk, több különböző architektúrával is elvégeztük a beágyazásokat. A választott modellek között szerepelt az „all-MiniLM-L6-v2”, amely 384 dimenziós vektorokat generál, valamint a „Snowflake/snowflake-arctic-embed-l-v2.0” ‎[8] modell, amely a chunkokat 1024 dimenziós térbe képezte le. Ez lehetővé tette, hogy a későbbi kísérletekben összehasonlítsuk a különböző dimenziójú embeddingek teljesítményét, mind a visszakeresési, mind a klaszterezési feladatokban.

Az embeddingek generálása során batch-szerű feldolgozást alkalmaztunk a memóriahasználat optimalizálása és a feldolgozási sebesség növelése érdekében. Minden feldolgozott chunk beágyazását elmentettük egy DataFrame struktúrában, illetve NumPy tömbként is, ami lehetővé teszi a hatékony, vektoralapú műveletek (mint például normalizálás, klaszterezés vagy valós idejű keresés) végrehajtását. Az így kapott vektorokat L2 normalizálással előfeldolgoztuk, hogy a koszinusz-hasonlóságon alapuló keresési és klaszterezési eljárások stabilabb és pontosabb eredményeket adjanak.

# 4 Klaszterezési módszerek implementációja és integrálása a retrieval folyamatba

A retrieval-alapú rendszerek, különösen a RAG architektúrák esetében, az egyik legfontosabb kérdés, hogy hogyan lehet hatékonyan és pontosan visszakeresni a releváns dokumentumokat vagy szövegrészeket a tudásbázisból. A hagyományos megközelítés szerint minden lekérdezés esetén a teljes adatbázist (vagy annak embeddingjeit) kell átvizsgálni, ami nagy adatméretek esetén számottevően lelassítja a válaszidőt. Ennek a problémának a kezelésére vezettük be a klaszterezés-alapú retrieval módszert, amelynek célja a keresési tér redukálása a reprezentatív centroidok segítségével. A módszer az offline és az online tanulási folyamatokban is kulcsszerepet játszik, különösen olyan környezetben, ahol az adatok folyamatosan frissülnek. A továbbiakban egy olyan módszert mutatok be, mellyel nagy mértékben felgyorsítható az információ visszakeresés sebessége a pontosság minimális csökkenése mellett, illetve bemutatásra kerül az is, hogyan lehet ezt a klaszterezés-alapú retrival módszert dinamikusan változó dokumentum halmazok esetén használni, biztosítva ezzel, hogy a rendszer mindig a legrelevánsabb és legaktuálisabb információt szolgáltassa a lekérdezésekhez.

A képen képernyőkép, diagram látható

Automatikusan generált leírás

2. ábra - Az általam készített RAG pipeline felépítése

A 2. ábrán látható architektúra dinamikusan változó dokumentum halmazok kezelésére épül. A folyamatosan érkező új dokumentumok áthaladnak a korábban említett adatelőkészítési lépéseken, majd elkészülnek a feldolgozott chunkokból a szövegbeágyazások. Ezt az online klaszterező algoritmus azonnal egy már meglévő vagy új klaszterhez rendeli, majd ezen klaszterek ismeretében indul a RAG alapját képező, hasonlóság alapú információ visszatérítés, melynek végeredményében a felhasználói bemenethez leginkább „közel” elhelyezkedő dokumentum részletek kerülnek meghatározásra. A továbbiakban részletesen bemutatom a centroid alapú és az online klaszterező megközelítéseket.

