Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

TDK dolgozat

Huszti Dorottya

SZÖVEGDOKUMENTUMOK ÖSSZEFOGLALÓINAK AUTOMATIKUS ELŐÁLLÍTÁSA

KONZULENS

Dr. Szűcs Gábor

BUDAPEST, 2019

Tartalomjegyzék

1 Bevezetés

A mindennapokban keletkező nagy mennyiségű struktúrálatlan szöveges adat feldolgozása szerkezetéből és méretéből kifolyólag hagyományos eszközökkel már nehezen valósítható meg. Ezért egyre nagyobb hangsúly helyeződik a szövegbányászatra, amely olyan korszerű megoldásokat foglal magában, amelyek megkönnyítik a nyers adatokban rejlő összefüggések kinyerését. A szöveganalitika egyik legnagyobb kihívása a természetes nyelvű dokumentumok tartalmának összefoglalása, amely egy hosszú, összefüggő szöveg tartalmának rövid, de pontos leírását jelenti [1]. Ez egy olyan összetett feladat, amely a szöveg megértését és a releváns információk kinyerését egyidejűleg igényli.

A generált összefoglalókat két nagyobb csoportba sorolhatjuk azok előállítási mechanizmusa alapján. Egyfelől beszélhetünk kinyerés alapú összefoglalókról, ha kizárólag az eredeti szöveg szavainak és mondatainak felhasználásával készítjük el az összegzést [2]. Az így létrehozott rövidítés minden szava kizárólag a forrás szövegből származik. A másik megközelítés az úgynevezett absztrakt összefoglaló generálása. Ez a módszer sokkal inkább egyfajta szöveggenerálás a szöveg felépítésének vizsgálata alapján. Az összefoglaló létrehozásakor már nem korlátozódik az eredeti dokumentum szó- és mondatkészletére, hanem eddig nem látott szavakat is felhasználhat az új mondatok előállításához [3].

A dolgozatom célja különböző automatikus összefoglaló generáló rendszerek megtervezése és megvalósítása volt. Először bemutatom a szövegdokumentumok összefoglalását és annak jelentőségét, valamint a feladathoz felhasznált technológiákat. Ezt követően részletesen ismertetem a gépi tanuláson (mély neurális hálón) alapuló összefoglaló generáló rendszer felépítését, majd a kinyerés alapú összefoglaló generáló rendszereket, a tervezéskor hozott döntéseket és az elért eredményeket. A dolgozatomat végül a különböző összefoglaló készítő rendszerek bemutatásával, értékelésével és összevetésével zárom.

2 Szövegdokumentumok összefoglalóinak előállítási módszerei

2.1 Absztrakt és kinyerés alapú összefoglaló előállítása

Az összefoglalók két típusát az 1-es fejezetben ismertetett absztrakt és kinyerés alapú összefoglalók jelentik. Ezek előállítási mechanizmusukban, valamint az összefoglaló jellemzőiben is jelentősen eltérnek. A két irányzatot és a köztük lévő különbségeket a 2.1. táblázat illusztrálja.

Kinyerés	Absztrakt
Joseph and Mary rode on a donkey to **attend** the annual **event** in **Jerusalem**. In the city, **Mary** gave **birth to** a child named **Jesus**.	
Joseph and Mary attend event Jerusalem. **Mary birth Jesus**.	**Joseph and Mary** came **to Jerusalem** where **Jesus** was born.

2.1. táblázat: Kinyert és absztrakt összefoglaló összehasonlítása[1]

A kinyerés alapú összefoglalók előállítása a szöveg kulcsszavainak és kulcsmondatainak kigyűjtésével történik, amelyet kivonatnak nevezünk. Ez a szövegelemek tartalomleíró képességének megkülönböztetését igényli, amelyet három fő lépés köré szerveznek: szöveg kisebb vizsgálandó egységekre bontása, szöveg elemeinek relevancia szerinti pontozása, kivonat szempontjából legfontosabb szövegelemek kiválasztása [2]. Az első lépésben a dokumentum megfelelő ábrázolásáról gondoskodnak, amely során a szöveget kisebb feldolgozandó egységekre bontják (például mondatokra és szavakra). Ezt a kivonathoz kiválasztott szövegelemek léptéke határozza meg: kulcsszavakból vagy kulcsmondatokból összeállított kivonat. Ezt követően a szövegelemeket egy választott pontozási módszer mentén értékelik, amelyek halmazát a kivonathoz felhasznált egységek szűkítik le. A szövegelemek pontozására használt módszereket négy kategóriába sorolhatjuk: statisztikai, nyelvészeti, gráf és gépi tanulás alapú megközelítések. A statisztikai csoportba az olyan módszerek tartoznak, amelyek a

[1] Towards Data Science: A Quick Introduction to Text Summarization in Machine Learning, https://towardsdatascience.com/a-quick-introduction-to-text-summarization-in-machine-learning-3d27ccf18a9f (2019. okt.15.)

szövegelemek szövegbeli előfordulása alapján végzik a pontozást: például szó gyakoriság [4], szókapcsolatok és szavak együttes előfordulása [4], súlyozott gyakoriság [5], TF-IDF [6], RAKE [7]. A szó gyakoriság alapú módszer azokat a szavakat jelöli meg kulcsszavakként, amelyek a szövegben legtöbbször szerepelnek. Ezt elsősorban olyan feladatoknál használják, ahol a leggyakoribb szavak kinyerése a cél: leggyakoribb hibák azonosítása ügyfélszolgálati rendszereknél. A szókapcsolatok és szavak együttes előfordulásának vizsgálata a szó N-gramok statisztikáját jelenti, ami a leggyakrabban együtt járó szavakból álló kifejezések azonosítására alkalmas: customer service, social media. A súlyozott gyakoriság módszere az előbb említett két módszerrel ellentétben a mondatok pontozására használható. A mondatok pontszáma az őket alkotó szavak gyakoriság alapú pontszámának összegéből jön létre. A statisztikai módszerek közül a két legelterjedtebb a TF-IDF és RAKE algoritmusok. A TF-IDF lényege, hogy a szöveg tartalmát a benne ritkán előforduló szavak ragadják meg a legjobban, ezért a szavakat szövegbeli gyakoriságuk alapján priorizálja. Ez abban az esetben hatékonyan használható technika, ha a szöveg főbb szavainak kinyerése a cél. A RAKE szintén gyakoriság alapján választja ki a kulcsszavakat a szövegből, amelyek egy vagy több szóból álló kifejezések is lehetnek. Szemben a szókapcsolatok és szavak együttes előfordulásának vizsgálatával, nem csak a bevett kifejezéseket képes megtalálni a szövegben, hanem a gyakran ismétlődő szavak és kifejezés határoló karakterek listája alapján detektálja az összetartozó szavakat. A második, nyelvészeti csoportba azok a megközelítések tartoznak, amelyek a szöveget és az őt alkotó szavakat nyelvészeti szempontokból vizsgálják: például szavak mondaton belüli szerepe (POS), a szavak egymáshoz való viszonya nyelvtani függőségek ábrázolása alapján. Ezek lényege, hogy bizonyos nyelvtani tulajdonsággal rendelkező szavak fontosabbak a tartalomleírás szempontjából: főnevek, igék és melléknevek hordozzák a lényegi információt a szövegről. A gráf alapú csoportot az olyan módszerek alkotják, ahol a gráf csomópontjai a kivonathoz felhasznált szövegelemek (szavak / mondatok), a köztük menő élek pedig az egymáshoz való viszonyukat fejezik ki. A gráf alapú csoport legelterjedtebb módszere a TextRank [8], ami a gráf alapú algoritmusok természete miatt egyaránt használható kulcsszavak és kulcsmondatok kinyerésére is. Elsősorban olyankor használják, amikor a szöveg lokális és globális információira is egyaránt szükség van. A gépi tanulási módszereket pedig olyan algoritmusok alkotják, amelyek példa adatokon történő tanítást követően képesek predikálni a szöveg kulcsszavait / kulcsmondatait: például SVM, neurális hálók, CRF [9]. A felsorolt pontozási módszerek közül az általam használtakat a 4.1-es fejezetben részletesen

ismertetem majd. A szövegelemek pontozását követően végül kiválasztják közülük azokat, amelyek a legnagyobb pontszámokat érték el, s így a leginkább alkalmasak a tartalom leírására. Ezeket a kinyert elemeket ezután egymás után fűzve létrejön a kivonat. A módszer előnye, hogy segítségével könnyen előállítható a rövidített változat, azonban számtalan problémával is járhat. Az egyik leggyakoribb hiba a szövegkohézió hiánya, amely akár a szavak vagy a mondatok szintjén is megmutatkozhat. A fenti táblázatban megfigyelhető, hogy a rövidítést alkotó két mondat tartalmilag egymástól teljesen független, valamint az őket alkotó szavak között sincsen semmilyen összetartó erő: „attend event Jerusalem". Ennek oka, hogy a szöveg főbb tartalmi elemeinek összefűzésekor nem elemezzük a köztük lévő kapcsolatot, nyelvtani szabályokat. Az elemzés hiánya miatt így többek között értelmetlen szöveg: „Mary birth Jesus", illetve szó- vagy mondat ismétlések alakulhatnak ki. Ennek a generálási módszernek tehát nincsen az eredeti dokumentumhoz hozzáadott értéke, mivel annak elemeiből építkezik.

Ezzel szemben az absztrakt összefoglalók elkészítése egy szöveggenerálási folyamat, amely a szöveg elemei közötti kapcsolatok azonosítását igényli. Az összefoglaló létrehozásakor az új mondatok előállításához a dokumentum szó- és mondatkészletétől eltérő szövegelemeket is felhasználhat. Az absztraktot alkotó „came, where, was, born" szavak (ld. 2.1. táblázat) már nem a forrás szövegből származnak. A módszer alapja az, hogy feltérképezi a szöveg elemeinek kapcsolatát és ezen relációk figyelembevételével legenerálja az absztrakt mondatait. Az ilyen típusú összefoglaló generáló rendszerek alapját leggyakrabban az úgynevezett sequence-to-sequence [10] (seq2seq) architektúra képezi, amely működése a 2.2-es fejezetben ismertetett rekurrens neurális hálókon nyugszik. Az architektúra szekvencia alapú feldolgozó mechanizmusa, illetve rekurrens jellege miatt az összefoglaló generáláson kívül számtalan feladatnál kimagaslóan jó eredményt ért el: például morfológiai reinflexió, beszédfelismerés, képaláírás generálása, időjárás előrejelzése idősor alapon. Ezeknél a feladatoknál a bemenetek és kimenetek szekvenciális formában állnak rendelkezésre és a kimeneti szekvencia adott elemének előállításához elengedhetetlen, hogy a sorozat korábban feldolgozott elemeiből gyűjtött információt is felhasználja.

Az absztrakt megközelítés legnagyobb hátrányát a bonyolult generálási folyamat jelenti, amely során előfordulhatnak szóismétlések, valamint a szókészlet bővítése is extra komplexitást vihet a rendszerbe. Viszont szemben a kinyerés alapú módszerrel, az ily módon előállított absztraktok kohezívak lesznek, mert a szövegalkotás a kontextus

elemzése mellett történik [2]. Ennek köszönhetően a generálás során a modell egyezteti a nyelvtani szabályokat (például cselekvés ideje, helye, személye): „came to Jerusalem”, így leginkább ezek hasonlítanak majd az ember által készített összefoglalókra. A következő fejezetben a seq2seq alapját képező rekurrens neurális hálókat ismertetem részletesen.

2.2 Rekurrens neurális háló (RNN)

A rekurrens neurális hálók (RNN) a neurális hálók egy fajtája, amelyet elsősorban szekvencia alapú információelőállítás esetén használnak. Maga a hálózat RNN cellákból épül fel, amelyek képesek eltárolni a forrásból eddig gyűjtött információ alakulását. Ehhez a hálózat rekurrens jellegét használja fel, amely a hálózati réteg kimenetét minden feldolgozási lépésben visszatáplálja annak bemenetére [11]. Ez egyben azt is jelenti, hogy az RNN esetében megjelenik egy idő (t) dimenzió, ami a szekvencia feldolgozásának egy elemi lépése.

$$x = \left(x_1, \dots, x_{T_x}\right)$$

$$h_t = \phi(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y = \left(y_1, \dots, y_{T_y}\right)$$

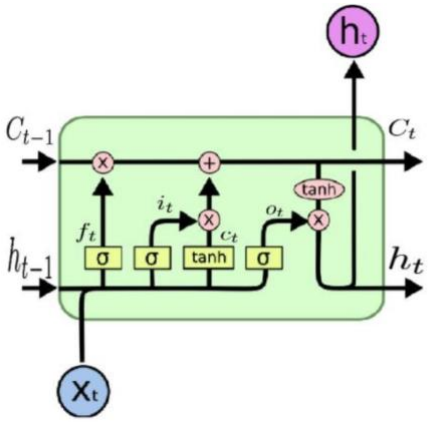
$$y_t = W_{hy}h_t$$

2.1. képlet: RNN működése [14]

A szekvencia feldolgozása során ugyanaz a folyamat hajtódik végre a sorozat minden elemére (x_t), de már az előző számítás eredményének figyelembevételével, azaz a már eddig előállított szekvenciaelemeket is beleszámítja a következő elem (y_t) generálásakor. Ehhez a hálózat adott időpillanatban vett rejtett állapotát (h_t), valamint súlyait (W) használja fel. A hálózat a rejtett állapotok számításához leggyakrabban sigmoid vagy tanh aktivációs függvényt (ϕ) alkalmaz [12].

2.2.1 Long Short Term Memory Networks (LSTM)

A Long Short Term Memory Networks [13] (LSTM) az RNN cellák egyik típusát jelenti, amelyet elsősorban hosszú szekvenciák feldolgozása esetén használnak. A hagyományos RNN-ek ugyanis nem képesek az egymástól távol elhelyezkedő szekvenciaelemek kapcsolatát feltérképezni [14].

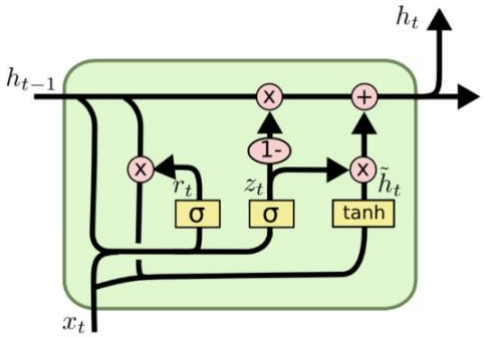


2.1. ábra: LSTM cella felépítése [14]

Az LSTM (ld. 2.1. ábra) ezt egy memória jellegű működéssel valósítja meg, amely az általa tárolt információt három kapu segítségével vezérli: bemeneti kapu (i), törlő kapu (f), kimeneti kapu (o). A bemeneti kapu feladata, hogy szabályozza a cellába történő információírást. Ez azt jelenti, hogy meghatározza az aktuális bemenetből (x_t) származó tárolandó információ mennyiségét és relevanciáját. A törlő kapu feladata, hogy a rendszer számára az eddig gyűjtött információk közül (h_{t-1}) kizárólag a releváns részleteket tartsa meg. A kimeneti kapu végül azt szabályozza, hogy a belső állapotból az LSTM mit jelenítsen meg a kimeneten (h_t). Az LSTM cella a számítása során sigmoid (σ) és tanh aktivációs függvényeket használ [14].

2.2.2 Gated Recurrent Units (GRU)

Az RNN cellák másik típusát a Gated Recurrent Units [15] (GRU) alkotja, amelyek szintén elsősorban hosszú szekvenciák feldolgozásánál népszerűek. Ezek a cellák egyfelől már csak a belső rejtett állapotokat használják az információelőállításhoz, szemben az LSTM cellákkal, amelyek a cella állapotot (c_t) is vizsgálják.



2.1. ábra: GRU cella felépítése[2]

Ezenkívül a GRU cellák mindössze kétféle kapuból épülnek fel: visszaállító (r) és frissítő kapuból (z). A frissítő kapu feladata, hogy meghatározza mely információkra emlékezzen a rendszer. A visszaállító kapu ehhez hasonlóan a korábban gyűjtött információk megőrzéséről dönt [16]. A GRU az LSTM-hez hasonlóan a sigmoid és tanh aktivációs függvényekkel dolgozik. Az egyszerűbb felépítésének köszönhetően gyorsabb számítási sebességet biztosít a neurális hálók számára, míg az LSTM összetettsége miatt esetenként pontosabb eredményhez vezet. A következő fejezetben a rekurrens neurális hálókkal generált összefoglalók kiértékelésénél használt módszereket mutatom be.

