ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	**Turóczi Dávid**
2021	Hallgató törzskönyvi száma:	**T/006420/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Turóczi Dávid**
Törzskönyvi száma: T/006420/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Jogszakmai anyagok osztályozása mély tanuló algoritmusok használatával
Legal documents classification with deep learning technique

Intézményi konzulens: Dr. habil. Szénási Sándor
Külső konzulens:

Beadási határidő: 2021. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
Szoftvertervezés és –fejlesztés specializáció

Tartalomjegyzék

1. Bevezetés

Nagyon sok feladat megoldásánál alkalmaznak napjainkban gépi tanuláson alapuló megoldásokat, mivel az utóbbi évtizedek nagy technológiai fejlődése lehetővé tette, hogy a már 1950-es évektől kezdve kitalált és fejlesztett technikákat ténylegesen a gyakorlatban is nagy hatékonysággal lehessen alkalmazni. Így a 2010-es évektől kezdve rengeteg eddig nehezen megoldható problémára láttak napvilágot gépi tanulás segítségével készített megoldások. Ahhoz, hogy ilyen nagy ütemben fejlődni tudjanak ezek a módszerek két dologra volt szükség: korlátlan erőforrásra és nagy mennyiségű adatra, amelyekben fel lehet ismerni valamilyen szabályosságot. [1]

Ahogy ezek a feltételek mind rendelkezésre álltak szinte minden területen hatalmas áttöréseket értek el ilyenek a hangfeldolgozás, képfeldolgozás, szövegelemzés és sok más hasonló terület. Az orvostudományban bizonyos betegségek felderítésére, előrejelzésére próbálják tanítani ezeket a rendszereket egyre nagyobb sikerrel. Megjelentek az egyre nagyobb pontossággal működő arcfelismerő programok, illetve az önvezető autók is egyre jobban működnek és ezekben mindenhol nagy szerepet játszik a gépi tanulás.

A sok terület közül az NLP (Natural Language Processing) azaz a természetes nyelvi feldolgozás hatalmas fejlődése teszi lehetővé, hogy nagy mennyiségű szöveget is képesek legyünk egyszerre lefordítani szinte a világ pármely nyelvére másodpercek alatt. Az NLP lehetővé teszi a számítógépeknek, hogy emberekkel emberi nyelven legyenek képesek kommunikálni, illetve azt is megértsék egyre nagyobb hatékonysággal, amit az emberek írnak vagy épp mondanak. Ezek a képességek teszik lehetővé, hogy különböző írott szövegeket is a számítógép számára érthető formátumra alakítsunk át és ennek segítségével már egy modellt betanítva van lehetősége megtalálni a szövegek, dokumentumok közötti összefüggéseket. [2]

Kutatásomban dokumentumok osztályozásához szükséges technikákat vizsgálom hatékonyság és szempontjából. Majd megvizsgálom a már létező ezen a területen készült kutatásokat és ezek alapján megtervezek majd implementálok egy magyar nyelvű Európai Uniós dokumentumokat osztályozó programot. Végül ennek az osztályozónak a paramétereit változtatva próbálom meg optimalizálni, azaz a lehető legnagyobb pontosságot elérni.

2. Hasonló kutatás

Mielőtt nekikezdtem a projektemnek, megvizsgáltam milyen hasonló kutatások vannak ebben a témakörben. Találtam egy olyan kutatást, ami bár nem csak ezzel a témakörrel foglalkozik, de fő része a természetes nyelvi feldolgozás és amiben egyezik az én kutatásommal, hogy abban is felhasználták tesztelésre az EUR-LEX által nyújtott adatokat.

A kutatás címe: „*A New Hybrid Based on Long Short-Term Memory Network with Spotted Hyena Optimization Algorithm for Multi-Label Text Classification" [30]*

A kutatás az enyémhez hasonlóan több címkés osztályozás, annyi különbséggel, hogy ebben kifejezetten az LSTM réteg használatára, illetve a Spotted Hynea optimalizáló algoritmusra összpontosítanak. „*In the LSTM network, the Skip-gram method is used to embed words into the vector space." [30]* A kutatásban a Skip-gram algoritmus segítségével ágyazták be a szavakat a vektor térbe. A kutatásban az összes szöveg száma 19348 darab volt ezekhez 3993 címke kapcsolódott. Átlagos egy dokumentumhoz 5.32 különböző címke kapcsolódik. Az általuk elért eredmények közül az egyik legjobb pontosság a 46,43% volt. [30]

Ebben a kutatásban is hasonlóan az enyémhez, egy dokumentumot több osztályba is be lehet sorolni, viszont én a modellemet csak kifejezetten egy címkére vizsgálom majd meg, ezzel a kutatással ellentétben. Ennek az az oka, hogy egy hierarchikus osztályozási rendszerről van szó, amely 6 szintű is lehet és gyakran a különböző címkék között alsó szinten van eltérés én pedig csak a felső 20 osztályra koncentrálok. Ezért önmagában, majd az én eredményeimmel összehasonlítani nem feltétlen lehetséges megfelelő mértékben az eltérő feladat meghatározás miatt, viszont jó alapot nyújt az elérendő célhoz.