## 4.1 Centroid-alapú retrieval logika

A centroid-alapú retrieval egyik legfontosabb lépése a klaszterezés utáni centroidvektorok kiszámítása, amelyek a klaszterek középpontjait reprezentálják a nagy dimenziós embedding térben. Ezek a centroidok a klaszterben található dokumentum- vagy szövegembeddingek átlagát képezik, így minden klaszterhez egyetlen, reprezentatív vektor rendelhető. Matematikailag ez a folyamat úgy írható le, hogy az adott *n* darab embedding *e1, e2, ..., en* alapján a centroid vektor *c* az alábbi képlettel számolható:

Ez az egyszerű átlagolás ugyan lineáris művelet, de szemantikailag nagy jelentőséggel bír, hiszen a centroid a klaszter elemeinek közös irányát mutatja a vektortérben, azaz azt a pontot, amely a legjobban jellemzi a csoport tartalmi témáját. Másképpen fogalmazva: a centroid egy absztrakt reprezentációja annak a fogalmi mezőnek, amelyet a klaszter elemei körülírnak.

Amikor egy új lekérdezés érkezik, a hagyományos, minden dokumentumot átvizsgáló kereséssel szemben itt nem az összes embeddinggel, hanem csak a centroidokkal történik az első hasonlóság-számítás, amihez koszinusz hasonlóságot használtunk.

A lekérdezés embeddingje tehát minden centroidhoz viszonyítva kap egy hasonlósági értéket. Ebből a rendszer kiválasztja a legmagasabb értéket mutató néhány klasztert, majd csak ezekben a klaszterekben végez el részletes keresést. Ez a megközelítés drasztikusan csökkenti a számítási igényt, hiszen a centroidok száma tipikusan több nagyságrenddel kisebb, mint az eredeti chunkoké. Például, ha az adatbázis egymillió chunkból áll, de ezeket ezer klaszterre osztjuk, akkor a rendszer az első körben mindössze ezer hasonlóság-számítást végez, nem pedig egymilliót.

Ezzel a módszerrel egy kétszintű visszakeresési folyamat valósul meg. Az első szint a klaszterszintű szűrés, amely gyors, közelítő keresést biztosít. Itt a cél nem az, hogy a legpontosabb találatokat kapjuk, hanem hogy a keresési tér méretét jelentősen leszűkítsük. A második szint a chunk-szintű szűrés, amely az előző lépésben kiválasztott releváns klasztereken belül történik. Ebben a fázisban már minden chunk embeddinget közvetlenül összevetünk a lekérdezés embeddingjével, így ez a keresés sokkal pontosabb, ugyanakkor a csökkentett adathalmaz miatt továbbra is gyors marad.

Azután, hogy a rendszer azonosította a legjobb embeddingeket a legrelevánsabb klaszterekben a koszinusz hasonlóság alapján, a végső chunkok visszatérítése előtt csak azokat a chunkokat térítjük vissza, ahol a koszinusz hasonlóságuk meghalad egy előre definiált küszöbértéket. Ez a megközelítés biztosítja, hogy a felhasználónak csak a legrelevánsabb és legmegbízhatóbb információk érkezzenek, minimalizálva a zajt és a pontatlan találatokat. Erre azért van szükség, hogy abban az esetben, ha olyan felhasználói kérdés érkezik, amelyre a tudásbázisban nincs megfelelő válasz, ne térítsünk vissza olyat, amivel esetleg a későbbiekben félrevezetjük a retrieval architektúra felett elhelyezkedő intelligens rendszereket.

A kétszintű struktúra egyensúlyt teremt a sebesség és a pontosság között. A klaszterszintű előszűrés hatékonyan csökkenti a számítási komplexitást, míg a chunk-szintű finomítás garantálja, hogy a végső találatok valóban relevánsak maradjanak. A centroid-alapú retrieval további előnye, hogy könnyen skálázható. Új dokumentumok beérkezése esetén elegendő a hozzájuk tartozó embeddinget kiszámítani és a megfelelő klaszterhez rendelni, illetve a centroidot újraszámolni. Ezzel elkerülhető a teljes embeddingtér újratanítása.