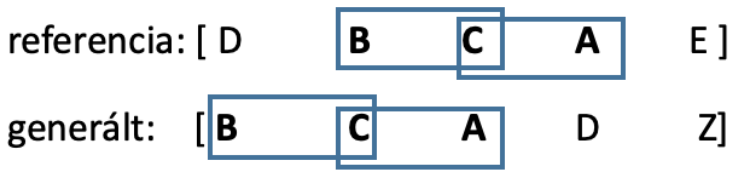
2.3 Összefoglalók automatikus kiértékelése (ROUGE)

A számítógép által generált összefoglalók automatikus kiértékelése legtöbbször az úgynevezett ROUGE [17] (Recall-Oriented Understudy for Gisting Evaluation) csomag segítségével történik. A ROUGE olyan metrikák gyűjteménye, amelyek a generált és az elvárt referencia összegzés közti egyezéseket vizsgálják, s a közös részek mértéke alapján értékelik őket. Az egyes metrikák között a különbséget mindössze az egyezés alapegysége jelenti. Eszerint öt fő mérőszámot különböztetünk meg: ROUGE-N, ROUGE-L, ROUGE-W, ROUGE-S és ROUGE-SU. A továbbiakban ezek közül a ROUGE-N és ROUGE-L metrikákat fejtem ki.

[2] Isaac Changhau: LSTM and GRU – Formula Summary, https://isaacchanghau.github.io/post/lstm-gru-formula/ (2019. máj.12.)

2.3.1 ROUGE-N

A ROUGE-N metrika a generált és referencia összefoglalók közötti n-gram egyezéseket keresi meg. Az n-gramok jelen esetben szó n-gramokat jelentenek, azaz a szövegen n méretű ablakot végigcsúsztatva próbálja a mindkét összefoglalóban előforduló szó n-eseket megtalálni. Ehhez nyilvántartja az egyes összefoglalókból létrejött n-gramokat és ennek a két n-gram listának keresi meg a metszetét.



2.2. ábra: ROUGE-2 példa

Ennek működésére a 2.2. ábra mutat példát, ahol minden karakter egy különálló szót reprezentál. A ROUGE-1 metrika alkalmazásánál ekkor minden szó önálló egységként szerepel, azaz a referencia összefoglalóból D, B, C, A, E, míg a generált összefoglalóból B, C, A, D, Z 1-gram lista jön létre. A ROUGE-2 esetén ezzel szemben két szó alkot egy vizsgálandó alapegységet, ezért a 2.2. ábra alapján a referencia összefoglaló n-gram listája (D, B), (B, C), (C, A), (A, E), a generált összefoglaló listája pedig a (B, C), (C, A), (A, D), (D, Z) párokat tartalmazza. A ROUGE-N érték számítása a 2.2. képlet szerint történik, ahol az n_gram_{Gen} és n_gram_{Ref} a generált és referencia összefoglaló n-gramjait jelöli. Ezalapján a ROUGE-2 érték 2 / 4 = 0,5, mert a két lista metszetét a (B, C) és (C, A) 2-gramok jelentik és a referencia összefoglalóból négy darab 2-gram képezhető.

$$\frac{\left|(n\_gram_{Gen}) \cap (n\_gram_{Ref})\right|}{\left|(n\_gram_{Ref})\right|}$$

2.2. képlet: ROUGE-N metrika számítása

A ROUGE-N metrika tehát egyfajta fedési értékként értelmezhető, ami megmutatja, hogy milyen arányban fedhetők le a referencia összefoglaló n-gramjai a generált összefoglaló n-gramjaival. A ROUGE-2-nél kapott 0,5-es érték azt jelenti, hogy a generált összefoglaló a referencia összefoglaló elemeinek felét tartalmazza.

2.3.2 ROUGE-L

A ROUGE-L a leghosszabb, nem feltétlenül összefüggő olyan szekvenciát keresi meg, amely mindegyik szövegben (a referencia és generált összefoglalóban) egyidejűleg előfordul. A közös rész meghatározásánál nem szükséges, hogy a közös szavak közvetlenül egymást kövessék a szövegben, de a közös rész szavainak sorrendje meg kell hogy egyezzen.

referencia:	[man	buys	two	tickets]
generált1:	[**man**	buy	**two**	**tickets**]
generált2:	[**two**	**tickets**	man	buy]

2.3. ábra: Mondat szintű ROUGE-L példa

A 2.3. ábra a mondat szintű ROUGE-L számítását mutatja be egy példa mentén, ahol a kiemelt szavak a generált és referencia összefoglalók leghosszabb közös szekvenciáját jelölik. Ezen az ábrán az látható, hogy a referencia és az első generált összefoglaló közti leghosszabb közös szekvencia a man, two, tickets szavakból áll. A másik generált összefoglaló esetében azonban már csak a two, tickets szavak alkotják a leghosszabb közös részt. Ennek oka, hogy a második generált összefoglaló a man, two, tickets elemeket a referencia összefoglalóétól eltérő sorrendben tartalmazza: two, tickets, man, ezért a man elemet nem tekinti egyezésnek.

A mondat szintű ROUGE-L érték számítása a 2.3. képlet szerint történik, ahol az R_{LCS} a fedést, a P_{LCS} a pontosságot, az F_{LCS} az F mértéket jelöli, amelyek közül utóbbi a fedés és a pontosság súlyozott harmonikus közepét jelenti. A képletben az LCS függvény a referencia és generált összefoglaló egy mondata közti leghosszabb közös szekvenciát, tehát a sorrendjében megegyező közös szavak számát, az m a referencia összefoglaló, az n pedig a generált összefoglaló adott mondatának szószámát adja meg.

$$R_{LCS} = \frac{LCS(X,\ Y)}{m}, P_{LCS} = \frac{LCS(X,\ Y)}{n}$$

$$\beta = \frac{P_{LCS}}{R_{LCS}}, F_{LCS} = \frac{(1 + \beta^2)\, R_{LCS}\, P_{LCS}}{R_{LCS} + \beta^2\, P_{LCS}}$$

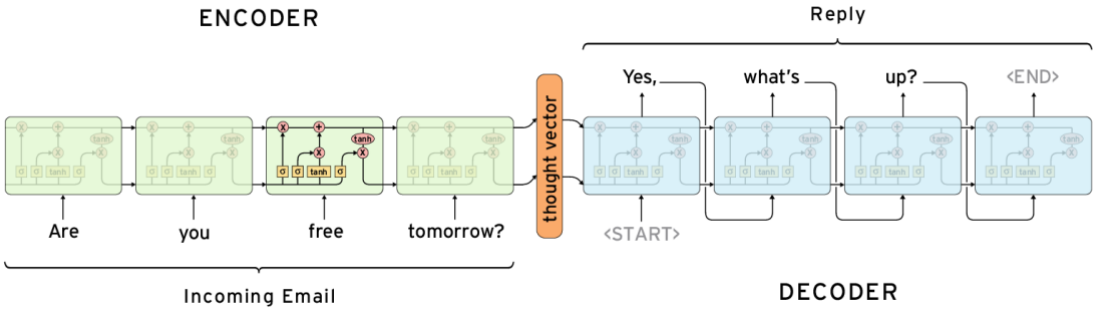
2.3. képlet: Mondat szintű ROUGE-L metrika számítása [17]

3 Absztrakt alapú összefoglaló generálás tervezése és megvalósítása

3.1 Modell tervezése

3.1.1 Sequence-to-sequence architektúra

Az absztrakt alapú összefoglaló generáló rendszerem alapjául a sequence-to-sequence architektúrát választottam, amely szekvencia alapú feldolgozó mechanizmusa miatt jól illeszkedik az absztrakt generáláshoz hasonló szöveggenerálási feladatokhoz. Ez a szekvenciális feldolgozás ugyanis igazodik a szöveg felépítéséhez, amely a mondatok és a szavak sorozataként, azaz szekvenciájaként értelmezhető. Az architektúra alkalmazásának másik előnye, hogy a hagyományos neurális hálókkal ellentétben képes több komponens egyidejű egymásra hatását is figyelembe venni. Ez az absztrakt generálás esetén a kontextusvizsgálat előnyével jár. A felépítésének köszönhetően továbbá kezelni tudja az eltérő hosszúságú bemeneti és kimeneti szekvenciákat, így az absztrakt generáló rendszer hosszú forrás szövegéhez képes rövid összefoglalót előállítani.



3.1. ábra: A sequence-to-sequence architektúra felépítése[3]

A seq2seq architektúra két fő egységre osztható fel, az úgynevezett enkóderre és dekóderre, amelyek működése a 2.2-es fejezetben bemutatott rekurrens neurális hálókon alapul. Az enkóder feladata a bemeneti szekvencia kódolása, amelyet a dekóder kimeneti szekvenciává alakít vissza. Az absztrakt generálásnál a bemenet a hosszú szöveg, aminek

[3] Towards Data Science: Sequence to sequence tutorial, https://towardsdatascience.com/sequence-to-sequence-tutorial-4fde3ee798d8 (2019. okt.15.)

kódolt állapota alapján (thought vector/kontext vector) a dekóder előállítja a generált kimenetet, az absztraktot. A két egység munkájának összehangolásához biztosítani kell közöttük a kommunikációt. Ehhez két speciális vezérlő karakterre van szükség: a start-of-sequence (START) karakter a szekvencia elejét határozza meg, míg az end-of-sequence (END) a sorozat végét jelöli. Mindkét egység a START hatására kezdi meg a feldolgozást és az END elérésekor áll le. Ez a két jelző karakter teszi lehetővé, hogy az architektúra eltérő hosszúságú bemeneteket, illetve kimeneteket tudjon kezelni. A START és END észlelése között eltelt időben iteratív módon dolgozzák fel a szekvenciákat, ahol minden lépésben a sorozat egy elemi egységét vizsgálják. Az általam készített rendszer alapfelépítése megegyezik a hagyományos seq2seq (ld. 3.1. ábra) felépítésével, ahol a szekvencia alapegysége egy szó.

$$x = (x_1, \dots, x_{T_x})$$

$$h_t = f(x_t, h_{t-1})$$

$$c = q(\{h_1, \dots, h_{T_x}\})$$

3.1. képlet: Enkóder működése [18]

A szekvenciák feldolgozása során az enkóder minden iterációban (t) a bemenet (x_t) egy rejtett állapotát (h_t) hozza létre az eddig feldolgozott elemek (x_1, ..., x_{t-1}) figyelembevételével. A szekvencia feldolgozásának végső rejtett állapota az úgynevezett thought vektor (c) lesz, ami az END érzékelésekor jön létre. Ekkor a START karakterrel jelez a dekódernek, hogy a kódolás végetért. A thought vektor előállításához az enkóder nem lineáris függvényeket (f, q) használ. Az f például lehet a 2.2.1-es fejezetben bemutatott LSTM, míg a q függvény jellemzően a végső rejtett állapottal, a h_{Tx} értékkel tér vissza [18].

$$y = (y_1, \dots, y_{Ty})$$

$$s_t = f(s_{t-1}, y_{t-1}, c)$$

$$p(y_t \mid \{y_1, \dots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$$

$$p(y) = \prod_{t=1}^{T} p(y_t \mid \{y_1, \dots, y_{t-1}\}, c)$$

3.2. képlet: Dekóder működése [18]

A kimenet generálásakor a dekóder a thought vektor (c) alapján a korábbi részegységek vizsgálata mellett (y_1, ..., $y_{t-1.}$) generálja a következő kimeneti szekvenciaelemet (y_t). Ehhez kiszámolja minden generált szekvenciaelem feltételes valószínűségét, amelyhez a g nem lineáris függvény a dekóder aktuális állapotát (s_t) is figyelembe veszi. A hagyományos seq2seq esetében az adott lépésben az elemek közül mindig a legnagyobb valószínűségűt választja ki, amit hozzáfűz az eddigi sorozathoz. Ezeket a feltételes valószínűségeket minden kimeneti szekvenciára végül összeszorozza [18]. A rendszer végső kimenete az END karakter hatására jön létre.

A következő fejezetekben olyan technológiákat mutatok be, amelyek hozzájárulnak a seq2seq-kel generált absztraktok jósági értékeinek javulásához. Ezek ismertetését az embedding bemutatásával kezdem.

3.1.2 Embedding réteg

A mély tanulás hatékonyságának növelése érdekében az adatok megfelelő előkészítése elengedhetetlen, mert az esetek többségében valamilyen extra információt szolgáltatnak a rendszer számára vagy érthetőbb formába transzformálják azokat. Az egyik ilyen adatelőkészítési metódus, amely az adatok közötti kapcsolatok feltárására szolgál az úgynevezett embedding réteg.

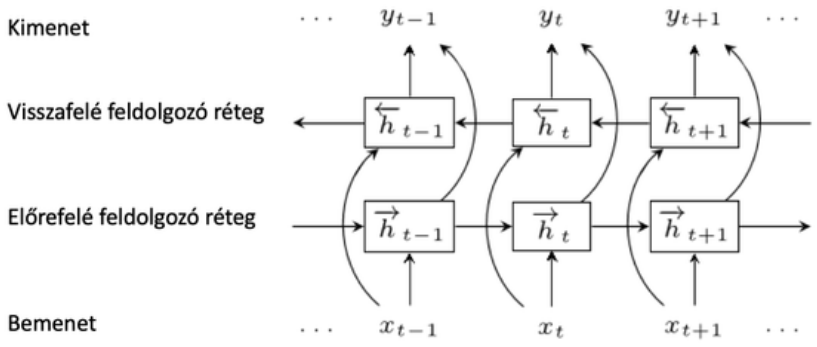
Az embedding általában a neurális hálók első rétegét képezi, amely az összetartozó bemeneti adatok előfeldolgozását végzi. Az egyik típusát az ú.n. „szó embedding” jelenti, amely olyan feladatok esetén használható, ahol az adatok ábrázolása az absztrakt generáláshoz hasonlóan a szavak szintjén történik. A „szó embedding” lényege, hogy a szavak megfelelő számvektor reprezentációját állítja elő. Erre azért van szükség, mert a neurális háló számára a szöveges adatokat számokká kell transzformálni. A szavak számokká történő leképezése azonban önmagában még nem fejezi ki azok egymáshoz való viszonyát: girl: 1, boy: 2, woman: 3 esetén nem derül ki, hogy a girl és woman szavak hasonló jelentést hordoznak. A „szó embedding” feladata, hogy a megfelelő reprezentációk segítségével az össszetartozó szavakat egymáshoz hasonlónak ábrázolja, hogy minél hatékonyabb és gyorsabb legyen a tanulási fázis. Az electrical, electricity, battery, voltage szavak hasonló jelentést hordoznak, így a szó embbeding őket egymáshoz közelinek ábrázolja. Mivel az embedding az egyes szekvenciaelemekhez jelentésük alapján megfelelő vektorreprezentációt rendel, így az előbb említett electrical,

electricity, voltage szavak egymáshoz hasonló vektorokkal lesznek ábrázolva a modell számára.

A következő fejezetben az enkóder / dekóder kétirányúsításával egy másik olyan módszert ismertetek, amely beépítésével még több információ szolgáltatható a szekvenciáról az architektúra számára.

3.1.3 Egy- és kétirányú enkóder és dekóder

A seq2seq alapját képező enkóder és dekóder feldolgozási mechanizmusa egyaránt kétféle lehet. Ezalapján megkülönböztetünk egy- és kétirányú kódolót és dekódolót. Az egyirányú eset a rekurrens háló alap állapotát jelenti, amikor a szekvencia feldolgozása kizárólag elölről történik, azaz a soron következő elem kódolásakor/dekódolásakor csak a korábbi szekvenciaelemeket veszi figyelembe.



3.2. ábra: Kétirányú LSTM működése [4]

Ezzel szemben a kétirányú típus két rekurrens neurális hálóból épül fel, ahol az egyik RNN a szekvencia elejéről, míg a másik a sorozat végéről indulva végzi a feldolgozást. A végső állapot ennek a két RNN-nek a belső állapotainak összefűzéséből $(\overrightarrow{h_t}, \overleftarrow{h_t})$ jön létre. Az egyirányú mechanizmus előnye, hogy a kétirányúhoz képest gyorsabb tanítást tesz lehetővé. Ez ugyanis kevesebb paraméterrel rendelkezik, mint a kétirányú változata, így kevésbé számításigényes. Ez egyben a hátrányát is jelenti, mivel ezáltal kevesebb információ áll rendelkezésére a helyes eredmény előállításához. Az absztrakt generálásnál a kétirányúság bevezetésével a seq2seq a szekvencia aktuális

[4] Towards Data Science: Elmo Embeddings in Keras with TensorFlow hub, https://towardsdatascience.com/elmo-embeddings-in-keras-with-tensorflow-hub-7eb6f0145440 (2019. okt.15.)