3. A szakdolgozatban megvalósítandó rendszer

3.1. A probléma felvetése

Manapság a technológia hatalmas fejlődésének következtében egyre nagyobb mennyiségű adat áll a rendelkezésünkre, akár képi akár szöveges adatokra gondolunk. Mind a két adattípusra jellemző, hogy a hatalmas mennyisége miatt nehézzé válik a felcímkézésük. Ahhoz, hogy valamilyen adatról el tudjuk dönteni, hogy azok besorolhatóak-e egy adott osztályba szükségünk van alapismeretekre, amely lefedi az adott témakört. Ez különösen dokumentumok esetében nehézkes amikor nem feltétlen elég ránézni, mint egy képre, hanem meg kell értenünk annak tartalmát. Tovább nehezíti ezt a feladatot, ha valamilyen specifikus nyelvezetet vagy tudást igényel a megértésük.

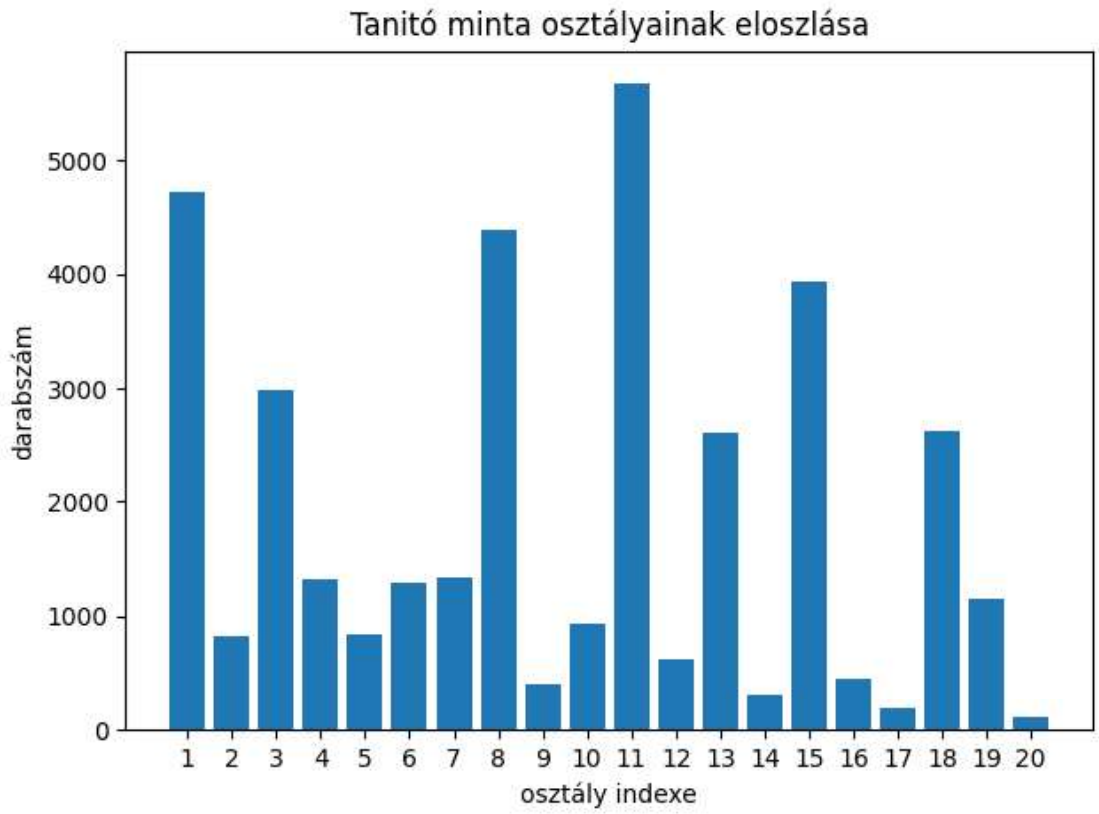
A világon minden állam, szervezet és cég hatalmas mennyiségű jogi dokumentumot készít el. Ezeket rengeteg szempont szerint van lehetőségünk csoportosítani. Ezeknek a csoportosításoknak nagy része adott, amikor elkészül egy dokumentum, de ez nem minden esetben igaz. Amennyiben nincsennek felcímkézve a dokumentumok akkor rengeteg munkát igényelne az, hogy emberi munkaerővel utólagosan csoportokba rendezzünk azokat.