## 4.2 Online klaszterezés helye a retriveal folyamatban

A hagyományos, statikus klaszterezési eljárásokkal szemben, amelyek egy rögzített adathalmazon futnak le egyszer, az online klaszterezés olyan dinamikus megközelítést valósít meg, amely képes a rendszerbe folyamatosan érkező új adatpontokat, jelen esetben új szöveges chunkokat vagy dokumentumokat, valós időben integrálni a meglévő klaszterstruktúrába. Ennek az adaptív viselkedésnek különösen nagy jelentősége van olyan retrieval pipeline-okban, amelyek élő adatfolyamokat, gyakran frissülő dokumentumkorpuszt vagy felhasználói generált tartalmat kezelnek. Ennek megoldására a pipeline-ba egy online klaszterezési réteget integráltunk, amely képes az új embeddingeket folyamatosan beilleszteni a meglévő klaszterstruktúrába, miközben adaptívan frissíti a klaszterek centroidjait. Ez az integráció teszi lehetővé a rendszer valós idejű tanulását és alkalmazkodását a folyamatosan változó adatkörnyezethez.

### 4.2.1 A pipeline technikai felépítése

A beérkező dokumentumokat az *3 Adathalmaz és előfeldolgozás* fejezetben leírtaknak megfelelően először feldolgozzuk, majd pedig az aktuálisan használt sentence transformer modell segítségével elkészítjük a chunkok beágyazásait. Az online klaszterezés központi célja, hogy minden újonnan érkező embeddinget hatékonyan beillesszen a megfelelő klaszterbe, vagy új klasztert hozzon létre, ha a hasonlósági feltételek ezt indokolják.

Az algoritmus működésének alapja a centroidalapú klaszterezés, ahol minden embeddinget a legközelebbi klaszterközépponthoz rendelünk egy választott távolságmérték alapján. Az implementáció támogatja az Euklideszi és a koszinuszos metrikát, így alkalmazható mind abszolút értékbeli, mind irányfüggő hasonlóságokra épülő feladatokban. A modell a tanulási folyamat során minden beérkező embedding esetén kiszámítja az vektor és a jelenlegi centroidok közötti távolságot, majd az embeddinget ahhoz a klaszterhez rendeli, amelyhez a legkisebb távolság tartozik. A hozzárendelés során Euklideszi távolság metrikát feltételezve minden *xi* beágyazási vektort ahhoz a klaszterhez rendelünk, amelynek középpontja *ri* a legközelebb esik hozzá:

Ahol *Ci* jelöli az embeddinghez tartozó klaszter indexét, melyet az embedding vektor és a legközelebb eső klaszter középpont euklideszi távolság négyzete alapján határozunk meg. Ezzel szemben koszinuszos metrika esetén a hasonlóságot az egységnormára hozott vektorok skalárszorzatából számítjuk.

A pipeline az induláskor a kezdeti centroidokat a kmeans++ ‎[10] inicializálási eljárással határozza meg, amely véletlenszerű, de távolságalapú szelekciót alkalmaz a kezdeti klaszterközéppontok eloszlásának optimalizálására. Ez a megoldás biztosítja, hogy a centroidok már a kezdeti fázisban reprezentálják az adathalmaz különböző régióit, így gyorsabb és stabilabb konvergenciát eredményez.

Az algoritmus egyik legfontosabb tulajdonsága a dinamikus klaszterszám-kezelés. Amennyiben egy új embedding vektor távolsága minden meglévő centroidtól meghalad egy előre definiált küszöbértéket, az algoritmus új klasztert hoz létre abból a pontból. Ez a mechanizmus lehetővé teszi, hogy a modell alkalmazkodjon az adatstruktúra változásaihoz, és új mintázatok megjelenésekor automatikusan bővítse a klaszterkészletét. Ugyanakkor beállítható egy maximális klaszterszám, amely megakadályozza a klaszterek korlátlan növekedését.