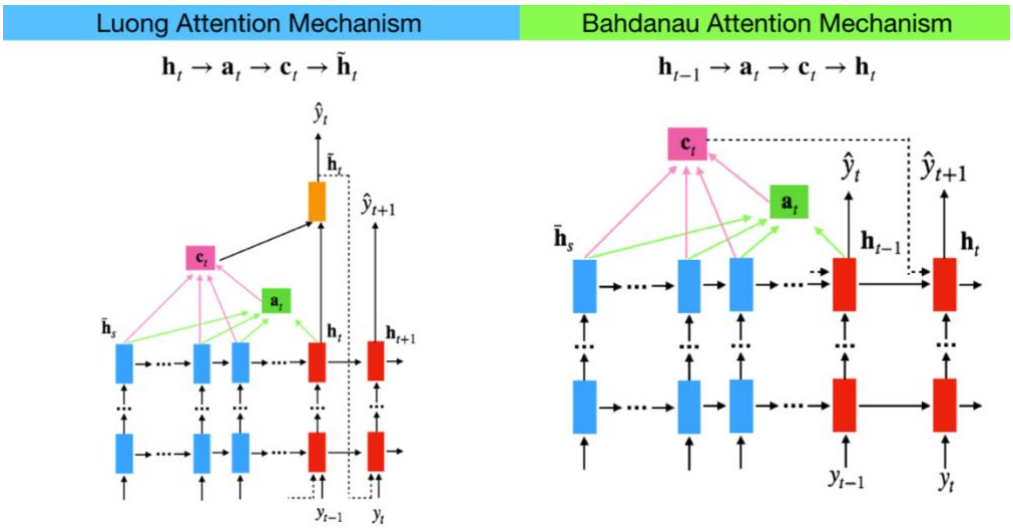
elemének predikálásakor az őt megelőző és követő elemeket, tehát a kontextust is megvizsgálja.

A következő fejezetben az attention mechanizmust ismertetem részletesen, amely a kétirányúsításhoz hasonlóan arra szolgál, hogy a szekvenciaelemek generálásakor a rendszer a szekvencia bizonyos részeinek fokozott figyelmet szenteljen.

3.1.4 Attention mechanizmus

A hálózat hatékonyabb működése érdekében a dekódolás során attention [18] mechanizmus használható. A dekóder alapesetben, azaz attention nélkül csak a dekóder rejtett rétegeiből állítja elő a kimenetet. Ebben az esetben a dekódernek nincs információja arról, hogy az egyes szekvenciaelemek külön-külön milyen információt hordoznak. Az attention segítségével azonban az enkóder rejtett rétegek belső állapotaiból képzett, az úgynevezett kontext vektort is figyelembe vesszük a kimenet előállításánál. A mechanizmus ezzel extra információt szolgáltat a sorozatról, mert az elemek reprezentációjához enged a dekóder számára hozzáférést, valamint segít feltérképezni a szekvenciaelemek közötti korrelációt.

Az attention mechanizmusnak két fő csoportját különböztetjük meg: globális és lokális attention. A globális az enkóder összes rejtett állapotát (szekvenciaelemét) figyelembeveszi a következő elem generálásához, míg a lokális csak ezek egy részére (szekvenciaelemek részhalmazára) fókuszál. Az attention két legelterjedtebb típusa a globális Bahdanau és a globális, illetve lokális formában is előforduló Luong attention.



3.3. ábra: Luong és Bahdanau attention összehasonlítása [19]

A Luong attention (ld. 3.3. ábra a t. időpillanatban előállított szekvenciaelemet (^y_t) a dekóder t. időpillanatbeli rejtett állapotából (h_t) és az enkóder összes rejtett állapotából (h_s) generálja. A Bahdanau szintén a dekóder és enkóder rejtett állapotait használja, de csak közvetett módon, ugyanis a t. időpillanatban generált (kimeneti) szekvenciaelemhez közvetlenül csak a dekóder előző (t-1.) időpillanatának rejtett állapotát használja fel. Egy másik különbséget a két attention mechanizmus kontext vektor (c_t) integrálási módja jelenti: a Luong attention a kontext vektorból és a dekóder t. időpillanatbeli rejtett állapotából egy RNN jellegű működésen keresztül generálja a t. időpillanathoz tartozó (kimeneti) szekvenciaelemet. A Bahdanau attentionnél pedig a kontext vektor az aktuális időpillanatbeli rejtett dekóder állapottal együtt használható a dekóder következő rejtett állapotának meghatározására. Az utolsó különbséget a score kiszámítási mód jelenti, ami a dekóder rejtett állapotának és az enkóder rejtett állapotainak összehasonlítására szolgál [19]. A továbbiakban a Bahdanau attention működését fejtem ki részletekbe menően, mert a Luong attentionnél kevesebb komplexitást visz az amúgyis számításigényes absztrakt generáló rendszerembe.

$$score(h_t, h_s) = a(h_{t-1}, h_s)$$

$$\alpha_{ts} = \frac{\exp\left(score(h_t, h_s)\right)}{\sum_{s'=1}^{S} \exp\left(score(h_t, h_{s'})\right)}$$

$$c_t = \sum_{s=1}^{S} \alpha_{ts} h_s$$

$$h_t = RNN(h_{t-1}, [\, c_t, h_{t-1}])$$

3.3. képlet: Dekóder működése attentionnel [19]

A 3.3. képlet a Bahdanau attention működését mutatja be. Eszerint a kontext vektor t. időpillanatbeli állapotának (c_t) kiszámításakor súlyozottan (α_{tj}) figyelembe veszi az adott időpillanatig létrejött rejtett enkóder állapotokat (h_s). Az α_{tj} súlyok egy j hosszúságú vektort (súlyrendszert) alkotnak (ld. a 3.3. ábra, ami a_t-vel van jelölve. A súlyozás során azt vizsgálja, hogy a bemenet s. pozíciója mennyire egyezik a kimenet t. pozíciójával (score(h_t,h_s)), ahol az „a" egy előrecsatolt neurális hálót jelöl [18]. A dekóder t. időpillanatbeli rejtett állapota (h_t) a t-1. rejtett dekóder állapotból, valamint a kontext vektor összefűzéséből számítódik. A t. rejtett dekóder állapotból pedig a következő (azaz a t+1.) időpillanatban lesz előállítva a generált kimenet.

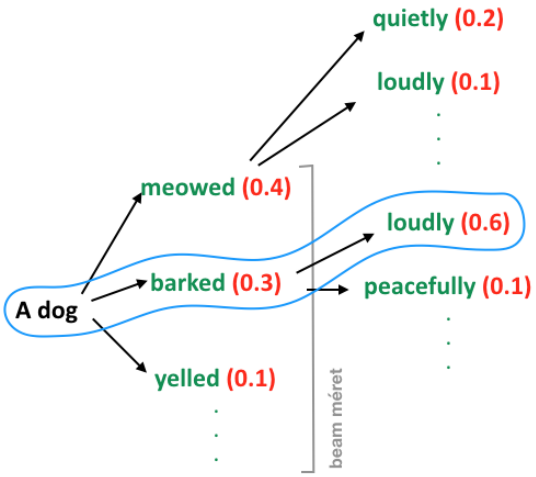
Az attentionnel kiegészített dekóder számára tehát több információ áll rendelkezésre a helyes kimenet generálásához. A kimeneti szekvencia előállításakor azonban a rendszert félrevezetheti, ha a sorozat egy elemét helytelenül generálja. A szekvenciaelemek predikálásához ugyanis a korábban generált elemeket is figyelembe veszi. A következő fejezetben egy olyan dekódert mutatok be, amely csökkenti az ilyen helytelen sorozatok előállításának valószínűségét.

3.1.5 Beam Search dekóder

A Beam Search egy olyan heurisztikus kereső algoritmus, amely egy keresési faként fogható fel. Az algoritmus a fa minden szintjén értékeli az egyes elemek valószínűségét, majd csökkenő sorrendbe rendezi őket ezen érték alapján. Ezután kiválasztja az n darab legnagyobb valószínűségű elemet az adott szinten [20]. Ez az n szám az úgynevezett beam méret, ami a keresési feltételnek leginkább megfelelő jelöltek számát határozza meg. Ezt követően a keresést kiterjeszti az n csúcs mindegyikének irányába. Az előbb említett lépéssorozatot ismétli, amíg a fa leveleihez nem ér. Közülük végül azt az elemet választja, amelyik a legnagyobb valószínűséggel rendelkezik. Az

algoritmus előnye, hogy mivel minden lépésben több jelöltet is megtart, ezért kisebb eséllyel csúszik félre a keresés eredménye az egyes elágazási pontoknál hozott döntések miatt.

A Beam Search egy módosított változata a sequence-to-sequence architektúra által generált absztraktok jóságát is lényegesen javíthatja. Elsősorban a dekódolási szakaszban nyújthat segítséget, hogy a rendszer a kontextusba leginkább illeszkedő szekvenciaelemet állítsa elő. A dekóder minden időpillanatban a generált elemek közül a beam méretnek megfelelő legvalószínűbb szekvenciaelemeket tartja meg, amelyeket jelölteknek nevezünk [21]. Minden jelöltet külön-külön hozzáfűz az eddig előállított szekvenciához, így egy szekvenciahalmazt hoz létre. Innentől minden iterációs lépésben a halmaz összes szekvenciáját egymás után a dekóder bemenetére juttatja és legenerálja a sorozatok következő elemét, amelynek valószínűségét hozzáadja az eddig előállított szekvencia valószínűségéhez. A dekóder végső predikciója az a sorozat lesz, amelyik összességében a legnagyobb valószínűséggel rendelkezik. A 3.4. ábra ezt a folyamatot szemlélteti, ahol az „A dog” sorozat következő elemének generálásakor a hagyományos módszer használata hibás döntéshez vezetne. Bár az aktuális lépésnél a „meowed” szó rendelkezik a legnagyobb 0,4-es valószínűséggel, mégis a „barked” szó ágán lévő szekvencia éri el végeredményben a legjobb 0,9-es valószínűségértéket.

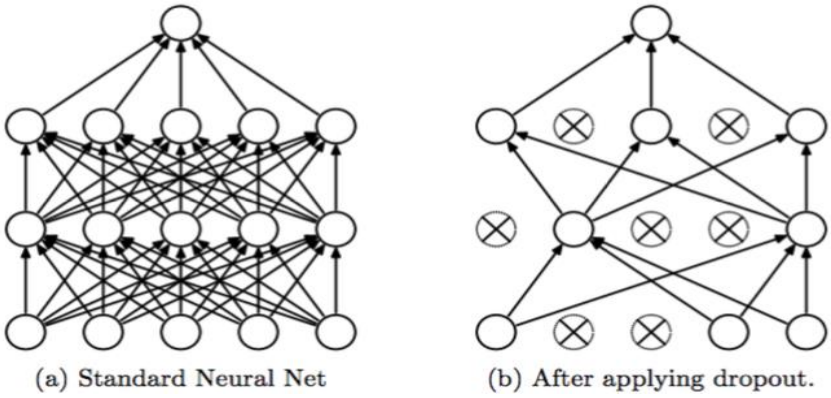


3.4. ábra: Beam Search dekóder működése példa

A továbbiakban egy olyan módszert ismertetek, ami hozzájárul ahhoz, hogy az absztrakt generálásakor a dekóder ne adathalmaz specifikus jellemzőket sajátítson el. Ennek köszönhetően a modell más adathalmazok dokumentumaihoz is képes összefoglalót generálni.

3.1.6 Dropout

A neurális hálók tanítása során gyakran előforduló jelenség a túltanulás, amely a tanító adathalmaz legapróbb jellemzőinek (zaj) túlzott elsajátítását jelenti. Ez a modell általánosító képességének torzulásával jár, ezért az nehezen vagy egyáltalán nem használható más adathalmazok predikálására.



3.5. ábra: Dropout működése [22]

A dropout a túltanulás megelőzésére szolgáló regularizációs technika, amely a kimeneti réteg kivételével iterációnként véletlenszerűen kihagy neuronokat a számításból. Ez a módszer a neuronokat arra kényszeríti, hogy sokkal robosztusabb, sok különböző adathalmaz esetén használható jellemzőket sajátítsanak el. Ezt a dropout következtében kialakuló kiegyensúlyozottabb súlymátrix teszi lehetővé. Ez a regularizációs technika (ld. 3.5. ábra a neurális hálók méretcsökkenését eredményezi, ami gyorsabb tanítási iterációkhoz vezet, viszont a jobb általánosítóképesség miatt a teljes tanítási fázis idejének megnövekedésével jár [22].

3.2 Modell megvalósítása

3.2.1 Referencia modellek

Az absztrakt generáló rendszer tervezésekor Peter Liu és Xin Pan megoldásából származó ötleteket vettem alapul[5]. Az általuk készített rendszer a sequence-to-sequence architektúrára épül, amelyet kétirányú enkóderrel, valamint egyirányú dekóderrel valósítottak meg. Az enkódert és dekódert alkotó rekurrens neurális hálókban az LSTM

5 Github: Textsum, https://github.com/tensorflow/models/tree/master/research/textsum (2019. okt.15.)

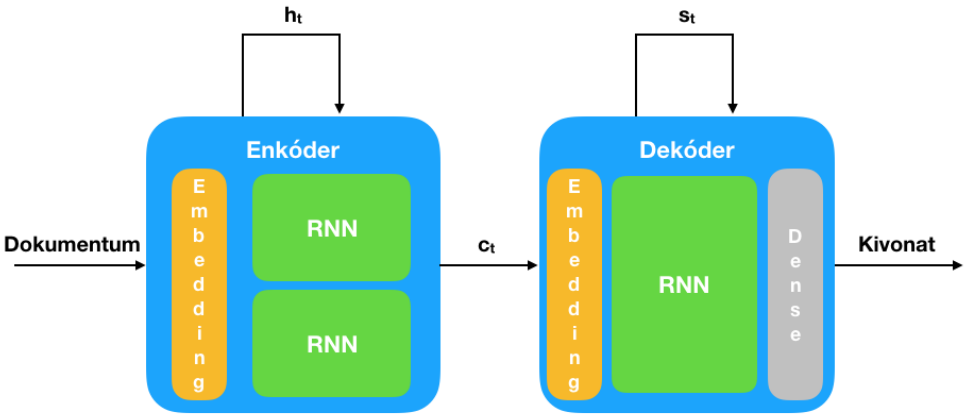
cellát használták, valamint mindkét egység első rétegeként embeddinget alkalmaztak a bemeneti szekvenciák előzetes feldolgozására. A modelljükben a dekóder működése a beam search algoritmus alapján történik, amelyet attention mechanizmussal egészítettek ki. A rendszer adatábrázolási stratégiája a szövegeket a szavak szintjén ábrázolja, mely során minden szóhoz egy természetes számot rendel. A szöveg ábrázolásakor hat speciális karaktert használtak, amelyek a szöveg struktúrálására szolgálnak: paragrafus kezdete, paragrafus vége, mondat kezdete, mondat vége, dokumentum kezdete, dokumentum vége. Ezekkel a modell számára biztosítanak extra információt, hogy a szöveg felépítését jobban megtanulhassa. Az ismeretlen szavak kezelését egy UNK token bevezetésével, míg a szótárméret túlcsordulását a szókészlet méretének korlátozásával valósították meg. A változó hosszúságú bemenetek kezelését pedig egy rögzített szekvenciahossz bevezetésével érték el. A szekvenciahossz standardizálására használt egyik módszer az úgynevezett padding, amely a szekvencia adott hosszúságra történő kiegészítését jelenti. Ehhez általában egy speciális karaktert használnak, amellyel a szekvenciahosszbeli eltéréseket feltöltik. Liu és Pan rendszere kétféle megközelítést is használ az egységes szekvenciahossz kialakítására. Egyfelől a fix szekvenciahossznál rövidebb sorozatokat padding segítségével kiegészítik, míg a hosszabb szekvenciákat ehhez a hosszhoz levágják a későbbi szekvenciaelemek elhagyásával.

Az általam készített absztrakt generáló rendszer implementálásához Zhedong Zheng[6] sequence-to-sequence forráskódjából indultam ki. Az architektúra egyirányú enkóderből, valamint Luong attentionnel [23] kiegészített beam search dekóderből állt, ahol mind az enkóder, mind a dekóder első rétege egy embedding réteg volt. A forráskód nem tartalmazott semmilyen adatábrázolási stratégiát, a modell tanítására szolgáló logikát, az eredmények visszamérésére használható metrikát, valamint a feltanított modellek és paramétereik lementését és visszatöltését végző kódrészletet sem. A saját rendszeremben megvalósítottam ezeket a hiányzó funkciókat, valamint néhány architektúrális változtatást is bevezettem az absztraktok finomítása érdekében: kétirányú enkóder, Bahdanau attention [18], Dropout, modell dimenzióinak korrigálása.

6 Github: Unclear about how to integrate AttentionWrapper with BeamSearchDecoder, https://github.com/tensorflow/tensorflow/issues/11904# (2019. okt.15.)

3.2.2 Architektúra

Az általam készített rendszer alapját a sequence-to-sequence architektúra képezi, amelynek bemenete a hosszú, összefoglalandó dokumentumot reprezentáló szekvencia, kimenete pedig a generált összefoglaló szekvenciája. Ezek részletes ábrázolásáról és az adathalmazról a 3.3-es fejezetben olvashatnak bővebben.