Ezért a folyamatosan fejlődő gépi tanulás segítségével lehetne készíteni egy olyan megoldást, amely az emberi munkaerőt nélkülözve másodpercek alatt képes lenne eldönteni, hogy milyen csoportba is tartozik a dokumentum. Ahhoz, hogy ezt sikerüljön megvalósítanom nagy mennyiségű adatnak kell rendelkezésemre állnia, szerencsére az európai unió minden jogi dokumentuma nyilvánosan elérhető és felhasználható. Így rendelkezésemre áll megfelelő mennyiségű adat.

3.2. A probléma fontossága

A dokumentumok hatékony feldolgozása és osztályozása napjainkban egy elég nagy problémát jelent. Ez a probléma nagyobb szervezeteknél és nagyvállalatoknál egyaránt előfordul. Nem csak az említett jogi területen, hanem sok más olyan területen, ahol nagy mennyiségű szöveges adatot kell feldolgozni. Ebben az esetben az egyszerű fizikai módszerek nem tudják tartani a lépést, azzal a hatalmas mennyiségű adathalmazzal, amely nap mint nap keletkezik. Ez a folyamat emberekkel elvégeztetve magas szakértelmet, és nagyon hosszú idő befektetést igényel. Ezért mindenképpen érdemes egy olyan rendszert létrehozni, amely ezt az időt és költséget a töredékére csökkenti.

4. A tanító adathalmaz



1. ábra: Tanító minta eloszlása

Az indexekhez tartozó nevek a 6.1-es pontban találhatók

A rendelkezésemre álló adatokat az eurlex.com-ról szereztem be, ahol nyilvánosan elérhető minden Európai Unió által kiadott jogi dokumentum. Itt rákerestem minden olyan magyar nyelven is elérhető dokumentumra, amelynek van fejezetkódja. Ezeknek a dokumentumoknak lekértem az adatait, azaz a celex[1] számot, amellyel be lehet azonosítani a dokumentumokat, illetve a fejezetkódot, ami az osztálycímke. Egy egységes link alapján a celex számot helyettesítve elérhetővé válik magának a dokumentumnak a szövege. Legeneráltam minden dokumentumhoz egy linket (https://eur-lex.europa.eu/legal-content/HU/TXT/HTML/?uri=CELEX:), amely felhasználásával le tudtam tölteni mindet html formátumban. Igy benne maradtak a html tagek, amelyeket ki kellett vonnom valahogy a dokumentumokból, ezekre jó megoldást jelentett a html2text könyvtár használata. Igy kaptam meg összesen 32000 dokumentumot sima txt formátumban. A probléma az adathalmazzal, mint a fenti ábrán látszik az, hogy nem

[1] A celex szám a dokumentumok egyedi azonosítója az eurlex.com-on, ami minden nyelven azonos.

kiegyenlített az adatok eloszlása, és erre a problémára megoldást kell majd találnom, ezt majd a következő pontban részletezem. [14]

5. Dokumentumok előfeldolgozásának lépései és lehetőségei

Dokumentumok feldolgozása során több mindent szem előtt kell tartanunk.
Az adatok előfeldolgozása és tisztasága az egyik legfontosabb befolyásoló tényező abban, hogy milyen minőségű hálózatot tudunk kialakítani. Nagyban befolyásolni tudják a végeredmény kimenetelét. Ezért nagy hangsúlyt kell arra fektetni, hogy a lehető legtöbb, de ezek mellett minőségében is megfelelő adatot gyűjtsünk össze.

5.1. Nincs elegendő adat

Olyan problémák megoldásánál, ahol nem áll rendelkezésre megfelelő mennyiségű adat, nem célszerű gépi tanulást alkalmazni. Egy tanításnál hatalmas mennyiségű adatra van szükség ahhoz mindenképpen, hogy jó eredményt érjünk el. Bár igaz, hogy kisebb komplexitású feladatok esetében jóval kisebb adatmennyiség is elegendő lehet. A komplex feladatok megoldásánál amikor sok változó együttesen befolyásolja a kimenetet ilyen például a képfelismerés jóval nagyobb mennyiségű adatra van szükség. [5]

Másik probléma, ami előfordulhat, hogy van elegendő rendelkezésre álló adat, viszont azok eloszlása nem megfelelő, így a gyakorlatban lényeges kisebb osztályok esetlegesen nem kapnak elég teret, hiszen a tanítómintában lényegesen alacsonyabb volt a számuk. [5]

Az én esetemben az Európai Uniós jogban összesen több mint 1 millió publikus adat áll bárki rendelkezésére. Ezek közül én a magyar nyelvre lefordítottakra koncentrálok, és mivel én felügyelt tanítást fogok alkalmazni (későbbiekben kitérek erre, hogy miért), csak azok a dokumentumok jöhetnek szóba, amelyek rendelkeznek címkével. Ilyen jellegű dokumentumból összesen 32000 magyar nyelvű érhető el. Ez mennyiségében megfelelő lehet a tanításhoz, de csak modell megléte után lehet levonni pontos következtetéseket.