A képen szöveg, képernyőkép, diagram, virág látható

Automatikusan generált leírás

3. ábra - Minta példa az online klaszterező algoritmus működésének folyamatára

A klaszterközéppontok frissítése inkrementális statisztikai frissítési képlettel történik, amely egyensúlyt teremt a korábban látott és az újonnan beérkező adatok hatása között. Erre látható egy minta példa a 3. ábrán. Legyen *nold* a klaszter korábbi elemszáma, *m* az új adathalmaz mérete, *rold* a korábbi centroid, és *rbatch* az új pontok átlaga. A frissített klaszterközéppont a következőképpen számítható:

Tehát minden klaszter esetében az algoritmus kiszámítja az adott batch-hez tartozó pontok átlagát és varianciáját, majd ezeket az értékeket a korábbi klaszterparaméterekkel kombinálva pontosítja a centroid helyét és a klaszter szórását. Ez a megközelítés megőrzi a korábbi információt, ugyanakkor fokozatosan beépíti az új adatokat, így a rendszer folyamatosan tanul anélkül, hogy az előző állapotot elfelejtené.

Az online klaszterezés előnye, hogy nem igényel az egész adathalmaz memóriában tartását, és alkalmas folyamatosan változó környezetekhez.Azonban a korábban említettek alapján egyértelműen látszik az is, hogy ez a megközelítés is, minden más online klaszterező algoritmus, függ az adatok érkezési sorrendjétől, amely egy hátrányukat képezi. Ez a sorrendfüggőség különösen akkor válik kritikussá, ha az adatok nem véletlenszerű sorrendben, hanem például időben rendezve érkeznek. Ilyen esetekben a korai minták nagyobb hatással lehetnek a klaszterek kezdeti pozícióira, ami torzításhoz vezethet.

Ennek elkerülése végett bevezettük a modell egy további sajátosságát, amely a klaszterek összeolvasztása. Ez a mechanizmus akkor aktiválódik, ha két centroid közötti távolság egy előre megadott határérték alá csökken. Ilyen esetben a két klasztert súlyozott átlagolással egyesíti, miközben frissíti a varianciaértékeket. Ez a módszer segít elkerülni a redundáns vagy egymáshoz túl közeli klaszterek kialakulását, valamint elősegíti a stabilabb és értelmezhetőbb klaszterstruktúra fenntartását.

A távolságmetrika kiválasztásától függően az algoritmus automatikusan végrehajtja a szükséges normalizálást: koszinuszos metrika esetén a bemeneti vektorokat és a centroidokat egységnormára skálázza, biztosítva ezzel a hasonlósági mérés konzisztenciáját.

Az implementáció mindemellett egy state management mechanizmust is tartalmaz, amely lehetővé teszi a modell aktuális állapotának (centroidok, számlálók, varianciák, feldolgozott adatok száma) kinyerését és mentését. Ez különösen hasznos online tanulási környezetben, ahol a folyamat bármikor megszakítható és újraindítható anélkül, hogy az addigi tudás elveszne.

A centroidok frissítése és új embeddingek klaszterekbe sorolását követően elkezdődhet a retrieval fázis a korábban bemutatott módszer alapján. Először a felhasználói kérdést klaszter-szinten hasonlítjuk össze, majd csak ezt követően végzünk részletes keresést a legrelevánsabb klaszterekben szereplő embeddingek között. Végül pedig visszatérítjük az így kapott leghasonlóbb embedding vektorokat, melyek alapján a későbbiekben egy nagy nyelvi modell képes lehet a felhasználói kérdésre egy pontosított választ adni.

# 5 Eredmények és összehasonlítás

## 5.1 Klaszterezési módszerek hatékonyságának vizsgálata

Az eredmények bemutatását és a kiértékelési módszerek ismertetését a különböző klaszterezési algoritmusok hatékonyságának összehasonlító vizsgálatával kezdjük. A kísérletek célja annak feltárása volt, hogy az egyes megközelítések milyen mértékben képesek hatékonyan és stabilan kezelni a dinamikusan változó, nagyméretű adatbeáramlást. Ennek érdekében három különböző klaszterezési módszert elemeztünk és értékeltünk kvantitatív módon.

Az első vizsgált módszer az előző fejezetben részletesen bemutatott online K-Means algoritmus, amely az adatok folyamatos érkezéséhez adaptálódva frissíti a klaszterközéppontokat, és képes a korábbi tanulási eredmények megőrzése mellett az új információk integrálására.