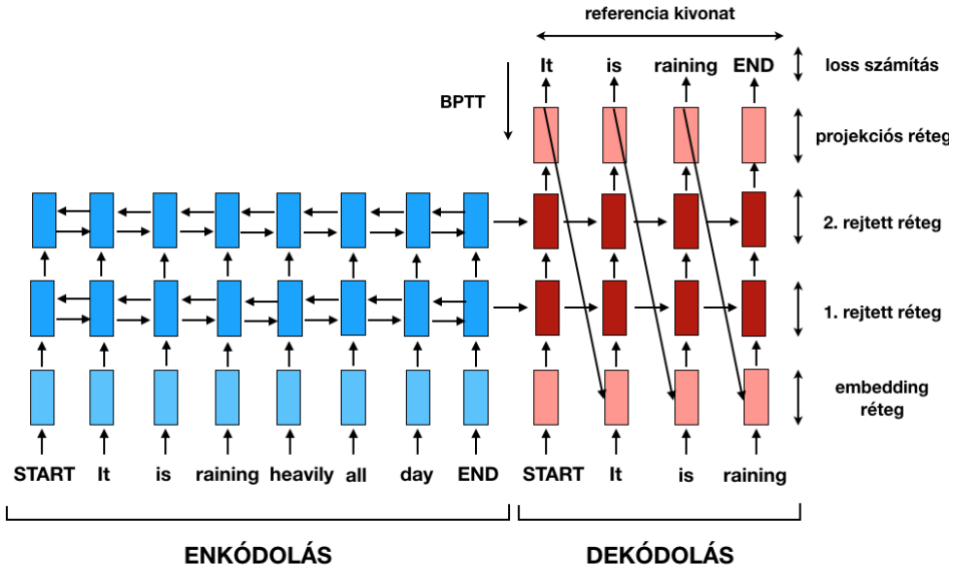


3.6. ábra: Architektúra felépítése

Az architektúra első rétegeként egy embedding réteget definiáltam az enkóder hatékonyabb működése érdekében. Ez a réteg a bemeneti szekvenciák előfeldolgozásáért felel, amely lehetővé teszi az adatok kifejezőbb ábrázolását, ezáltal extra információt szolgáltat a kódoló számára azok egymáshoz való viszonyáról. Az embedding réteget egy kétirányú enkóder követi, amelyet úgy terveztem, hogy egy szekvenciaelem feldolgozásakor az előtte és utána levő elemeket is figyelembe vegye. Ez a típusú kódoló nagyon jól illeszkedik a szövegösszegzés feladatához, ahol különösen fontos a kontextus vizsgálata, mert a kulcsgondolatok meghatározásában nagy szerepet játszanak a szavak és a szavak mondaton belüli szerepe, környezete. Az RNN cellák közül a modellben az LSTM-et használom, mert ezek jobban alkalmazhatók hosszú szekvenciák esetén. Az enkóder és a dekóder feldolgozási mechanizmusát a rendszerben szinkron módon valósítottam meg. Ez azt jelenti, hogy a dekóder nem az enkóder végső rejtett állapota, azaz a szekvencia utolsó elemének feldolgozásakor keletkező állapot alapján végzi a dekódolást, hanem attentiont használ. Ennek köszönhetően a kódoló által az adott időpillanatban feldolgozott szekvenciaelemből létrejött rejtett állapot (h_t) súlyozva (c_t) rögtön a dekóder bemenetére kerül. A súlyozás a 3.1.4-es fejezetben leírtak szerint történik, ami lehetővé teszi, hogy a dekóder a következő elem generálásakor az elem közvetlen környezetéből származó információt nagyobb súllyal vegye figyelembe.

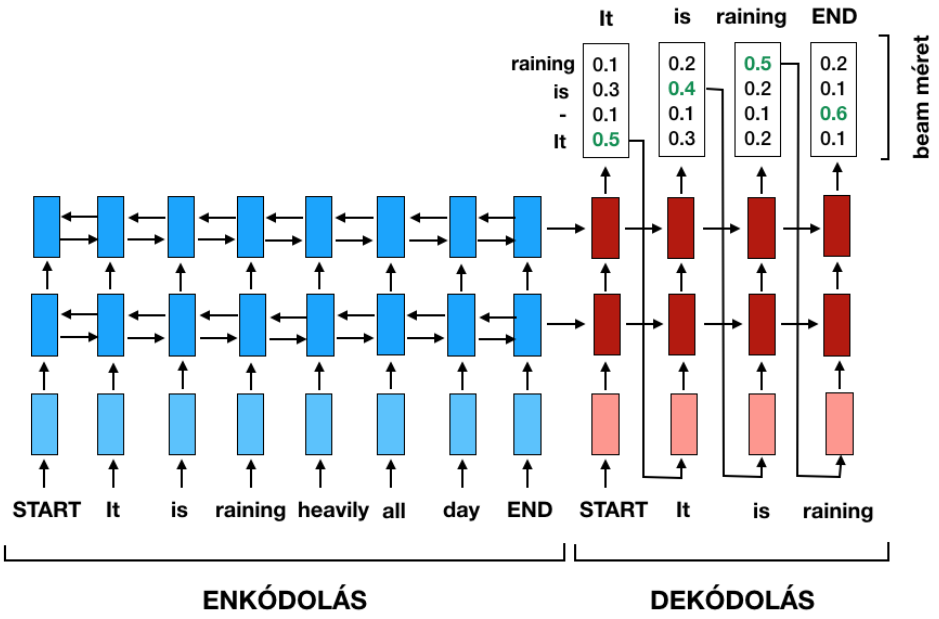
Például: „More people are learning English in Hungary than in Croatia.” esetén az „are” szó generálásakor a „people” és „learning” szavak relevánsabb információt hordoznak, mint a mondat távolabb elhelyezkedő szavai (pl. „More”).

A dekóder bemenetére kerülő adatokat először egy embedding réteggel dolgozom fel, amely a korábban létrehozott embedding mátrix alapján előállítja az egyes szekvenciaelemek vektor reprezentációját. Az adatokat innentől iteratív módon egy egyirányú dekóder dolgozza fel, amelynek minden elem generálásakor egy rejtett állapota jön létre (s_t). Ez a rejtett állapot segít a dekódolónak, hogy eltárolja az eddig generált sorozatból származó információkat. A rendszer működése során kétféle dekódert használtam, amelyek architektúrájuk és paramétereik tekintetében megegyeznek, mindössze a modellt adatokkal ellátó stratégiában különböznek [24].



3.7. ábra: Tanításhoz használt struktúra

A 3.7. ábra a tanításhoz használt struktúrát és annak működését mutatja be, amihez a hagyományos seq2seq egyirányú dekóderét használom. A tanítás során a kétirányú enkóder kinyeri a hosszú dokumentumokból származó információkat, a dekóder pedig megtanulja, hogy miként generáljon belőle rövid összefoglalókat. A tanítási fázisban a generált és elvárt kimenetek összehasonlítása alapján kiszámolom a loss értékeket, majd a backpropagation through time (BPTT) információterjesztő mechanizmus segítségével frissítem a neurális háló egyes rétegeinek súlyértékeit.



3.8. ábra: Teszteléshez használt struktúra

A 3.8. ábra ezzel szemben a predikcióhoz használt architektúrát mutatja be, amelynél beam search dekódert alkalmaztam. A beam search dekóder előnye ugyanis leginkább ebben a fázisban mutatkozik meg. Ez a típusú dekóder egy szekvenciaelem generálásakor több jelöltet is megtart, így képes a kontextusba leginkább illeszkedő elem megtalálására. A dekóder kimeneti rétegeként egy úgynevezett dense réteget definiáltam, amelynek mérete megegyezik a szókészletével. Ennek a rétegnek a kimenete az RNN által generált következő szekvenciaelem lesz. A rendszer kimenetén végül akkor jelenik meg az összefoglaló, amikor a dekódoló az utolsó szekvenciaelemet is előállította. A beam search dekóder esetén ekkor a rendszer több kimeneti szekvenciával rendelkezik, de közülük a legnagyobb valószínűséggel rendelkező lesz kiválasztva. Az egyetlen kimeneti számsorozatot ezután visszaalakítom szöveges formába a könnyebb ellenőrzés érdekében [25].

3.3 Adathalmaz és adatelőkészítés

3.3.1 Adathalmaz

Az általam kiválasztott korpusz az 1,3 millió összefüggő cikk és absztrakt párból álló Newsroom adathalmaz [26]. A korpuszt alkotó angol nyelvű adatok 38 különböző forrásból származnak, ahol az összefoglalók egyaránt ötvözik az absztrakt és kinyerés alapú stratégiákat. A Newsroom korpusz egy train, test és dev adathalmazból áll, melyek 995 041 db, 108 862 db és 108 837 db cikk-összefoglaló párt tartalmaznak. Az

adathalmaz azonban tartalmaz olyan adatokat, amelyek nem valódi cikkek vagy összefoglalók. Ezek közös jellemzője, hogy túlságosan rövidek: „© Telegraph Media Group Limited 2016”, „Mr Cameron is determined to resist moves for a paving Bill in the current parliament Photo: Reuters”, „Wow”, „X”. Ezek várhatóan az adathalmaz közösségi médiából és kereső rendszerekből történő gyűjtése során kerültek bele. Első lépésként ezért adattisztítást végeztem, amely során csak azokat a cikk-összefoglaló párokat tartottam meg, amelyeknél a cikk legalább 200 karakter, míg az összefoglaló legalább 10 karakter hosszú.

Adathalmaz	Cikk-összefoglaló párok száma	Cikkek átlagos karakterszáma	Absztraktok átlagos karakterszáma
train	992 921	3931	162
test	108 633	3896	163
dev	108 574	3903	164

3.1. táblázat: Szűrt Newsroom adathalmaz statisztikája

A 3.1. táblázat alapján a train esetén ez 2120 db, a testnél 229 db, míg a devnél 263 db bemenet eltávolítását jelentette. Ezenkívül a fenti táblázat az így létrejött adathalmazok átlagos karakterszámát is szemlélteti. A leghosszabb cikk 1 017 573 karakterből áll, a leghosszabb absztrakt pedig 35 425 karakter hosszú. A rendszert ezért fel kellett készítenem arra, hogy hosszabb és rövidebb sorozatokat is kezelni tudjon.

3.3.2 Adatelőkészítés és adatreprezentáció

Az adathalmaz változatos összetétele miatt számos adatelőkészítési lépést kellett végrehajtanom, hogy az adatokat a rendszer számára elvárt formátumra hozzam. A neurális háló közvetlenül nem képes a szöveges adatok feldolgozására, ezért azokat először számára érthető formába, számvektorokká kellett alakítanom.

Ehhez először az összefüggő szövegek adattisztítására volt szükségem, amelyek sok esetben üres sorokat, speciális karaktereket tartalmaztak: „• For help and support in recovering from burn injuries, contact Burn Centre Care.”. Ezek a speciális karakterek nem hordoztak releváns információt a rendszer számára a szöveg tartalmát illetően, viszont megnehezítették annak megértését és a megfelelő kimenet előállítását. Ezért a cikkekből és absztraktokból eltávolítottam a nem alfanumerikus karaktereket a szóköz, valamint az alapvető mondatvégi írásjelek (.?!) kivételével. Ezután a szöveg

értelmezéséhez azt kisebb vizsgálandó egységekre kellett szétválasztanom. Ehhez először a bemeneteket mondatokra bontottam, ahol a mondathatárok megtalálását a mondatvégi írásjelek után beszúrt szóköz karakter segítette. Az így létrejött mondatokat további elemi egységekké, szavakká (token) választottam szét, mivel ezek hordozzák a szöveg lényegét megragadó információt. Az így létrejött tokenek mondaton belüli szerepüknek megfelelően eltérő formában szerepelhetnek, ezért az eredeti tokenekhez (pontosabban a token szótári alakjához, az ú.n. lemmához) egy part-of-speech taget (POS, mondaton belüli szerep) is rendeltem.

Mondat	**Token**	**Mondaton belüli szerep (POS)**	**(kisbetűs lemma, POS)**
He went to school.	He	PRON	(’he’, ’PRON’)
	went	VERB	(’go’, ’VERB’)
	to	ADP	(’to’, ’ADP’)
	school	NOUN	(’school’, ’NOUN’)
	.	PUNCT	(’.’, ’PUNCT’)

3.2. táblázat: Adattárolási stratégia bemutatása

Minden szót innentől (kisbetűs lemma, POS) formában tároltam el, melyek a szekvencia alapegységeként jönnek szóba. A tárolási mechanizmust egy példa mentén a 3.2. táblázat mutatja be, ahol például a went szót (’go’, ’VERB’) pár reprezentálja.

Az absztrakt összefoglalók esetén a bemenetek és kimenetek szókészlete eltérő lehet, ezért a tömörítendő szöveg szókészlete további szavakkal kiegészíthető. Az általam készített megoldás a bemenetek (azaz a hosszú dokumentumok) és a kimenetek (azaz a rövid összefoglalók) szókészletét egyaránt használhatja, tehát a modell egyetlen összevont szótárral dolgozik. Ez a tanító halmaz szókészletét jelenti, amelyet a rendszer a tanítását követően a teszteléshez elmentek. Ha a teszt adatok olyan szót tartalmaznak, amelyek nem szerepelnek ebben a szótárban, akkor azokat egy UNK tokennel helyettesítem.

3.4 Eredmények

3.4.1 Metrika

A rendszer pontosságának kiértékeléséhez a 2.3-as fejezetben bemutatott ROUGE-t használtam, amelyet az alábbi Python csomag[7] implementál. Ez az implementáció a ROUGE-N metrika ROUGE-1 és ROUGE-2 megközelítését használja, valamint a ROUGE-L metrikát tartalmazza.

Ezekkel a visszamérési módszerekkel a kiértékelés során a generált és referencia összefoglalók lemmatizált alakját hasonlítom össze. A rendszer a tokenek lemmatizált formában történő tárolása miatt nyers, lemmákból felépülő kimeneteket generál. Az absztraktot alkotó szavak tehát toldalék nélküli formában szerepelnek: „he run home.”. Annak érdekében, hogy a generált összefoglaló jósága ne torzuljon, a referencia összefoglalót is lemmatizált formában tárolom el. Ellenkező esetben a generált „he run home.” és referencia „He runs home.” összefoglaló közti különbség az eredmény torzulásához vezetne. Az értékelés során minden generált és referencia összefoglaló párt a ROUGE-1, ROUGE-2 és ROUGE-L értékekkel jellemzek. Ezek mindegyikéhez három érték tartozik: F_1 érték, pontosság és fedés. A pontosság megmutatja a generált összefoglaló helyesen prediktált és összes n-gramjának arányát, ahol a helyesen prediktált n-gramok a referencia (n_gram_{Ref}) és generált (n_gram_{Gen}) összefoglaló közös n-gramjait jelentik. A fedés ezzel szemben azt adja meg, hogy a generált összefoglaló n-gramjaival mennyire fedhető le a referencia összefoglaló, azaz a referencia összefoglaló mekkora részét sikerült helyesen előállítania a rendszernek. Ez a fedés a két összefoglaló közös n-gramjainak és a referencia összefoglaló összes n-gramjának arányával számítható. Végül az F_1 érték a pontosság és fedés harmonikus közepeként kapható meg.

[7] PyPI: rouge 0.3.2, https://pypi.org/project/rouge/ (2019. okt.15.)

$$\text{pontosság} = \text{Average}\left(\frac{|(\text{n\_gram}_{\text{Gen}}) \cap (\text{n\_gram}_{\text{Ref}})|}{|(\text{n\_gram}_{\text{Gen}})|}\right)$$

$$\text{fedés} = \text{Average}\left(\frac{|(\text{n\_gram}_{\text{Gen}}) \cap (\text{n\_gram}_{\text{Ref}})|}{|(\text{n\_gram}_{\text{Ref}})|}\right)$$

$$F_1 = 2 * \frac{\text{pontosság} * \text{fedés}}{\text{pontosság} + \text{fedés}}$$

3.4. képlet: Rendszer jóságát jellemző pontosság, fedés és F_1 érték számítása

A rendszer jóságának jellemzésére az egyes cikk-összefoglaló párok ROUGE-1, ROUGE-2 és ROUGE-L értékeinek átlagát használom (ld. 3.4. képlet). Az F_1 értékre azért van szükségem, mert a pontosság és fedés alapján az egyes modellek nehezen vethetők össze: egyik modellnél magas pontosság, de alacsony fedés, másik modellnél alacsony pontosság, de magas fedés[8]. Az F_1 érték ennek a két értéknek a harmonikus közepe, ezért a modellek összehasonlítására használható.

3.4.2 Teszt eredmények

A modellek hosszú tanítási ideje miatt a modell paraméter-optimalizációjára szolgáló kísérletek közül csak néhányat sikerült végrehajtanom, amelyek elsősorban a tanításra használt adathalmaz méretében különböznek. Eszerint a modell tanítását a Newsroom korpusz tanító adathalmazának véletlenszerűen kiválasztott 200 és 500 elemén hajtottam végre, amely során a validációs adathalmaz 80, illetve 200 bemenetét használtam fel validációs célra. A feltanított modellek összehasonlítása érdekében a tanítást egy közös paraméterkombinációval végeztem (ld. 3.3. táblázat).

[8] Stack Overflow: what is f1-score and what its value indicates, https://stackoverflow.com/questions/45963174/what-is-f1-score-and-what-its-value-indicates (2019. okt.15.)