5.2. A képzési adatok nem arányosan fedik le a problémateret

Ahhoz, hogy a modell általánosan jól működjön, tehát jelen esetben képes legyen minden osztályt nagyjából egyforma minőségben megjósolni, elengedhetetlen, hogy minden osztályban közel azonos mennyiségű adat álljon a rendelkezésünkre. Ebben az esetben, ha ez nem teljesül a következő probléma léphet fel: A modell hajlamos bizonyos osztályokat főként, amelyekből több volt a tanító mintában, előnyben részesíteni, ezáltal kiszorulnak bizonyos osztályok.

Ezen a területen már merülhet fel probléma az adathalmazommal, hiszen vannak olyan osztályok, amelyek jóval gyakrabban fordulnak elő, mint mások. Sajnos jelen esetben nincs arra lehetőség, hogy ezeket egyszerű plusz adatok begyűjtésével pótoljam.

Bizonyos témakörökben több jogi dokumentum készül, mint másokban, ezért talán nem is lehetne tökéletes mintát begyűjteni. Négy megoldás van erre a problémára:

- A kisebbségben lévő osztályokhoz tartozó elemekhez újakat kell hozzáadni
- Oversampling [2]azaz túlmintavételezés a kisebbségi osztályokból
- Undersampling [3]azaz alul mintavételezés a többségi osztályokból
- Módosíthatjuk a költségfüggvényt, hogy a kisebbségben lévő osztályok téves osztályozása nagyobb büntetéssel járjon, mint a többségi osztályoké [7]

5.2.1. Undersampling

Ebben az esetben úgy próbáljuk meg orvosolni a tanítóminta osztályai közötti különbségeket, hogy eltávolítunk elemeket azokból az osztályokból, amelyek jóval több elemet tartalmaznak. Természetesen ennek megvan az a nagyon nagy hátránya, hogy ezáltal drasztikusan csökkenhet a tanító mintáink száma. Ezt tehát csak akkor tudjuk hatékonyan alkalmazni, ha a többségi osztályokból történő elem eltávolítások után is megfelelő mennyiségű tanító minta áll a rendelkezésünkre. [6]

5.2.2. Oversampling

Ebben az esetben a kisebbségi osztályokhoz próbálunk meg szintetikus módon újabb osztályokat generálni. Ennek egy lehetséges módszere a SMOTE könyvtár használata, amely képes a rendelkezésünkre álló adatok alapján új szintetikus adatokat létrehozni. Mivel ez a módszer csak numerikus adatokon működik a dokumentumainkat át kell alakítanunk előtte, és mivel egy dokumentum megfelelő leírásához több száz dimenziós vektorra van szükség ez megnehezíti az ilyen módon történő adatgenerálást. [7]

5.2.3. SMOTE

A SMOTE egy olyan könyvtár, amelynek segítségével szintetikus adatokat tudunk előállítani. Úgy működik, hogy kiválasztjuk azt az adathalmazt, amelyekből szeretnénk még több adatot. Ezek között az adatok között vonalakat húz és ezekre a vonalakra kerülnek majd az új mintapontok. Lényegében ezzel az eljárással korlátlan mennyiségű adatot tudunk létrehozni. Viszont a valódi adatokhoz képest megvan az a hátránya, hogy nem tudhatjuk milyen átfedések vannak az osztályok között, igy ezek a belegenerált adatok átlóghatnak és illeszkedhetnek más osztályokba is. Emellett nagy előnye, hogy nem jár adatvesztéssel ennek a módszernek a használata, ami kifejezetten pozitív egy

[2] Az oversampling a kevés minta esetén több minta szerzése vagy generálása
[3] Az undersampling a túl sok minta esetén kevesebb minta felhasználása

olyan helyzetben amikor az adatok nagy részét el kellene hagynunk, hogy kiegyenlítsük az adatkészletünket. A SMOTE mint a hálózat is numerikus adatokat tud csak befogadni ezáltal még fontosabbá válik a dokumentumok minél nagyobb pontossággal történő átalakítása, hiszen ez nem csak a hálózatot befolyásolhatja, hanem az új szintetikus adatok minőségét is.