A második vizsgált megközelítés a MiniBatchKMeans algoritmus, amely a klasszikus K-Means továbbfejlesztett, részben online működésre képes változata. Ez a módszer szintén batch-alapú feldolgozást alkalmaz, azonban nem támogatja a klaszterszám dinamikus módosítását a beérkező adatok függvényében. Ennek következtében a modell merevebb, és kevésbé alkalmas az olyan környezetekben történő alkalmazásra, ahol az adatok eloszlása időben változhat, vagy új minták jelenhetnek meg.

A harmadik elemzett módszer az alap K-Means algoritmus, amely a legelterjedtebb klaszterezési eljárások közé tartozik. Bár egyszerű és hatékony statikus környezetben, hátránya, hogy teljes újratanítást igényel, amennyiben új adatok érkeznek, így valós idejű vagy folyamatosan frissülő rendszerek esetén kevéssé alkalmazható.

A vizsgálatok során kipróbálásra került egy negyedik módszer is, a River keretrendszerben található KMeans algoritmus, amely kifejezetten online, streaming jellegű adatok feldolgozására lett tervezve. Ez az algoritmus a learn\_one() függvény segítségével képes új adatpontok egyenkénti megtanulására, ami előnyös lehet folyamatosan érkező, valós idejű adatfolyam esetén. Ugyanakkor ez a megközelítés a jelen munka szempontjából nem bizonyult célszerűnek, mivel nem teszi lehetővé az adatok batchenkénti feldolgozását, illetve nem támogatja egy nagy, előre rendelkezésre álló adathalmazról történő inicializálást ‎[11].

Mivel kutatásom egyik központi célja egy olyan klaszterezési megoldás kialakítása volt, amely nagy méretű, előre ismert tudásbázisra épül, és képes annak dinamikus, folyamatos frissítésére a későbbiekben beérkező adatok alapján, ezért elengedhetetlen követelmény volt a batch alapú feldolgozás és az előzetes inicializálhatóság. Ennek megfelelően a River KMeans algoritmus a további elemzésekből kizárásra került.

Először vizsgáljuk meg a különböző algoritmusokat futásidő szempontjábó. Ehhez a már bemutatott SQuAD adathalmazt használjuk, amiből a chunkolást követően 384 és 1024 dimenziós szövegbeágyazásokat is készítettünk. Összesen 84007 darab embeddingünk van 18891 különböző dokumentumból, amiket most klaszterezni szeretnénk.

A képen szöveg, sor, Diagram, lejtő látható

Automatikusan generált leírás

4. ábra - KMeans, MiniBatchKMeans és online KMeans futásideje 384 és 1024 dimenziós embeddingeken különböző számú klasztert feltételezve.

A képen sor, szöveg, Diagram, diagram látható

Automatikusan generált leírás

5. ábra - A MiniBatchKMeans és online KMeans futásideje 384 és 1024 dimenziós embeddingeken a maximális indulási klaszterszám függvényében. (4. ábra részlete)

Ahogy az a 4. ábrán is jól látható a KMeans algoritmus jóval lassabban képes elvégezni a klaszterezést, mint az online vagy MiniBatchKMeans. Ez nem meglepő eredmény, hiszen egy klasszikus KMeans működése során minden iterációban az összes adatpontot újraértékeli, hogy kiszámolja a távolságokat a centroidokhoz, majd újra frissítse azokat. Ez azt jelenti, hogy az egész adathalmaznak egyszerre kell elérhetőnek lennie a memóriában, ami nagy adatok esetén komoly korlátozó tényező. Vizsgáljuk meg most az 5. ábra alapján a MiniBatchKMeans és online KMeans algoritmusokat futásidő szempontjából. Itt az látható, hogy egészen addig, amíg nem érünk el nagyobb kiindulási klaszterszámot, a két algoritmus ugyanolyan eredményt ér el. Azonban, amint a klaszterszám megnövekszik a MiniBatchKMeans futásideje elkezd megnőni. Ezzel szemben az online KMeans algoritmus még nagy klaszterszámok esetén is kiegyensúlyozott teljesítményt nyújt futásidőben.