Paraméter	Vizsgált paraméterértékek
Batch méret	1
Neuron szám	128
Learning rate	0,001
Epoch	20
Beam méret	5
Embedding méret	128

3.3. táblázat: Vizsgált paraméterértékek

Tanító adathalmaz mérete	Metrika	Pontosság	Fedés	F_1 érték
200	ROUGE-1	0,193	0,055	0,082
	ROUGE-2	0,001	0,001	0,001
	ROUGE-L	0,185	0,051	0,055
500	ROUGE-1	0,041	0,011	0,016
	ROUGE-2	0,0	0,0	0,0
	ROUGE-L	0,041	0,011	0,011

3.4. táblázat: Modellek visszamérése a validációs adathalmazon

A 3.4. táblázat a modellek tanítása közben a validációs adathalmazon mért jósági értékeket tartalmazza. Ebből leolvasható, hogy például a 200 bemeneten tanított modell a ROUGE-1 metrika alapján 19,3 %-os átlagos pontosságot, 5,5 %-os átlagos fedést és 8,2 %-os átlagos F_1 értéket ért el. Ez azt az információt közvetíti a modellről, hogy a forrás dokumentum szavainak csak kis részét használja fel az összefoglaló generálásához. Ez az összefoglaló generáló rendszer erős absztrakt tulajdonságát jelzi, miszerint képes elvonatkoztatni a forrás szöveg szókészletétől. A jóval magasabb pontosság érték pedig azt mutatja meg, hogy a generált összefoglalót alkotó szavaknak átlagosan 19,3 %-a releváns, azaz átlagosan a szavak 19,3 %-ára valóban szükség van a helyes kimenet előállításához. A ROUGE-1 és ROUGE-2 metrika közötti nagy különbségből továbbá azt is megtudhatjuk, hogy bár a rendszer az összefoglalót alkotó szavakat külön-külön jól generálja, de a szomszédos szavakat szinte sosem sikerül eltalálnia. A modell számára a problémát az okozza, hogy a szavakat milyen sorrendben állítsa elő. Ezt a ROUGE-1 és ROUGE-L értékek közötti eltérés is megerősíti, mert ezek mindegyike szavankénti egyezést vizsgál. A különbséget a kettő között mindössze az jelenti, hogy a ROUGE-L a

szavak helyes sorrendjét is elvárja. A köztük lévő differencia ezért azt jelzi, hogy bár a modell a megfelelő szavakat generálja, ezek helyes sorrendjét nem tudja meghatározni.

A modellek tesztelését a teszt adathalmaz 5 darab, véletlen módon összeállított, külön-külön 100 bemenetet tartalmazó részhalmazán végeztem, amelynek eredményeit a **3.5**. táblázat tartalmazza. Ezzel a tesztelési stratégiával az volt a célom, hogy megvizsgáljam a modellek pontosságának szórását. Eközben azt tapasztaltam, hogy az F_1 értékek szórása az öt adathalmaz esetén 0 és 0,003 között mozog.

Tanító adathalmaz mérete	**Metrika**	**F_1 érték**				
		1	**2**	**3**	**4**	**5**
200	ROUGE-1	0,005	0,0	0,008	0,008	0,009
	ROUGE-2	0,0	0,0	0,0	0,0	0,0
	ROUGE-L	0,003	0,0	0,005	0,004	0,005
500	ROUGE-1	0,012	0,003	0,005	0,005	0,005
	ROUGE-2	0,0	0,0	0,0	0,0	0,0
	ROUGE-L	0,01	0,002	0,003	0,003	0,002

3.5**. táblázat: Modellek által elért eredmények az öt különböző teszt adathalmazon**

Tanító adathalmaz mérete	**Teszt adathalmaz mérete**	**Tanító adathalmaz szavainak száma**	**Teszt adathalmaz szavainak száma**	**Közös szavak száma**
200	500	18104	25872	9731
500		28142		12606

3.6. táblázat: Tanító és teszt adathalmazok szókészletének összehasonlítása

A tesztelés során elért jósági értékek alapján elmondható, hogy a validációs adathalmazon elért eredményekhez képest az F_1 értékekben akár 0,08-os visszaesés is előfordulhat. Ennek egyik oka, hogy a modell a teszteléshez is a tanító adathalmaz szókészletét használja fel. Ez a szótár a tanításhoz használt bemenetek elenyésző száma miatt kevés szót tartalmaz, melynek következtében a teszt adatokat alkotó szavak nagy része hiányzik belőle. Az ilyen szavakat a teszt adatok ábrázolásakor UNK tokennel helyettesítem, ami megnehezíti a modell számára a szöveg összefüggéseinek és szükséges szavainak megállapítását. A 3.6. táblázat a tanító és teszt adathalmazok szókészletét

hasonlítja össze, amelyből leolvasható, hogy például a 200 bemeneten tanított modell a teszt adatok ábrázolásához 9731 szót használ fel a 18104 elemű szótárból, míg a maradék 16141 ismeretlen szót UNK tokennel helyettesíti. Emiatt a teszt adatok szavainak 62,4 %-a UNK token formájában jelenik meg a modell számára [25].

4 Kinyerés alapú összefoglaló készítés tervezése és megvalósítása

4.1 Kulcsszó és kulcsmondat kinyerési módszerek

A kinyerés alapú összefoglalók generálása a 1-es fejezetben leírtaknak megfelelően kizárólag a dokumentum építőelemeinek felhasználásával történik. Ezért a forrásszöveg főbb tartalmi elemeinek kiválasztásához szükség van az egyes alkotóelemek relevancia szerinti differenciálására. Ennek érdekében a kinyerés alapú összefoglalókat generáló rendszerek a szöveg minden alkotóeleméhez egy pontszámot rendelnek, ami az adott elem fontosságát fejezi ki a tartalom szempontjából. A kivonat előállítása történhet a kulcsszavak, illetve a kulcsmondatok kinyerésével egyaránt, ami a legmagasabb pontszámot elért szavak vagy mondatok kiválasztását jelenti. Ebben a fejezetben az ilyen kulcsszó és kulcsmondat kinyerésére használt bevett módszerek közül ismertetek néhányat, illetve példák mentén szemléltetem a működésüket.

4.1.1 Term Frequency - Inverse Document Frequency (TF-IDF)

A TF-IDF (Term Frequency - Inverse Document Frequency) a szöveges dokumentumok információ kinyerésénél használt egyik legnépszerűbb algoritmus, amely a szöveg lényegének kiemelésére szolgál. Az algoritmus alapját az az elképzelés képezi, miszerint a szöveg tartalmát a benne ritkán előforduló szavak ragadják meg legjobban. A gyakran ismétlődő szavak (stopwords) ugyanis általában nem hordoznak érdemi információt a kulcsszókinyerés szempontjából: névelők, kötőszavak, névmások. A TF-IDF ezek kiszűrése érdekében minden szóhoz egy szógyakoriság alapú súlyértéket rendel, amely annál nagyobb értéket vesz fel minél ritkábban fordul elő a szó a szövegben, a dokumentumon belüli szógyakorisággal pedig egyenesen arányos. Az algoritmus által kiválasztott kulcsszavak végül a legnagyobb súlyértékkel rendelkező szavak lesznek.

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

4.1. képlet: Term Frequency (TF) kiszámítása [27]

A TF-IDF a súlyértékek kiszámításakor a szavak két fő szövegbeli tulajdonságát veszi figyelembe. Az egyik a 4.1. képlet által leírt $TF_{i,j}$ tényező, ami megadja az i. szó

gyakoriságát a szöveget alkotó j. dokumentumban. Ehhez megvizsgálja hányszor fordul elő a szó az adott dokumentumban ($n_{i,j}$) a dokumentumot alkotó szavak számához képest ($\Sigma_k n_{k,j}$).

$$IDF_i = log(\frac{N}{df_i})$$

4.2. képlet: Inverse Document Frequency (IDF) kiszámítása [27]

$$TF - IDF_{i,j} = TF_{i,j} * IDF_i$$

4.3. képlet: TF-IDF kiszámítása [27]

A másik tényező az IDF_i érték, ami az i. szó szövegbeli relevanciáját írja le. Az IDF_i érték (ld. 4.2. képlet) a szöveget alkotó összes dokumentum számából (N) és az i. szót tartalmazó dokumentumok számából (df_i) számítható [27]. A szavakhoz tartozó TF-IDF érték kiszámítása végül a 4.3. képlet szerint a két tényező szorzataként kapható meg.

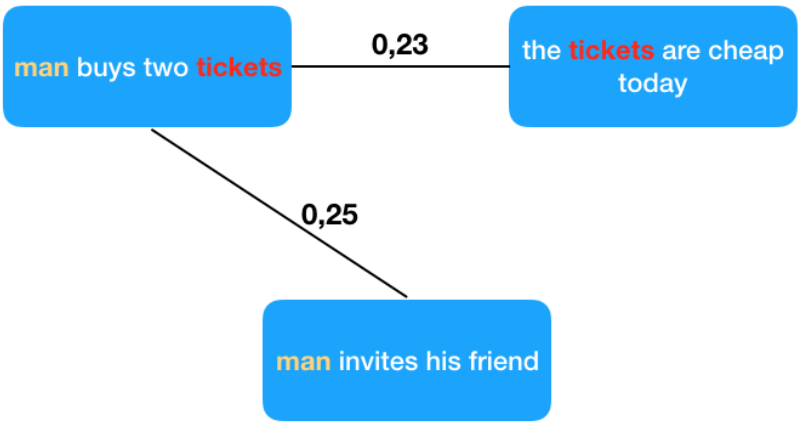
4.1.2 TextRank

A TextRank egy olyan gráf alapú algoritmus, amely kiszámítja a szöveg alkotóelemeinek szemantikai fontosságát. Az alapgondolata a PageRank [28] algoritmusból származik, ami az egyes weboldalak fontosságának megállapítására szolgál. A fontosság a PageRank esetében azt jelenti, hogy milyen valószínűséggel fogja a felhasználó meglátogatni a weboldalt, míg a TextRanknél azt fejezi ki mennyire fontos az adott elem a dokumentum tartalmának leírásában. Ennek megfelelően a TextRank algoritmusnál a gráf csomópontjai nem a weboldalak, hanem a szöveg szavai vagy mondatai annak függvényében, hogy a kulcsszavak vagy a kulcsmondatok kiválasztása a cél [29]. A csomópontok fontosságának meghatározásában a köztük menő élek játszanak kulcsszerepet. Ha két csomópont szomszédos a gráfban akkor az él kezdőpontjaként viselkedő csúcs arra szavaz, hogy az él végpontját jelentő csomópont fontos a tartalom összefoglalása szempontjából: man – buys él esetén a man szó növeli a buys token fontosságát, a buys szó pedig a man értékét. Egy csúcs annál fontosabb lesz, minél több szavazattal rendelkezik és minél fontosabb csomópontok szavaztak rá [8].

A TextRank a dokumentumot először a feladat függvényében szavakra vagy mondatokra bontja, majd megépíti belőle a szöveg szemantikai kapcsolatait reprezentáló gráfot. Ezután iteratív módon kiszámítja a gráf csomópontjainak fontosságát. Egy csúcs minél magasabb értéket ér el, annál fontosabb a tartalom leírása szempontjából. Végül a

csúcsokat fontosság szerint csökkenő sorrendbe rendezi és kiválasztja az N darab legnagyobb pontszámot elért csomópontot, amelyeket egymás után fűzve előáll a kivonat. A TextRank algoritmus előnye tehát az, hogy egy adott csomópont fontosságát a gráf egészéből rekurzív módon nyeri, nem kizárólag lokális információkra támaszkodik. Ennek köszönhetően nem igényel nyelvspecifikus ismereteket a kulcsmondatok és kulcsszavak kinyeréséhez, így akár címek vagy hosszabb összefoglalók generálására is alkalmas [8].

A kulcsmondatok kinyerésénél nem használható az imént bemutatott élmeghatározó módszer. Ennek oka, hogy a csomópontok néhány szónál lényegesen hosszabbak [8]. Ezért a gráfban két csomópont akkor lesz szomszédos egymással, ha az őket alkotó mondatok közös szóval rendelkeznek, a gráf élei pedig a csúcsok átlapolódásának mértékével súlyozódnak.



4.1. ábra: Gráfépítés működése kulcsmondatok kinyerésénél

$$E(S_i, S_j) = \frac{\left|\{w_k \mid w_k \in S_i, w_k \in S_j\}\right|}{log_2(|S_i|) + log_2(|S_j|)}$$

4.4. képlet: Két csomópont közti él súlyának kiszámítása [8]

A 4.1. ábra a gráfépítés működését szemlélteti egy példán, ahol a szomszédos csomópontok közös szavai kiemelve szerepelnek. Az élsúlyok számítása a 4.4. képlet szerint történik, miszerint a „man buys two tickets” (S_i) és „man invites his friend” (S_j) csúcsokat összekötő él súlya 0,25, mert a két mondat mindössze a man szóban egyezik meg (w_k), s mindkettő egyaránt négy szóból áll: 0,25 = 1 / ($\log_2 4 + \log_2 4$).

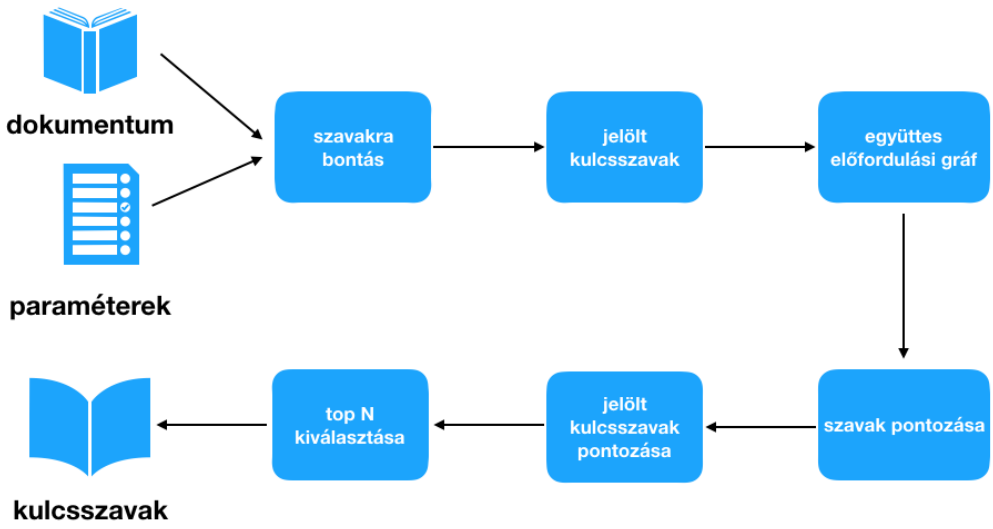
$$S^k(V_i) = (1-d) + d * \sum_{V_j \in In(V_i)} \frac{E_{ji}}{\sum_{V_m \in Out(V_j)} E_{jm}} S^{k-1}(V_j)$$

4.5. képlet: Fontosság számítása kulcsmondatok kinyerésénél [8]

Az algoritmus ezután kiszámítja a csúcsokban elhelyezkedő mondatok fontosságát. Ehhez a csomópontok mindegyikéhez 1-es kezdő fontosság értéket rendel, majd a 4.5. képlet alapján minden iterációban frissíti az értéküket, ahol az E_{ji} a j. csúcsból az i. csúcsba menő él súlyát jelenti.

4.1.3 Rapid Automatic Keyword Extraction (RAKE)

A RAKE (Rapid Automatic Keyword Extraction) egy automatikus kulcsszó kinyerésre szolgáló algoritmus, amely a kulcsszavak meghatározásakor a szavak gyakoriságát és együttes előfordulását vizsgálja. A RAKE a korábbi fejezetekben említett TF-IDF és TextRank módszerektől eltérő módon képes több szóból álló kifejezéseket is összetartozó kulcsszóként kiválasztani, ezért alkalmas a szöveg tartalmának pontosabb összefoglalására. Az összetartozó szókapcsolatok megállapításakor arra támaszkodik, hogy a több szóból álló kulcsszavak ritkán tartalmaznak stopszavakat, illetve írásjeleket: például and, the, of. Ezért a szöveg megfelelő strukturálásának érdekében az algoritmus által elvárt bemeneti paraméterek a stopszó és a kifejezéshatárolójelek listája [7].