A felsorolt technikák közül az undersampling használata nem volna kedvező számomra, hiszen a rendelkezésemre álló adatok között van olyan osztály, amely pár száz elemet tartalmaz, illetve olyan is, ami több ezret. Jelen esetben 20000-nél is több adatot kellene emiatt kidobnom. Ami túlzottan nagy veszteség lenne. Az oversampling önmagában egy jó megoldást jelentene, hiszen adatveszteség nélkül sikerülne kiegyensúlyozni az adatkészletet, de problémát jelenthet, hogy elég komplex adatokról van szó és emellett nagy mennyiségről is. Ezért érdemes lehet mind a két technikát használni, de ennek a pontos mértékét csak a tesztelés után lehet megállapítani, hogy miként reagál rá a modell.

5.3. Adatok minősége

Azok az adatok, amelyek hibákat, kiugró értékeket és egyéb zajt tartalmaznak nagyban ronthatják a teljesítményt. Ezért mielőtt átadnánk a modellünknek az adatokat meg kell tisztítanunk azokat. Ez a tisztítási folyamat nagyban függ az adatok fajtájától. Az én esetemben szöveges adatokról van szó így ezeknek az adatoknak a tisztítását fogom részletezni:

A dokumentumokban lévő felesleges írásjeleket és egyéb jelöléseket el kell távolitani, hiszen ezek nem hordoznak magukban lényeges információt. A nagy betűket is érdemes kis betűkké alakitani, hogy egyként kezelje ezeket a szavakat, hiszen nekem jelen esetben nem hordoz plusz jelentést a nagy betűk használata.

A stop szavak eltávolításával is segíthetjük a modellünket. Ezek olyan szavak, amelyek egy nyelvben nagyon nagy mennyiségben fordulnak elő. Ez annyira nem okoz nagy problémát egy dokumentum osztályozás esetében. Viszont ami már okozhat problémát és a magyar nyelvben elég erőteljesen jelen van az a rengeteg különböző szóalak.

Törekedni kell arra is, hogy a lehető legtöbb zaj el tudjuk távolitani a dokumentumokból. Ilyenek az írásjelek, értelmetlen szavak, számok, egyéb olyan jelölések, amelyek nem egy osztály jellemzésére szolgálnak csak például bekezdéseket jelölnek.

6. Dokumentumok átalakítási technikáinak összehasonlítása

6.1. Miért van szükség a szavak/dokumentumok átalakítására

Ha valamilyen témában szeretnénk keresni mondjuk az interneten alap esetben az arra a témára jellemző szavakkal próbálkoznánk, viszont ez nem fedi le az összes lehetőséget hiszen csak azokat a bejegyzéseket adná vissza, amelyek tartalmazzák az adott kulcsszavakat, nem lehetnek se elírva se pedig más szóval helyettesítve, mivel ekkor már nem találnánk egyezést.
Ezért találták ki a szavak beágyazását, amely a szavak közötti összefüggéseket és hasonlóságokat is figyelembe véve reprezentálja a szavakat. Ugyan ez igaz a dokumentumokra is, ahol már nagyobb egységek is függenek egymástól, és ezen nagyobb egységek, mint például mondatok és bekezdések közötti kapcsolat is információt hordoz. Több lehetséges módszer is van arra, hogy dokumentumokat ábrázoljunk számvektorokként. A következőkben ezeket a lehetséges megoldásokat fogom bemutatni. [3]

6.2. Word2Vec

A Word2Vec megismerése is fontos, mivel ezen a technikán alapszik a Doc2Vec is, amely segítségével már képesek vagyunk nem csak szavak, hanem szó korlát nélküli dokumentumokat is numerikus vektorokká alakítani.

A Word2Vec algoritmus neurális hálózati modellt használ a szó asszociációk megtanulására egy nagy szövegkorpuszból. Vagyis a betanított modell képes felismerni a szinonim szavakat vagy további szavakat javasolni egy részmondathoz. Ez azt jelenti, hogy egy előre meghatározott értelmes mondatrészt, egy megfelelően betanított neurális hálózat képes értelmesen befejezni.

A szóbeágyazás egyik legegyszerűbb módja a One-Hot[4] vektorok alkalmazása, amelyek a rendelkezésünkre álló szavak mennyiségét veszik alapul, ez lehet például 5000 szó, ebben az esetben 5000 elemből álló vektorok keletkeznek, amelyek mindegyike egy adott szót reprezentál. Ezzel alapvetően az a probléma, hogy minél nagyobb a szókincs annál hosszabbak a vektorok, illetve egy ilyen vektor semmilyen jelentést nem hordoz önmagában az egymáshoz jelentésben közel álló szavakról. [26]

[4] Olyan vektorok, amelyek csak 0-ás karaktereket alkalmaznak kivéve egy helyen, ahol 1-es karaktert, így hordoznak egyedi jelentést.

6.3. Doc2Vec

A Doc2Vec a „Document to Vector” összevonásából jön, magyarul „dokumentum vektorrá”, ami magára a működésére utal. A Doc2Vec dokumentumokat alakít át vektorokká.