# 6 Összegzés és kitekintés

# Utolsó simítások

Miután elkészültünk a dokumentációval, ne felejtsük el a következő lépéseket:

* Kereszthivatkozások frissítése: miután kijelöltük a teljes szöveget (Ctrl+A), nyomjuk meg az F9 billentyűt, és a Word frissíti az összes kereszthivatkozást. Ilyenkor ellenőrizzük, hogy nem jelent-e meg valahol a "Hiba! A könyvjelző nem létezik." szöveg.
* Dokumentum tulajdonságok megadása: a dokumentumhoz tartozó meta adatok kitöltése (szerző, cím, kulcsszavak stb.). Erre való a Dokumentum tulajdonságai panel, mely a Fájl / Információ / Tulajdonságok / Dokumentumpanel megjelenítése úton érhető el.
* Kinézet ellenőrzése PDF-ben: a legjobb teszt a végén, ha PDF-et készítünk a dokumentumból, és azt leellenőrizzük.

Irodalomjegyzék

1. Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, Percy Liang, (2016.06.16.),SQuAD: 100,000+ Questions for Machine Comprehension of Text, ArXiv, [arXiv:1606.05250](https://arxiv.org/abs/1606.05250)
2. Varun, (2020.09.27.), Cosine similarity: How does it measure the similarity, Maths behind and usage in Python, towardsdatascience.com ,<https://towardsdatascience.com/cosine-similarity-how-does-it-measure-the-similarity-maths-behind-and-usage-in-python-50ad30aad7db/>
3. Shailja Gupta (Carnegie Mellon University, USA) Rajesh Ranjan (Carnegie Mellon University, USA) Surya Narayan Singh (BIT Sindri, India), (2024.10.03.), A Comprehensive Survey of Retrieval-Augmented Generation (RAG): Evolution, Current Landscape and Future Directions, ArXiv, <https://arxiv.org/pdf/2410.12837>
4. Ashour, Wesam & Fyfe, Colin. (2008). Online clustering algorithms. International journal of neural systems. 18. 185-94. 10.1142/S0129065708001518.
5. Li Juanzi, Hu Linmei, Ouyang Tinghui, Alkawsi Gamal Abdulnaser, (2006), Online Clustering, ScienceDirect, <https://www.sciencedirect.com/topics/computer-science/online-clustering>
6. Reimers, Nils and Gurevych, Iryna, (2019. november), Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks, Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, <https://arxiv.org/abs/1908.10084>
7. The Devs from Zilliz, (2025), How do Sentence Transformers relate to large language models like GPT, and are Sentence Transformer models typically smaller or more specialized?, Milvus, <https://milvus.io/ai-quick-reference/how-do-sentence-transformers-relate-to-large-language-models-like-gpt-and-are-sentence-transformer-models-typically-smaller-or-more-specialized>
8. Puxuan Yu, Luke Merrick, Gaurav Nuti, Daniel Campos, (2024.12.04.), Snowflake’s Arctic Embed 2.0 Goes Multilingual: Empowering Global-Scale Retrieval with Inference Efficiency and High-Quality Retrieval, snowflake, <https://www.snowflake.com/en/engineering-blog/snowflake-arctic-embed-2-multilingual/>
9. Shanmukha Ranganath, (2024.10.05.), RAG 101: Chunking Strategies, towardsdatascience.com, <https://towardsdatascience.com/rag-101-chunking-strategies-fdc6f6c2aaec/>
10. Lance Galletti, (2024.02.22.), Kmeans ++ From Scratch, Medium, <https://medium.com/@gallettilance/kmeans-from-scratch-24be6bee8021>
11. Hoang-Anh Ngo, (2022 september), ONLINE CLUSTERING: ALGORITHMS, EVALUATION, METRICS, CHALLENGES, APPLICATIONS AND BENCHMARKING WITH RIVER, @Télécom Paris, IP Paris, The University of Waikato, Artificial Intelligence Institiut, <https://a3nm.net/work/seminar/slides/20220927-ngo.pdf>