4.2. ábra: RAKE algoritmus működése

Az algoritmus működésének főbb lépéseit (ld. 4.2. ábra a „Compatibility of systems of linear constraints over the set of natural numbers” mondat példáján ismertetem. Első lépésként a RAKE az összefüggő szöveget felbontja szavakra:

compatibility, of, systems, of, linear, constraints, over, the, set, of ,natural, numbers. Ezután a szavak listájában megvizsgálja, hogy a szomszédos szavak közül melyek vonhatók össze egyetlen összetartozó kifejezéssé. Ha a szomszédos szavakat olyan karakter vagy szó határolja, amely a bemenő paraméterként megadott stopszó vagy kifejezéshatárolójelek listájában szerepel, akkor azokat összevonja egyetlen kulcsszóvá, amit innentől jelölt kulcsszónak tekint. A példában „linear” és „constraints” szavakat az „of” és „over” stopszavak határolják, ezért azok összevonhatók egyetlen kifejezéssé. Az összevonások után a jelölt kulcsszavak listája a következő: compatibility, systems, linear constraints, set, natural numbers [7].

	compatibility	systems	linear	constraints	set	natural	numbers
compatibility	1	0	0	0	0	0	0
systems	0	1	0	0	0	0	0
linear	0	0	1	1	0	0	0
constraints	0	0	1	1	0	0	0
set	0	0	0	0	1	0	0
natural	0	0	0	0	0	1	1
numbers	0	0	0	0	0	1	1

4.1. táblázat: Példa együttes előfordulási gráfja [7]

Ezt követően a jelölt kulcsszavak szavaiból elkészíti az együttes előfordulási gráfot. A 4.1. táblázat a példa együttes előfordulási gráfját mutatja be, ahol az egyes mezők az i. sor és j. oszlop szavainak közös előfordulási számát írják le. Eszerint a [linear,linear] = 1, mert a „linear” szó egyszer fordul elő a jelölt kulcsszavak listájában a „linear constraint” kifejezés tagjaként. Ezenkívül a linear szó sorában a [linear,constraints] = 1, mert a jelölt kulcsszó lista egy darab „linear constraint” kifejezést tartalmaz, ahol a linear és constraint szavak együtt szerepelnek.

	compatibility	systems	linear	constraints	set	natural	numbers
deg(w)	1	1	2	2	1	2	2
freq(w)	1	1	1	1	1	1	1
deg(w)/freq(w)	1	1	2	2	1	2	2

4.2. táblázat: Példa szavainak pontozása [7]

Az együttes előfordulási gráf alapján ezután a gráfot alkotó szavak pontozása következik. Ez történhet szógyakoriság alapján (freq(w)), ami a gráf átlójában szereplő értékeket jelenti: [linear,linear] = 1, ezért freq(linear)=1. A másik pontozási módszer a szavakhoz tartozó gráf fokszám (deg(w)), ami a szó sorában szereplő értékek összegét jelenti a gráfban: [linear] = 1 + 1. A pontozásnál ezentúl lehetőség van az előbbi két módszer kombinálására is. A 4.2. táblázat a példa szavainak pontozását tartalmazza a három módszer szerint. Az algoritmus végül megállapítja a jelölt kulcsszavak pontszámát, ami az őket alkotó szavak pontszámainak összegéből áll elő. A jelölt kulcsszavak pontszámának megállapítására a példában a deg(w)/freq(w) módszert használva azt kapjuk, hogy: compatibility (1), system (1), linear constraints (4), set (1), natural numbers (4). Az algoritmus kimenete végül az N darab legnagyobb pontszámmal rendelkező jelölt kulcsszó lesz: N = 2 esetén a „linear constraints” és „natural numbers”[7].

4.2 Rendszer által generált kivonatok

A kinyerés alapú összefoglaló generáló rendszer megvalósításakor arra törekedtem, hogy egy olyan megoldást készítsek, amely ötvözi a már kész rendszerek által használt bevett módszereket. Ennek érdekében a feladatot többféle nézőpontból is megvizsgáltam, mely során a kivonatok előállításához kulcsszó és kulcsmondat alapú megközelítéseket is felhasználtam.

4.2.1 Kulcsszó alapú kivonatok

A rendszer által generált összefoglalók első típusánál kizárólag kulcsszavak kiválasztására szolgáló algoritmusokat alkalmaztam. Ezek az algoritmusok a szöveget

alkotó szavak különböző tulajdonságai alapján kiválasztják a tartalom szempontjából legrelevánsabb kulcsszavakat, amelyeket a rendszer összefűz egyetlen kivonattá. A rendszer implementálásakor kétféle ilyen kulcsszó kinyerésre szolgáló módszert használtam.

Az első a 4.1.1-es fejezetben bemutatott TF-IDF volt, melyet az Sklearn[9] Python könyvtár segítségével építettem be a rendszerbe. Az algoritmus működése azonban bizonyos tekintetben korlátozott. Egyfelől nem képes az eltérő szóalakok kezelésére, ezért előfeldolgozást igényel: dog és dogs szavakat két külön szóként értékeli. Másrészt nem vizsgálja a szavak szemantikai kapcsolatait, ezért több szóból álló kifejezések kinyerésére nem alkalmas: linear equation kifejezést külön linear és equation szavakként ismeri fel. A másik általam használt kulcsszó kinyerő módszer a 4.1.3-as fejezetben ismertetett RAKE volt, amely erre a problémára kínál megoldást. A RAKE ugyanis amellett, hogy a TF-IDF-hez hasonlóan kis számításigényű, szógyakoriság alapú algoritmus, képes a szomszédos szavak szemantikai kapcsolatának vizsgálatára is. Ennek köszönhetően az általa előállított kulcsszó listában megjelennek olyan összetett kifejezések is, amelyek a szöveg szomszédos, összetartozó szavaiból épülnek fel. A RAKE algoritmust a Rake-nltk[10] Python könyvtár felhasználásával építettem be a rendszerbe a kivonatok jobb tartalom lefedése érdekében.

4.2.2 Kulcsmondat alapú kivonatok

A rendszer által generált összefoglalók másik típusát a kulcsmondatok kinyerésével előállított kivonatok képezik. Ezek a kivonatok a forrás szövegből kiválasztott mondatok egymás után fűzésével jönnek létre, ezáltal sokkal kohézívabb, könnyebben érthető összefoglalót eredményeznek. Ennek oka, hogy a kulcsszavakat az algoritmus a szöveg különböző mondataiból, bekezdéseiből válogatja össze, a kulcsmondatok kiválasztásakor pedig az egyes mondatok egész egységként kerülnek át a kivonatba. A rendszer megvalósításakor ezért két ilyen kulcsmondatokon alapuló algoritmust is kipróbáltam.

Ezek közül az egyik a 4.1.2-es fejezetben bemutatott TextRank algoritmus volt. Ez az algoritmus a kinyerés alapú összefoglalót generáló rendszerek alapját képezi, mert

9 Scikit learn: sklearn.feature_extraction.text.TfidfVectorizer, https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (2019. szept. 16.)
10 PyPI: rake-nltk 1.0.4, https://pypi.org/project/rake-nltk/ (2019. okt.15.)

a szöveg szemantikai kapcsolatait nem kizárólag lokális információkra támaszkodva vizsgálja. A szöveg egyes mondatainak fontosságának megállapításakor nem csak a szomszédos mondatokat veszi figyelembe, hanem a dokumentumot alkotó összes mondatot. Ezalapján képes a szövegből a tartalom leírására leginkább alkalmas kulcsmondatok kiválasztására. Az algoritmust a Gensim[11] Python könyvtár felhasználásával építettem be a rendszerbe.

A TextRank használatának negatív oldala, hogy komplex gráf alapú algoritmus révén nagy számítási kapacitást igényel, ezért egy kisebb erőforrásigényű, bevett módszert is megvizsgáltam. A súlyozott gyakoriság módszere összeszámolja minden szóhoz, hogy az egész szövegben hányszor fordul elő. Ezt követően minden szóhoz egy súlyozott gyakoriságot kalkulál, úgy hogy a szóhoz tartozó gyakoriság értéket elosztja a szövegben legtöbbször előforduló szó gyakoriságával. Minden mondatra kiszámol egy pontszámot, ami a benne előforduló szavak súlyozott gyakoriságának az összege. Ezalapján a rendszer végül az N darab legnagyobb értékkel rendelkező mondatból előállítja a kivonatot[12]. A módszer tehát mindössze a szógyakoriságok vizsgálatával képes a kulcsmondatok kiválasztására, azonban emiatt a hosszabb, több szóból álló mondatok tartalom leíró szerepüktől függetlenül nagyobb pontszámmal rendelkezhetnek, ami az összefoglaló pontatlanságához vezethet.

4.2.3 Kulcsszó és kulcsmondat alapú kivonatok

A rendszer által generált összefoglalók harmadik típusát azok a kivonatok alkotják, amelyek előállítása a kulcsszavak és kulcsmondatok kombinálásával történik. Ezek a típusú összefoglalók ötvözik a kulcsmondat alapú kivonatok kohézívabb tartalom leírását és a kulcsszó alapú módszerek által biztosított jobb tartalomfedést. A tisztán kulcsmondatokra épülő kivonatok ugyanis sok esetben csak a tartalom egy részét fedik le, mert a rendszer csak a szöveg bizonyos részeiből (mondataiból) állítja össze az összefoglalót, míg a kulcsszavakat a szöveg egészéből válogatja össze. A két

11 Gensim topic modelling for humans: summarization.summarizer – TextRank Summariser, https://radimrehurek.com/gensim/summarization/summariser.html (2019. okt. 15.)

12 Stack Abuse: Text Summarization with NLTK in Python, https://stackabuse.com/text-summarization-with-nltk-in-python/ (2019.okt. 15.)

megközelítés kombinálásakor a kiválasztott kulcsszavak, valamint a kulcsmondatokat alkotó szavak halmazának metszetét és unióját vizsgáltam.

A metszet művelet tulajdonképpen a közös szavak tartalom leíró szerepét erősíti meg. Azok a szavak, amelyek egyaránt előfordulnak a kulcsszó és kulcsmondat kinyerő módszerek szólistájában is lényegesebbek a kivonat szempontjából. Ezzel szemben az unió műveletet azért vizsgáltam, mert az egyik módszer által kinyert szavak halmazát kiegészíti a másik módszer szavaival az összefoglaló jobb tartalom lefedése érdekében.

4.3 Adatok

4.3.1 Adathalmazok

A kinyerés alapú kivonat generáló rendszer megvalósításakor kétféle adathalmazt használtam fel, amelyek két különböző feladat elé állítják a rendszert: a szöveghez cím, valamint több mondatból álló összefoglaló generálása.

A cím generálási feladathoz a DUC 2003[13] (Document Understanding Conference) adathalmazt alkalmaztam. Ez a korpusz 624 egy darab hosszú mondatból álló bemenetet tartalmaz, amelyek mindegyikéhez 4 darab egyetlen rövid mondatból álló emberek által készített összefoglaló tartozik. A négy összefoglaló a tartalom leírása szempontjából azonos információt tartalmaz, mindössze a megfogalmazásban térnek el egymástól: „Magnetic pulse series sent through brain may ease schizophrenic voices” és „Yale finds magnetic stimulation some relief to schizophrenics' imaginary voices.”. Ennek köszönhetően a kinyerés alapú kivonatozó rendszer nagyobb eséllyel állít elő olyan összefoglalót, amely a megfelelő kifejezéseket tartalmazza. A DUC 2003 adathalmaz tehát összesen 2496 darab ilyen adatpárt biztosít a rendszer számára.

	Bemenet	**Kivonat1**	**Kivonat2**	**Kivonat3**	**Kivonat4**
szóhossz	9-377	4-15	6-13	5-16	7-14
karakterhossz	43-2138	39-100	44-105	36-101	47-115

4.3. táblázat: DUC 2003 adathalmaz statisztikája

13 DUC 2003: Documents, Tasks, and Measures, https://duc.nist.gov/duc2003/tasks.html (2019.okt.15.)

A 4.3. táblázat: DUC 2003 adathalmaz statisztikája

az adathalmaz részletes statisztikáját tartalmazza, amelyből leolvasható, hogy a bemenetek és a kivonatok hossza egyaránt változó. A korpusz további érdekessége, hogy bizonyos kivonatok kulcsszó felsorolásnak felelnek meg: “DNA, Chromosome, Brzustowicz, drugs, genetic, environmental, schizophrenia, anti-psychotic, pregnancy, statistical”. Az ilyen típusú összefoglalókat úgy szűrtem ki az adathalmazból, hogy a legalább 6 darab felsorolást (vesszőt) tartalmazó adatpárokat eltávolítottam. A szűrt adathalmaz így végül 2321 darab bemenet-összefoglaló párt tartalmazott.

A másik feladathoz, a több mondatból álló összefoglalók generálásához a 4.3.1-es fejezetben ismertetett Newsroom korpuszt használtam. Ez az adathalmaz a hosszú, összefüggő dokumentumokhoz több mondatból álló összefoglalókat tartalmaz, amelyet többek között azért alkalmaztam, hogy az absztrakt és kinyerés alapú kivonatozó rendszeremet összehasonlíthassam. A Newsroom adathalmazról bővebben a 3.3.1-es fejezetben, az absztrakt kivonatozó rendszernél olvashatnak.

4.3.2 Adatelőkészítés

A két adathalmaz által tartalmazott bemenetek összefüggő, zajos szöveg formájában állnak rendelkezésre. Ebben a formában a korpusz nem alkalmas a kinyerés alapú kivonatozó rendszer számára, ezért a DUC és Newsroom adathalmazokon egyaránt ugyanazokat az adatelőkészítési lépéseket végeztem el. Ezekre a természetes nyelvű szövegekre jellemző zaj eltávolítása miatt, valamint a főbb tartalmi elemek könnyebb azonosítása érdekében volt szükségem.

Első lépésként a mondathatárok könnyebb felismerése érdekében a mondatvégi írásjelek után egy szóköz karaktert illesztettem be. Az így struktúrált szöveget ezután kétféle megközelítés szempontjából is megvizsgáltam: alfanumerikus és alfabetikus karakterek megőrzése. Az alfanumerikus irányzatnál a szövegből mindössze az angol abc betűit, a számokat, a szóköz karaktert, valamint az alapvető mondatvégi írásjeleket tartottam meg. A mondatvégi írásjelek, valamint a szóköz karakter megőrzésére azért volt szükségem, hogy a mondat- és szóhatárok pontos meghatározását később elősegítsék. Az alfabetikus megközelítés ettől csak annyiban különbözik, hogy a speciális karaktereken túl a számokat is eltávolítottam a szövegből. A számok ugyanis bizonyos esetekben az összefoglaló szempontjából lényeges információt hordoznak, néha azonban megtévesztik a kivonatozó rendszert. Az így létrejött szűrt szöveget ezt követően kisebb vizsgálandó

egységekre bontottam: először mondatokra, majd szavakra. Ezt a szavak egységes formára hozása követte, hogy a rendszer a különböző szóalakokat ne különböző szavakként értékelje: Dog és dogs. Ennek érdekében a szavakat egységesen kisbetűs formájúra alakítottam, majd minden szót a szótári alakjával helyettesítettem (lemmatizálás). Végül az eddig mondatonként külön tárolt szavakat egyetlen tömbbe vontam össze, hogy a kivonatozó ne külön a mondatokból válassza ki a kulcsszavakat / kulcsmondatokat, hanem a szöveg egészéből. Ezeket az adattisztítási lépéseket a hosszú, összefüggő szövegen kívül az adathalmazt alkotó összefoglalókon is elvégeztem. Erre azért volt szükségem, mert a rendszer a kulcsszavakat / kulcsmondatokat az előfeldolgozott szövegből nyeri ki, ezért azzal megegyező formában fűzi össze őket összefoglalóvá. Emiatt a kiértékeléskor az eredmény torzulásához vezetne, ha a szövegben a szavak lemmatizált alakjukkal szerepelnének, míg az összefoglaló toldalékolt formában tartalmazná őket.

4.4 Eredmények

A rendszer által generált összefoglalók kiértékeléséhez a 2.3.1-es fejezetben ismertetett, összefoglalók automatikus kiértékelésére szolgáló ROUGE-N metrika ROUGE-1 megközelítését használtam.

4.4.1 Cím generálási feladat eredményei

A cím generálási feladatnál a rendszer jóságát a DUC 2003-as adathalmazon értékeltem ki. A korpusz összetétele miatt a feladatot kizárólag kulcsszó alapú kivonatok generálásával lehetett megvalósítani. A forrás szöveg ugyanis egyetlen hosszú mondatból épül fel, ezért nincs lehetőség a kulcsmondatok kiválasztásával rövidebb összefoglaló előállítására. Az általam készített rendszer ezért a TF-IDF és RAKE kulcsszavazó algoritmusokat használja a kivonatok összeállításához.