2013-ban készült el a word2vec, amely képes a szavak kontextusának információit ábrázolni szám vektorok segítségivel. Ez a fajta megközelítés rendkívül nagy népszerűségre tett szert. Ez után jelent meg a doc2vec, amely egy felügyelet nélküli algoritmus és ezt a módszert bővíti ki dokumentumokra, amely azt jelenti, hogy képes egy egész dokumentum tartalmát ábrázolni egy szám vektor segítségével, és abba reprezentálni a dokumentum tartalmát. Az egyik nagy előnye a technikának, hogy szóhossz függetlenül képes feldolgozni a dokumentumokat, ezáltal nincs szükség a szóhossz kiegyenlítésére semleges karakterek használatával.[4]

A Doc2Vec esetében a skip-gramm és a CBOW algoritmusok kerültek kiegészítésre annak érdekében, hogy azokat a dokumentumokra is megfelelően lehessen alkalmazni. Itt minden bekezdés vektor egy adott dokumentumhoz van hozzárendelve és emellett a szóvektorok meg vannak osztva a dokumentumok között. [25]

A Doc2Vec jól használható jogi szövegek esetében. Még akkor is, ha a jogi szövegek átlagos hossza meghaladja a tizezer szót. Az általam talált tanulmány szerint nagyon nagy terjedelmű jogi dokumentumok esetében, ahol az átlagos dokumentum hossz több tízezer szó, ott teljesítmény növekedés érhető el, ha kisebb részekre bontjuk fel a dokumentumokat. Ebben a tanulmányban a Doc2Vec-et használták dokumentum ábrázolásra és magas 98%-os test és validációs pontosságot is sikerült elérniük vele. Ez azt bizonyítja, hogy az én feladatom esetében is jól alkalmazható lehet a Doc2Vec technológia. Ezzel szemben az én adatkészletemben az átlagos dokumentum körülbelül 3000 szóból áll, így nem valószínű, hogy érdemes felbontani azokat. [13]

6.4. BERT

A Google BERT az utóbbi évek egyik legnagyobb áttörése a természetes nyelvi feldolgozást igénylő feladatok körében.

„BERT is a method of pre-training language representations. Pre-training refers to how BERT is first trained on a large source of text, such as Wikipedia. You can then apply the training results to other Natural Language Processing (NLP) tasks, such as question answering and sentiment analysis. With BERT and AI Platform Training, you can train a variety of NLP models in about 30 minutes.” [22]

Vagyis ez a módszer a transzformer architektúrán alapul. Emellett két irányban is meg tudja tanulni a szavak kontextusát. Mellette szól még, hogy hatalmas mennyiségű címkézetlen adatkészletet használtak fel az edzéséhez, igy több BERT is készült.
Idővel rengeteg a BERT-hez hasonló módszeren alapuló technológiai megoldás jelent meg. Ilyen például a XLNet, ami inkább rövidebb szövegekre ad vissza nagyon jó megoldásokat.
Ezen technológiák közül sokban gyakran méretkorlát van az átadható szövegek méretére vonatkozóan. Ezek miatt sokkal nehezebb velük megvalósítani egy ilyen probléma megoldását.
Kifejezetten nagy előnye ezeknek a technológiáknak, hogy hatalmas adatkészleten edzették ki őket, és ezáltal nagyon jól megtanulták a szavak közötti kontextusok ábrázolását, viszont ez egyben a hátránya is az én esetemben.

Az én dokumentumaimban lévő szókincs eléggé specifikus, és emellett magyar nyelvű, tehát mindenképpen szükségem lenne egy olyan transzformerre, amely magyar nyelvű jogi dokumentumok felhasználásával lett kiedzve. Talán ez a legnagyobb hátrány a nagy erőforrás igény mellett, bár itt lényeges megjegyezni, hogy minden technológia, amellyel szövegeket szeretnénk dokumentumként ábrázolni, elég magas erőforrásigénnyel rendelkezik. Minél pontosabb modellt szeretnénk létrehozni, annál inkább több időre, vagy több erőforrásra lesz szükségünk. [16][17]

6.5. Szöveg vektorizáló módszer kiválasztása

A szöveg vektorrá alakításához használandó módszer kiválasztáshoz a következő szempontokat fogom figyelembe venni: a dokumentumok átlagos hossza, a szöveg és a feladat fajtája. Ez a három tényező nagyban tudja befolyásolni, hogy milyen módszert érdemes alkalmazni.