Függelék

Általános információk

A diplomaterv szerkezete:

1. Diplomaterv feladatkiírás
2. Címoldal
3. Tartalomjegyzék
4. A diplomatervező nyilatkozata az önálló munkáról és az elektronikus adatok kezeléséről
5. Tartalmi összefoglaló magyarul és angolul
6. Bevezetés: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása
7. A feladatkiírás pontosítása és részletes elemzése
8. Előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések
9. A tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása
10. A megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek
11. Esetleges köszönetnyilvánítások
12. Részletesés pontos irodalomjegyzék
13. Függelék(ek)

Felhasználható a következő oldaltól kezdődő Diplomaterv sablon dokumentum tartalma. Ügyeljen a tanszék, a hallgató, a konzulens nevét és a beadás évét jelölő szövegdobozokra, mert azokra külön ki kell adni a frissítést. A mezők tartalma a sablonban a dokumentum adatlapja alapján automatikusan kerül kitöltésre (Fájl/Információ/Tulajdonságok/Speciális tulajdonságok).

A diplomaterv szabványos méretű A4-es lapokra kerüljön. Az oldalak tükörmargóval készüljenek (mindenhol 2.5cm, baloldalon 1cm-es kötéssel). Az alapértelmezett betűkészlet a 12 pontos Times New Roman, másfeles sorközzel.

Minden oldalon - az első négy szerkezeti elem kivételével - szerepelnie kell az oldalszámnak.

A fejezeteket decimális beosztással kell ellátni. Az ábrákat a megfelelő helyre be kell illeszteni, fejezetenként decimális számmal és kifejező címmel kell ellátni. A fejezeteket decimális aláosztással számozzuk, maximálisan 3 aláosztás mélységben (pl. 2.3.4.1.). Az ábrákat, táblázatokat és képleteket célszerű fejezetenként külön számozni (pl. 2.4. ábra, 4.2 táblázat vagy képletnél (3.2)). A fejezetcímeket igazítsuk balra, a normál szövegnél viszont használjunk sorkiegyenlítést. Az ábrákat, táblázatokat és a hozzájuk tartozó címet igazítsuk középre. A cím a jelölt rész alatt helyezkedjen el.

A képeket lehetőleg rajzoló programmal készítsék el, az egyenleteket egyenlet-szerkesztő segítségével írják le.

Az irodalomjegyzék szövegközi hivatkozása történhet a Harvard-rendszerben (a szerző és az évszám megadásával) vagy sorszámozva. A teljes lista névsor szerinti sorrendben a szöveg végén szerepeljen (sorszámozott irodalmi hivatkozások esetén hivatkozási sorrendben). A szakirodalmi források címeit azonban mindig az eredeti nyelven kell megadni, esetleg zárójelben a fordítással. A listában szereplő valamennyi publikációra hivatkozni kell a szövegben. Minden publikáció a szerzők után a következő adatok szerepelnek: folyóirat cikkeknél a pontos cím, a folyóirat címe, évfolyam, szám, oldalszám tól-ig. A folyóirat címeket csak akkor rövidítsük, ha azok nagyon közismertek vagy nagyon hosszúak. Internet hivatkozások megadásakor fontos, hogy az elérési út előtt megadjuk az oldal tulajdonosát és tartalmát (mivel a link egy idő után akár elérhetetlenné is válhat), valamint az elérés időpontját.

Fontos:

* a szakdolgozat készítő/diplomatervező nyilatkozata (a jelen sablonban szereplő szövegtartalommal) kötelező előírás Karunkon, ennek hiányában a szakdolgozat/diplomaterv nem bírálható és nem védhető!
* mind a dolgozat, mind a melléklet maximálisan 15 MB méretű lehet!

Jó munkát, sikeres szakdolgozat készítést ill. diplomatervezést kívánunk!