Módszer	Kulcsszavak száma	ROUGE-1 F_1 érték (alfanumerikus)	ROUGE-1 F_1 érték (alfabetikus)
TF-IDF	10	19,04 %	19,49 %
	15	21,7 %	22,0 %
	20	22,6 %	22,8 %
RAKE	10	22,8 %	23,05 %
	15	23,01 %	23,19 %
	20	22,9 %	23,18 %

4.4. táblázat: kinyerés alapú kivonatozó rendszer által a DUC 2003 adathalmazon elért eredmények

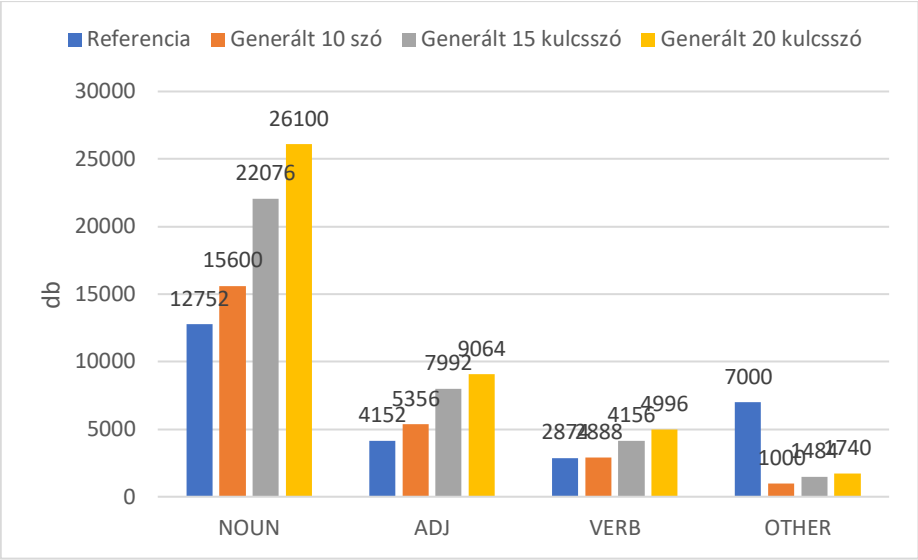
A 4.4. táblázat a cím generáláshoz alkalmazott módszereket, a kivonathoz felhasznált kulcsszavak számát, valamint a hozzá tartozó F_1 jósági értékeket tartalmazza a 4.3.2-es fejezetben ismertetett alfanumerikus és alfabetikus esetekre. A DUC 2003 adathalmaz összefoglalói a kétféle adatelőkészítési megközelítésnél átlagosan 10,8 és 10,6 szóból épülnek fel, ezért a generált kivonatok finomítása érdekében az algoritmusok 10, 15 végül 20 kulcsszót nyernek ki a szövegből. További paraméterértékeket azért nem vizsgáltam, mert a leghosszabb összefoglaló is legfeljebb 16 szóból áll (ld. 4.3. táblázat).

A 4.4. táblázat alapján az előkészített adatok közül az algoritmusok a legjobb eredményt a kizárólag alfabetikus karaktereket, szóköz karaktert és mondatvégi írásjeleket tartalmazó megközelítésen érték el. Az eredmények kiértékelését ezért a továbbiakban az ily módon előkészített adatok mentén végeztem. A táblázat alapján a két módszer közül ezen a korpuszon a RAKE ért el nagyobb jósági értéket, ami egyrészt az adathalmaz szövegeinek természetéből, valamint a két módszer által kinyert kulcsszavak szövegbeli szerepéből (POS) adódik. A DUC 2003 szövegei ugyanis részben tudományos természetűek, ezért számtalan összetett kifejezést tartalmaznak: magnetic field, mental illnesses. A TF-IDF algoritmus azonban csak különálló szavak kinyerésére alkalmas, a RAKE pedig képes több szóból álló kulcsszavak előállítására is, ezért a referencia kivonattal nagyobb átfedést érhet el.

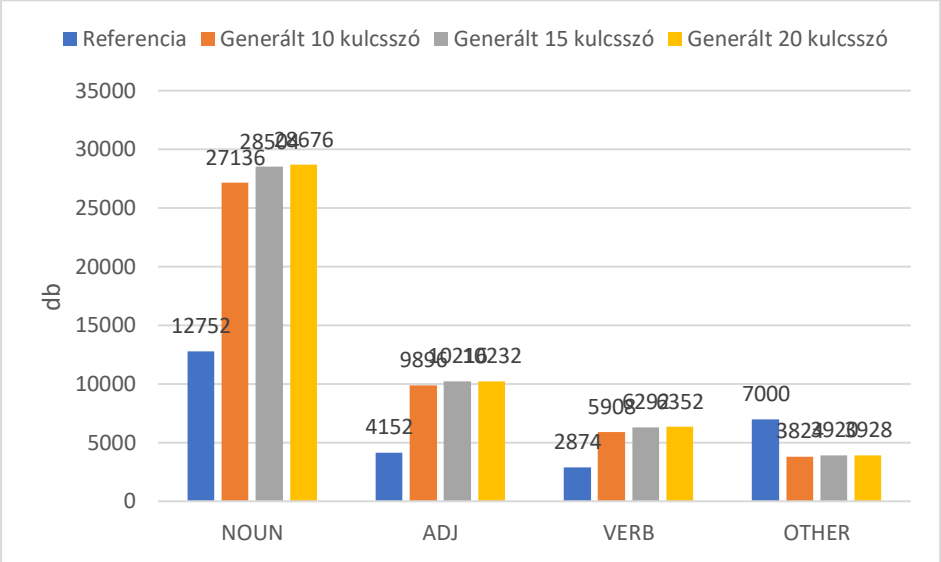
Összefoglalóhoz kinyert kulcsszavak száma	TF-IDF átlagosan RAKE kulcsszavak hány százalékát fedi le	RAKE összetett kifejezései hányszor fordulnak elő referencia összefoglalókban	TF-IDF hány százalékát fedi le RAKE referencia összefoglalókban előforduló összetett kifejezéseinek
10	52,6 %	17510	3,5 %
15	75,9 %	17530	4,9 %
20	86,9 %	17530	5,5 %

4.5. táblázat: TF-IDF és RAKE összehasonlítása kinyert kulcsszavak lefedettsége alapján

A 4.5. táblázat a TF-IDF-et és a RAKE-t hasonlítja össze a kinyert kulcsszavak lefedettsége szerint. Eszerint a TF-IDF találatai átlagosan a RAKE által kigyűjtött kulcsszavak szavainak 52-86 %-át tartalmazzák, az ezen kívül eső szavak általában az összetett kifejezések egyik tagjaként fordulnak elő. Ez a találati arány az egyik oka annak, hogy a két módszer F_1 értékeiben akár 3-4 %-os eltérés is mutatkozik (ld. 4.4. táblázat). A 4.5. táblázat továbbá azt is megmutatja, hogy körülbelül 17 ezer összetett kulcsszó található a referencia összefoglalókban, melyek szavainak mindössze 3,5-5,5%-át tartalmazza a TF-IDF által kinyert kulcsszólista.



4.1. diagram: DUC 2003 adathalmazon referencia és TF-IDF által generált összefoglalók POS statisztikája



4.2. diagram: DUC 2003 adathalmazon referencia és RAKE által generált összefoglalók POS statisztikája

A RAKE magasabb jósági értékeinek másik oka a két módszer által kinyert kulcsszavak szövegbeli szerepéből (POS), azaz a kinyert kulcsszavak szófajából adódik. A 4.1. diagram és 4.2. diagram a referencia és generált kivonatokat alkotó szavak szófajának eloszlását mutatja be a TF-IDF és a RAKE különböző vizsgált paraméterértékei mentén. Eszerint a referencia és generált összefoglalókban leginkább a főnevek (NOUN), melléknevek (ADJ) és igék (VERB) dominálnak, amelyek jellemzően a szöveg lényegét megragadó információt hordozzák. Az egyéb (OTHER) kategória az ADP, PUNCT, PART, ADV, CCONJ, NUM, DET, PROPN, PRON, X, INTJ, SPACE

tagekkel ellátott szavakat foglalja magában. A TF-IDF esetén a három domináns szófaj a kinyert kulcsszavak 95 %-át teszi ki, a RAKE-nél pedig 91-92 %-ot. Ezalapján azt várnánk, hogy a TF-IDF ér el jobb eredményt, azonban a RAKE a több szóból álló kulcsszavak miatt sokkal több kinyert szóval rendelkezik. Ez azt eredményezi, hogy például a 10 kulcsszóból generált összefoglalók esetén 80 %-kal több főnevet, melléknevet és igét gyűjt ki, mint a TF-IDF. Ez is nagy mértékben hozzájárul ahhoz, hogy a két módszer közül a RAKE alkalmasabb a DUC 2003 adathalmaz szövegeinek összefoglalására.

4.4.2 Több mondatból álló kivonat generálásának eredményei

A rendszer által generált több mondatos összefoglalók jóságát a Newsroom adathalmazon mértem vissza. A korpuszt alkotó összefoglalók előállítása a forrás szövegek felépítése miatt a kulcsszavak és kulcsmondatok kinyerésével egyaránt történhet. Ezért a cím generálási feladattal való összevetés érdekében a kulcsszavazó algoritmusok közül az ott alkalmazott TF-IDF-et és RAKE-et használtam, míg a tartalmat legjobban leíró mondatok kiválasztását a 4.2.2-es fejezetben ismertetett TextRank és súlyozott gyakoriság módszerével valósítottam meg.

Módszer	Paraméter	ROUGE-1 F_1 érték (alfanumerikus)	ROUGE-1 F_1 érték (alfabetikus)
TF-IDF	10 szó	3,6 %	4,7 %
	37 szó	9,1 %	9,9 %
	50 szó	10,5 %	11,2 %
RAKE	10 szó	13,4 %	13,7 %
	37 szó	15,3 %	15,5 %
	50 szó	15,2 %	15,5 %

4.6. táblázat: kulcsszavazó algoritmusok által a Newsroom adathalmazon elért eredmények

A 4.6. táblázat a rendszer által generált kivonatok első típusánál, a tisztán kulcsszavak kinyerésével generált összefoglalóknál alkalmazott módszereket, a kivonathoz felhasznált kulcsszavak számát és a hozzá tartozó F_1 értékeket tartalmazza. A kiértékelést a cím generálási feladattal megegyező módon a korpusz 2496 darab szöveg-

összefoglaló párból álló részhalmazán végeztem. A kulcsszavazó algoritmusoknál 10, 37 vagy 50 szót használok a kivonathoz, mert az alfanumerikus és alfabetikus megközelítésnél a Newsroom adathalmaz összefoglalói átlagosan 37,8, illetve 36,8 szóból épülnek fel. A táblázat alapján elmondható, hogy a kulcsszó alapú módszerek a kizárólag alfabetikus karaktereket, szóköz karaktert, valamint mondatvégi írásjeleket tartalmazó adatokon érték el a legjobb eredményt. Ez arra utal, hogy a TF-IDF és a RAKE a szövegben előforduló számokat a tartalomleírás szempontjából gyakran tévesen lényeges elemnek tekintették. Ezek eltávolításával a kulcsszavazók számára dominánsabbá váltak a szöveg lényegét jellemzően megragadó főnevek, igék és melléknevek, ami az eredmények javulásához vezetett. A Newsroom korpusz összefoglalóinak generálására a 4.6. táblázat eredményei alapján a két algoritmus közül a RAKE alkalmasabb.

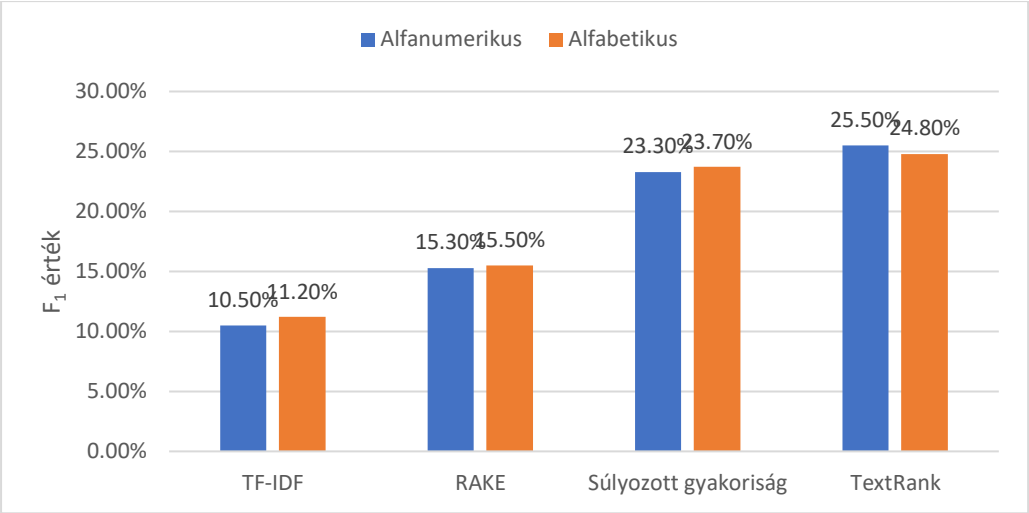
Módszer	**Paraméter**	**ROUGE-1 F_1 érték (alfanumerikus)**	**ROUGE-1 F_1 érték (alfabetikus)**
Súlyozott gyakoriság	2 mondat	23,3 %	23,6 %
	3 mondat	23,3 %	23,7 %
	5 mondat	22,5 %	22,9 %
TextRank	szöveg mondatainak 1 %-a	23,7 %	22,3 %
	szöveg mondatainak 2 %-a	25,5 %	24,8 %
	szöveg mondatainak 3 %-a	24,4 %	23,8 %

4.7. táblázat: kulcsmondat alapú algoritmusok által a Newsroom adathalmazon elért eredmények

A rendszer által generált kivonatok második típusának, a kulcsmondat alapú kivonatoknak a kiértékelését szintén a Newsroom 2496 elemű szöveg-összefoglaló párokból álló részhalmazán végeztem, amely eredményeit a 4.7. táblázat ismerteti. A súlyozott gyakoriság elvén működő kulcsmondat kiválasztási módszernél 2, 3 vagy 5 mondatból generálom a kivonatot, mert a korpusz összefoglalói átlagosan 2 mondatból állnak. Ezzel szemben a TextRank algoritmusnál a forrás szöveg mondatainak 1, 2 vagy 3 százalékát választom ki a kivonathoz, mert az összefoglalók a forrás dokumentum mondatainak átlagosan 1 %-ból épülnek fel. A kulcsmondat alapú módszereknél az eredmények vizsgálatakor nem állapítható meg egyértelműen, hogy az alfanumerikus

vagy alfabetikus megközelítést érdemes inkább használni. A súlyozott gyakoriság esetén a csak alfabetikus karaktereket, valamint szóközt és mondatvégi írásjeleket tartalmazó adatok vezetnek jobb eredményre. Ez a kulcsszó alapú kivonatgeneráló módszerekhez hasonló viselkedést mutat, mert a kulcsmondatok kiválasztásához a kulcsszavazó algoritmusokkal megegyező módon a szógyakoriságokat vizsgálja. Ezzel szemben a TextRank akkor képes magasabb jósági érték elérésére, ha a szövegből a számok nem kerülnek eltávolításra. A számok eltávolítása ugyanis információveszteséghez vezet, mivel esetenként a tartalom leírása szempontjából lényeges szerepet tölthetnek be. A súlyozott gyakoriságnál azért fontos a számok eltávolítása, mert a szavakhoz gyakoriság alapján rendel pontszámot, majd a mondatok fontosságát az őket alkotó szavak fontosság értékeinek összegével jellemzi. Emiatt a hosszabb mondatokat a szöveg szempontjából relevánsabbnak tekinti, így az elhanyagolható számok megőrzése a tartalom eltorzításához vezethet. A TextRank ellenben képes a szöveg egészére vonatkozó összefüggések vizsgálata miatt ezek megfelelő differenciálására.

A két módszer közül a legjobb eredményt a szöveg mondatainak 2 %-át felhasználó TextRank érte el 25,5 %-os F_1 értékkel. A korábban említett több információ biztosítása mellett ennek másik oka a Newsroom adathalmaz szövegeinek és összefoglalóinak változó hosszúsága, ami a kivonatozó rendszerek egyik legnagyobb kihívását jelenti. A súlyozott gyakoriság minden szövegből meghatározott számú mondatot választ ki, míg a TextRank képes az összefoglalóból kiválasztott mondatok számát a forrás szöveg relatív hosszához igazítani. Ennek köszönhetően a korpusz referencia kivonatainak terjedelméhez jobban közelítő kivonatot generál.

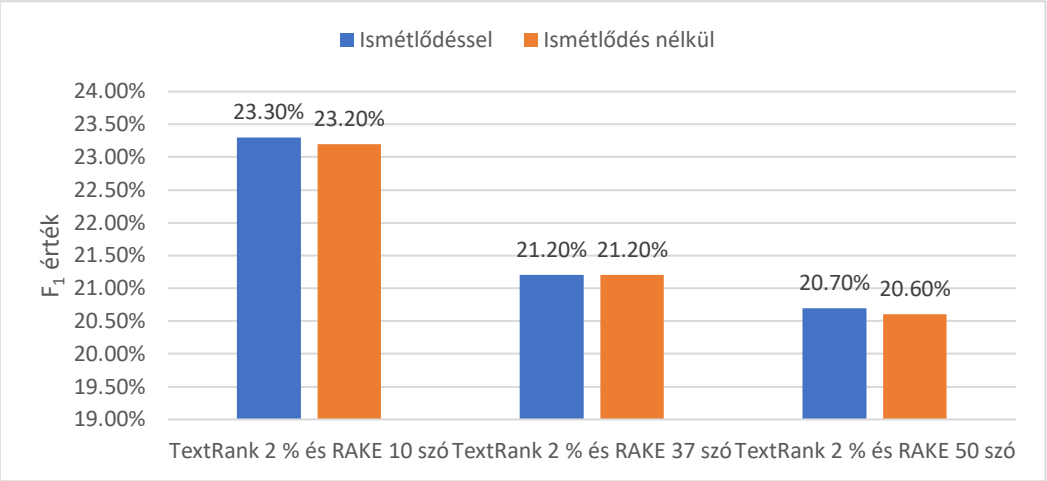


4.3. diagram: Kulcsszó és kulcsmondat alapú módszerek legjobb eredményeinek összevetése

A 4.3. diagram szerint a több mondatból álló kivonatok generálása esetén a két kivonat típus közül a kulcsmondat alapú megközelítések érték el a legjobb eredményt. A kulcsszó alapú módszerek ugyanis a szöveg különböző részeiből, bekezdéseiből válogatják össze a kivonat esetenként egymástól független elemeit. Emiatt előfordulhat, hogy a tartalom leírása szempontjából lényeges szövegrészekből mindössze egy-egy szót ragadnak ki. Ezzel szemben a kulcsmondat alapú kivonatolásnál a forrás szöveg komplett mondatait átemeli az összefoglalóba, ezáltal a lényeges részekből jobb mintavételezést biztosít. A kulcsmondat alapú módszerek dominanciájának másik oka a Newsroom adathalmaz szövegeinek természetéből fakad. A kulcsmondatok kiválasztásával létrejött kivonatoknál ahogy azt már említettem, a korpusz forrás szövegeinek és összefoglalóinak hossza változó, ami megnehezíti a rendszer számára az optimális méretű kivonat előállítását. A TF-IDF és a RAKE egyaránt csak fix szóhosszúsággal rendelkező összefoglalók generálására képes, ami változó hosszúságú szövegek esetén a tartalom nagymértékű lefedésére nem alkalmas. Ezzel szemben a két kulcsmondat alapú módszernél mindössze az átemelendő egységek (mondatok) száma limitált, egy-egy mondat szóhossza azonban változó lehet, ezért könnyebben adaptálódnak a korpusz változó hosszúságú összefoglalóihoz.