Vannak olyan megoldások, amelyek nem tudnak korlátlan hosszúságú szöveget átalakítani. Ez a probléma merül fel a BERT esetében. Bár a BERT tartalmaz előre betanított transzformereket, amelyek nagyon nagy pontosságot képesek elérni, de számomra ezek csak akkor felelnének meg, ha magyar jogi szövegeken tanították volna, hiszen egy specifikus szókészletről van szó.
A dokumentum osztályozásban kevésbé van szükség a szavak pontos kontextusának a feltérképezésére, szemben egy olyan feladattal, ahol valamilyen szöveget szeretnénk generálni vagy valamilyen érzelmeket felismerni benne. Bár volna lehetőség dokumentumokra is alkalmazni, de ez még nagyobb számítási kapacitást igényelne, vagy még több időt, ami a tesztelést nehezíti. Az pedig mindenképpen nagy hátrány, hogy ezáltal nincs lehetőség arra, hogy megfelelő mennyiségű tesztet futtassunk le, amennyiben nem áll rendelkezésünkre korlátlan erőforrás. Ezek alapján a problémák alapján inkább egy másik technológia használatát tartom célravezetőbbnek. [17]

A Doc2Vec nagy előnye számomra, hogy egyszerűen tudok benne modellt készíteni és ezáltal egy személyre szabott megvalósítást készíthetek belőle, amely jogi adatokkal van kiedzve és ezáltal sokkal hatékonyabban fog működni egy ilyen specifikus szókincsű területen is. Másik nagy előnye, hogy jól használható hosszú dokumentumok feldolgozása esetén is, mivel ezt a technikát kifejezetten dokumentum beágyazások elkészítésére találták ki.

A Doc2Vec kiedzésénél kell valamilyen módszer arra, hogy különböző paraméterekkel rendelkező modelleket tudjak összehasonlítani, és ezek alapján kiválaszthassam a megfelelő paramétereket. Egyik ilyen megoldás a Doc2Vec dokumentációjában elérhető módszer, amelyben a tanitó adathalmaz minden egyes dokumentumára egy új vektorral következtetünk és összehasonlítjuk őket a korábban tanított korpusszal. Ezesetben úgy teszünk, mintha még sose látta volna a modell ezeket az elemeket.
Ezáltal ezek a dokumentumok már nagyon jól illeszkednek a modellünkre, ezért a legjobb esetben azt kellene tapasztalnunk, hogy amikor lekérdezzük a dokumentumhoz leghasonlóbb dokumentumokat, akkor önmagát kapja vissza a legelső helyen. Amennyiben minél többször tapasztaljuk azt, hogy az átadott dokumentumhoz a modell ugyan azt a dokumentumot tartja a leghasonlóbbnak, annál jobb a modell pontossága.
Tehát lénygében azt nézzük, hogy ugyan azt a dokumentumot kapjuk-e vissza.

Emellett tesztelhetjük még olyan dokumentumokkal is, amelyeket még nem látott a modell, és ezeket szemmel tudjuk majd összehasonlítani, hogy mekkora egyezést tapasztalunk. Ezt lényegében úgy érdemes csinálni, hogy kiíratjuk a leghasonlóbb, a közepesen hasonló és a legtávolabb álló dokumentumot a még nem látott dokumentumhoz képest. Ezeket logikusan jelentés alapján megvizsgáljuk és ha megfelelő összehasonlításban kaptuk vissza a dokumentumokat, akkor ez alapján le tudjuk vonni a következtetéseket.

Ebbe számomra még az is nagyon pozitív, hogy el tudom menteni ezeket az adatokat, főleg az automatikus összehasonlítás esetében, és ezeket a későbbiekben fel tudom használni a hálózat optimalizálása érdekében. [15]

7. Csoportosítás osztályai

7.1. Címkék:

'Általános, pénzügyi és intézményi ügyek': 1,
'Vámunió és az áruk szabad mozgása': 2,
'Mezőgazdaság': 3,
'Halászat': 4,
'A munkavállalók szabad mozgása és szociálpolitika': 5,
'Letelepedési jog és a szolgáltatásnyújtás szabadsága': 6,
'Közlekedéspolitika': 7,
'Versenypolitika': 8,
'Adózás': 9,
'Gazdaság- és monetáris politika és a tőke szabad mozgása': 10,
'Külkapcsolatok': 11,
'Energia': 12,
'Iparpolitika és belső piac': 13,
'Regionális politika és a strukturális eszközök összehangolása': 14,
'Környezet, fogyasztók és egészségvédelem': 15,
'Tudomány, tájékoztatás, oktatás és kultúra': 16,
'A vállalkozásokra vonatkozó jogszabályok': 17,
'Közös kül- és biztonságpolitika': 18,
'A szabadságon, a biztonságon és a jog érvényesülésén alapuló térség': 19,
'Polgárok Európája': 20,

7.2. Fejezetkód

A fejezetkód egy hierarchikus osztályozási forma ez azt jelenti, hogy van alap 20 osztály és ezeken belül még több szinttel lejjebb is lehet haladni. Másik lényeges tulajdonsága, hogy egy dokumentumot nem csak egy osztályba lehet besorolni, hiszen nem feltétlen irható le csak egy osztállyal a tartalma. Ezért maximum 3 különböző osztályba lehet besorolni, persze lehetséges, hogy csak 1 osztályba tartozik bele. [11]