A rendszer harmadik típusú kivonatait a kulcsszó és kulcsmondat kinyerő módszerek kombinálásával létrejött összefoglalók alkotják. Ezek vizsgálatakor a korábbi két kivonat típus kiértékelése során szerzett tapasztalatokat használtam fel. Egyrészt a 4.3. diagram alapján a kombinált módszerek vizsgálatát az alfabetikus megközelítés mentén vizsgáltam, mert a TextRank kivételével a többi módszer ezen érte el a legjobb

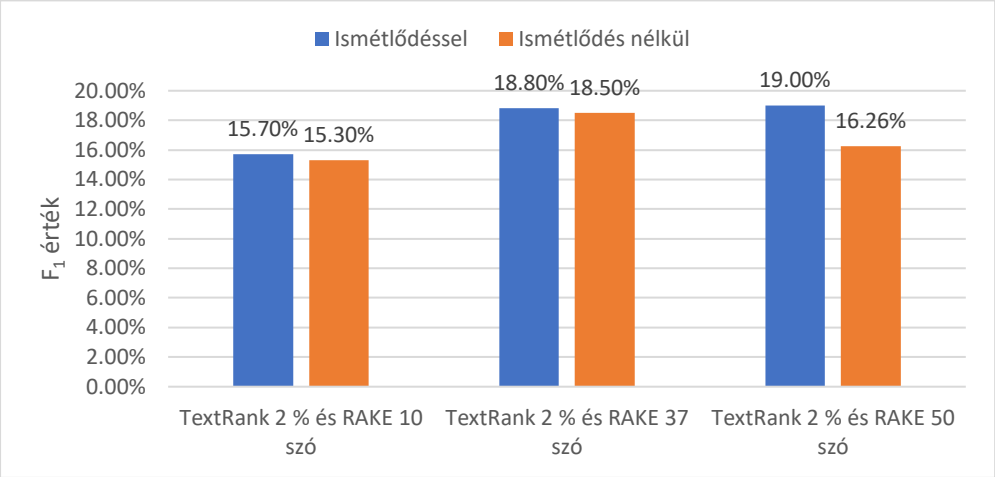
eredményt. Másrészt a kombinált kivonatok generálásához a kulcsszavazó, valamint kulcsmondatok kinyerésére szolgáló algoritmusok közül a saját kategóriájukban legjobb eredményt elérő RAKE-et és TextRanket használtam fel. Utóbbi esetén mindössze a legmagasabb jósági értékkel rendelkező, a forrás szöveg mondatainak 2 %-át kiválasztó konfigurációt vizsgáltam a továbbiakban.



4.4. diagram: Kulcsszó és kulcsmondat alapú módszerek uniójával létrejött kivonatok eredményei

A 4.4. diagram a RAKE és TextRank által kinyert elemek uniójával létrejött kivonatok jósági értékeit tartalmazza. Az unió műveletnél két esetet vizsgáltam: a klasszikus unió fogalmát (ismétlődés nélkül), ami a RAKE és TextRank ismétlődő szavait egyszer tartalmazza, valamint a szavak minden példányát megőrző ismétlődéses uniót. Az eredmények azt mutatják, hogy a rendszer az ismétlődő szavak minden példányának megőrzése esetén generálja a legjobb összefoglalókat. Az ismétlődések eltávolítása ugyanis információvesztéshez vezet, mert előfordulhat, hogy a referencia kivonatok egy adott szót többször is tartalmaznak. A 4.2.3-as fejezetben említett módon az unió művelet motivációja, hogy nagyobb tartalomlefedést biztosítson a két módszer találati listájának kombinálásával. Ezalapján azt várnánk, hogy minél nagyobb a két algoritmus által kinyert elemek halmaza, annál magasabb jósági értéket érnek el. Ezzel szemben a diagram azt mutatja, hogy a TextRank a legkisebb, mindössze 10 kifejezést kinyerő RAKE-kel használható a legjobban. A rendszer viselkedésében megfigyelhető másik ellentmondás, hogy a kulcsszó és kulcsmondat kiválasztó algoritmusok együtt rosszabb eredményt érnek el, mint a TextRank önmagában. A két ellentmondás a ROUGE-1 F_1 érték számítására vezethető vissza. A kivonatok jóságának mérésére a ROUGE-1 által kalkulált másik két metrika a pontosság és a fedés. A pontosság a generált kivonat helyesen prediktált szavainak számát mutatja meg, ami annál jobb, minél kevesebb felesleges szót

használ fel a rendszer a kivonathoz, míg a fedés mérőszám értéke a generált kivonat hosszától független. Az F_1 érték pedig a pontosság és a fedés harmonikus közepe, ezért a pontosság függvényében változik. Az unió műveletnél ez a legjobb és legrosszabb eredményt elérő kombinációk (a mondatok 2 %-ából és 10 vagy 50 kulcsszóból előállított kivonat) esetén a pontosság a felhasznált kulcsszavak számának növelésével 16,6 %-ról 13,5 %-ra csökken, míg a fedés 56 %-ról 64 %-ra javul.



4.5. diagram: Kulcsszó és kulcsmondat alapú módszerek metszetével létrejött kivonatok eredményei

A 4.5. diagram a TextRank és RAKE találati listájának metszetéből generált kivonatokat hasonlítja össze. Eszerint a metszet művelet az unióval ellentétesen viselkedik, ugyanis annál jobb kivonatot eredményez, minél tágabb a két módszer által kinyert elemek halmaza. Ennek oka, hogy a metszet csak azokat a szavakat használja fel a kivonathoz, amelyek mindkét találati listában előfordulnak. A kivonatok közös szavainak száma, ezért a felhasznált kulcsszavak számával megegyező módon viselkedik: a mondatok 2 %-ából és 10 vagy 50 kulcsszóból előállított kivonatnál a pontosság 23,4 %-ról 20 %-ra csökken, viszont a fedés 14,7 %-ról 24,9 %-ra javul. A metszet és az unió művelet viselkedésében továbbá két hasonlóság figyelhető meg. Egyrészt a RAKE és TextRank módszerek közös szavaiból generált kivonatok a közös szavak összes előfordulásának (ismétlődéses metszet) megtartása mentén éri el a legjobb eredményt. Másrészt a metszet az unió művelethez hasonlóan a két módszer kombinálásával rosszabb eredményt ér el, mint a TextRank algoritmus egyedül.

A vizsgált módszerek közül tehát a legjobb eredményt a Newsroom adathalmazon a várakozásokkal ellentétben az alfanumerikus adatokon futtatott, tisztán TextRank alapú kulcsmondatok kinyerésével előállított kivonatok érték el.

5 Összefoglalás

A dolgozatom fő célkitűzése egy absztrakt és egy kinyerés alapú összefoglaló generáló rendszer elkészítése volt. Az absztraktok generálását a rekurrens neurális hálókon alapuló sequence-to-sequence architektúrával valósítottam meg. A kiértékelés során azt tapasztaltam, hogy a rendszer erős absztrakt tulajdonsággal rendelkezik, azaz a neurális hálóknak köszönhetően az összefoglalóhoz a forrás dokumentum szókészletén kívül más szavakat is felhasznál. Az így létrejött absztraktok szavainak egy részét helyesen predikálja a rendszer, ezek helyes sorrendjének meghatározása azonban már komoly nehézséget jelent számára. A modellnek ehhez egy olyan kiterjedt tanító adathalmazra lenne szüksége, amely minél több ilyen kombinációt tartalmaz. A természetes nyelvű szövegek generálásánál ugyanis a nehézséget az jelenti, hogy ugyanaz a tartalom többféle módon, nyelvtani szerkezettel is leírható, ami megnehezíti a modell számára a helyes szórend elsajátítását. A hosszú tanítási idő miatt a modell tanítása túlságosan kevés adaton történt, ami a szórend problémán kívül a generálásnál felhasználható szavak korlátozott halmazát eredményezi. A tanító adatok bővítésével és egy külső, publikus szótár beépítésével a modell szöveggeneráló képessége jelentősen javulhatna. Az általam készített absztrakt generáló rendszer az összefoglalók jósága (1,2 %-os ROUGE-1 F_1) tekintetében lényegesen elmarad a hasonló megoldásokhoz képest. Azok a modellek ugyanis több hetes tanítással, paraméter-optimalizációval, valamint az adatok optimális méretű kötegekbe szervezésével érik el a jobb eredményt. Ez olyan erőforrásokat igényel, amelyek a dolgozatom elkészítése során nem álltak rendelkezésemre.

Az általam készített két összefoglaló generáló rendszer közül, ezért a jobb eredményt a tanítást nem igénylő, kevesebb erőforrást felhasználó kinyerés alapú megközelítéssel értem el. Ennek jóságát egy cím- és egy több mondatos kivonat generálási feladatban is megvizsgáltam. A címgenerálási feladatnál tisztán kulcsszó alapú kivonatokat generáltam, amelynél a kiértékeléshez használt DUC 2003 adathalmaz természete miatt a legjobb eredményt a több szóból álló kifejezések kigyűjtésére alkalmas RAKE érte el 23,19 %-os ROUGE-1 F_1 értékkel. A több mondatos összefoglalók előállításánál kulcsszó, kulcsmondat, valamint kulcsszó és kulcsmondat alapú kivonatokat generáltam. Azt tapasztaltam, hogy a gyakoriság alapú algoritmusok alfabetikus adatokon, míg a gráf alapúak alfanumerikus adatokon használhatóak inkább.

A Newsroom adathalmazon a kulcsmondat alapú módszerek érték el a legjobb eredményt. Ezek jobban alkalmazkodnak a korpuszt jellemző változó szöveghosszúsághoz, valamint jobban lefedik a dokumentum lényeges részeit. Közülük is a legmagasabb 25,5 %-os ROUGE-1 F_1 értéket a szöveg lokális és globális információinak kinyerésére is alkalmas, gráf alapú TextRank algoritmussal értem el. Ez a kivonathoz felhasznált kevesebb felesleges szó miatt önmagában jobb eredményre vezet, mint kulcsszó alapú módszerekkel kombinálva. A kinyerés alapú összefoglaló generáló rendszerem jóságának növelése érdekében a szöveg további adatelőkészítését (főnév, melléknév, ige szófajú szavak megtartása), a kivonathoz felhasznált kulcsszavak és kulcsmondatok számának szélesebb spektrumát, illetve a szövegelemek kinyerésére szolgáló újabb algoritmusok vizsgálatát tartanám célszerűnek.

Köszönetnyilvánítás

A TDK dolgozatban ismertetett eredmények a Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai Kar Balatonfüredi Hallgatói Kutatócsoport szakmai közössége keretében jöttek létre a régió gazdasági fejlődésének elősegítése érdekében. Az eredmények létrehozása során figyelembe vettük a balatonfüredi központú Rendszertudományi Innovációs Klaszter által megfogalmazott célkitűzéseket, valamint a párhuzamosan megvalósuló EFOP 4.2.1-16-2017-00021 pályázat támogatásával elnyert „BME Balatonfüredi Tudáscentrum" térségfejlesztési terveit.

A kutatás az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg (EFOP-3.6.2-16-2017-00013, Innovatív Informatikai és Infokommunikációs Megoldásokat Megalapozó Tematikus Kutatási Együttműködések).

Irodalomjegyzék

[1] Tian Shi, Yaser Keneshloo, Naren Ramakrishnan, Chandan K. Reddy. 2018. Neural Abstractive Text Summarization with Sequence-to-Sequence Models, arXiv preprint arXiv:1812.02303.

[2] Ferreira, R., de Souza Cabral, L., Lins, R. D., e Silva, G. P., Freitas, F., Cavalcanti, G. D., ... & Favaro, L. (2013). Assessing sentence scoring techniques for extractive text summarization. Expert systems with applications, 40(14), 5755-5764.

[3] Rush, A. M., Chopra, S., & Weston, J. (2015). A neural attention model for abstractive sentence summarization. arXiv preprint arXiv:1509.00685

[4] Beliga S. (2014). Keyword extraction: a review of methods and approaches.

[5] Stack Abuse: Text Summarization with NLTK in Python, https://stackabuse.com/text-summarization-with-nltk-in-python/ (2019. okt. 22.)

[6] Qaiser, Shahzad & Ali, Ramsha. (2018). Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents. International Journal of Computer Applications. 181. 10.5120/ijca2018917395.

[7] Rose, Stuart & Engel, Dave & Cramer, Nick & Cowley, Wendy. (2010). Automatic Keyword Extraction from Individual Documents. 10.1002/9780470689646.ch1.

[8] Michalcea, Rada & Tarau, Paul. (2004). TextRank: Bringing Order into Text.

[9] MonkeyLearn: Keyword Extraction A Comprehensive Guide to Extracting Keywords from Text, https://monkeylearn.com/keyword-extraction/ (2019. okt. 18.)

[10] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In Advances in Neural Information Processing Systems, pp. 3104–3112.

[11] Adventures in Machine Learning: Keras LSTM tutorial – How to easily build a powerful deep learning language model, http://adventuresinmachinelearning.com/keras-lstm-tutorial/ (2019. okt. 15.)

[12] A.I. Wiki: A Beginner's Guide to LSTMs and Recurrent Neural Networks, https://skymind.ai/wiki/lstm (2019. okt. 15.)

[13] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. Neural Computation, 9(8):1735–1780.

[14] Hacker Noon: Understanding architecture of LSTM cell from scratch with code, https://hackernoon.com/understanding-architecture-of-lstm-cell-from-scratch-with-code-8da40f0b71f4 (2019. okt. 15.)

[15] Junyoung Chung, C ¸aglar Gu ̈lc ¸ehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. CoRR, abs/1412.3555.

[16] Towards Data Science: Illustrated Guide to LSTM's and GRU's: A step by step explanation, https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21 (2019. okt. 15.)

[17] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In Text Summarization Branches Out: Proceedings of the ACL-04 Workshop, pages 74–81.

[18] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. CoRR, abs/1409.0473.

[19] CN Yah: Attention Variants, http://cnyah.com/2017/08/01/attention-variants/ (2019.okt.14.)

[20] Wikipedia: Beam Search, https://en.wikipedia.org/wiki/Beam_search#cite_note-2 (2019. okt. 15.)

[21] Markus Freitag, Yaser Al-Onaizan. 2017. Beam Search Strategies for Neural Machine Translation, arXiv preprint arXiv:1702.01806.

[22] Medium: Dropout in (Deep) Machine learning, https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5 (2019. okt. 15.)

[23] Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In EMNLP 2015, pages 1412–1421.

[24] mc.ai: Seq2seq model in Tensorflow, http://mc.ai/seq2seq-model-in-tensorflow/ (2019. okt. 15.)

[25] Szűcs G., Huszti D.: Seq2seq Deep Learning Method for Summary Generation by LSTM with Tow-way Encoder and Beam Search Decoder, IEEE 17th International Symposium on Intelligent Systems and Informatics Proceedings, SISY 2019, Subotica, Serbia, September 12-14, 2019, pp. 221-225.

[26] M. Grusky, M. Naaman, and Y. Artzi, "Newsroom: A dataset of 1.3 million summaries with diverse extractive strategies," in Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), vol. 1, 2018, pp. 708–719.

[27] freeCodeCamp: How to process textual data using TF-IDF in Python, https://www.freecodecamp.org/news/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3/ (2019. okt. 15.)

[28] Sergey Brin,Larry Page. 1998. The PageRank Citation Ranking: Bringing Order to the Web.

[29] Medium: Text Summarisation with Gensim (TextRank Algorithm) https://medium.com/@shivangisareen/text-summarisation-with-gensim-textrank-46bbb3401289 (2019. okt. 15.)