8. Rendszerterv és tanítás

8.1. Tanítás folyamata

Az első lehetséges mód a felügyelt dokumentum osztályozás, ami azt jelenti, hogy valamilyen külső mechanizmus jelen esetben emberek már elvégezték egy nagyobb adathalmazon a címkézés, osztályokba rendezést, és ezek alapján az információk alapján próbáljuk meg tanítani a hálózatunkat. [8]

A második lehetőség a felügyelet nélküli dokumentum osztályozás. Ebben az esetben nincs semmilyen előre meghatározott rend, ami szerint már tudnánk, hogy az adott dokumentumok milyen osztályba is lehetnek. Ebben az esetben azt tudjuk csinálni, hogy a modellre bízzuk, hogy ha talál a dokumentum között valamilyen fajta hasonlóságot akkor azokat egybe fogja gyűjteni. [8]

Az én esetemben mindenképp a felügyelt tanítás a megfelelő választás, mivel rendelkezésemre áll elég nagy mennyiségű címkézett adat. Illetve maga a szabályrendszer is adott. Magával a felügyelet nélküli tanítással az a probléma merülhetne fel, hogy például sok dokumentumban szerepelhet az aláíró felek neve, esetleg országok nevei és ezek az információk külön osztályt képezhetnek, ha úgy nézzük, hogy melyik országra vonatkozik egy dokumentum. A probléma ezzel az lenne, hogy számunkra lényeges információt nem szerzünk meg így.

A tanitás folyamata két nagyobb részre bontható, először egy Doc2Vec modellt kell elkészíteni és felparaméterezni, ezután ennek a modellnek a segítségével tudjuk betáplálni a másik modellünkbe az adatokat. Ezért belátható, hogy mind a két modell nagy pontossága nagyban hozzá fog járulni ahhoz, hogy ténylegesen milyen jóslatot fog visszaadni végül a rendszer.

8.2. Keretrendszer kiválasztása

8.2.1. TensorFlow

„TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.” [19]

Vagyis a TensorFlow egy nyílt forráskódú, bárki számára ingyenesen elérhető mély tanulási könyvtár. A Google fejlesztette ki és tartja fenn. Képes akár egyszerre több CPU-

n és GPU-n is futni, és akár mobil operációs rendszereken is futtatható. Általában az iparban használják, illetve nagyobb, komplexebb projekteknél, mivel jobban feladatra szabhatóak a rétegei mint a Keras-nak hiszen az pont erre épül. A szintaxisát nehezebb megtanulni, a Keras-al szemben bonyolultabb, és így a használata és elsajátítása is több időt vesz igénybe, de mégis nagyon nagy népszerűségnek örvend. Nagyon jól dokumentált. Az ipar egyik nagy része támogatja. Nagyon jó vizualizációs könyvtárakkal rendelkezik, tehát ezek segítségével nyomon tudjuk követni a tanitás folyamatát. Ilyen eszköz hozzá a TensorBoard. [9]

8.2.2. Keras

„Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible is key to doing good research." [20]

Vagyis a Keras egy pythonban írt, a Tensorflow-hoz hasonlóan nyílt forráskódú és bárki számára elérhető könyvtár. Szintén a Google-nél fejlesztették ki és a TensorFlow tetején fut. Úgy tervezték meg, hogy könnyen használható legyen, a felhasználóra összpontosít.

„The core data structures of Keras are layers and models." [20]

Könnyen és gyorsan el lehet készíteni benne egy modellt, ezáltal, ha gyorsan szeretnénk egy prototípust készíteni, és tesztelni, akkor ez a legjobb választás. Ezekből adódik, hogy nagyon gyors kísérletezést tesz lehetővé. Mindezek mellet nagyon jól dokumentált is. [9]

A Keras által biztosított technológiák, amelyeket fel lehet használni a modell felépítéséhez:

- Dense: A sűrűn összekötött réteg a legegyszerűbb neurális háló réteg. Maga a réteg úgy néz ki, hogy egy adott neuron bemenetére
  bekötjük az előző réteg összes neuronját. Egy ilyen rétegekből álló neurális háló kiedzése a többi bonyolultabb réteghez
  képest kevesebb időt igényel. A hálózat utolsó rétege alapvetően sűrűn összekötött réteg és itt alakul ki a végső kimenet. Itt
  az utolsó réteg neuronjainak száma megegyezik az osztályzásban résztvevő osztályok számával. [28]
- Dropout: Egyszerűen fogalmazva ez egy olyan réteg, amely egy adott ponton bizonyos valószínűséggel kihagy egy adott neuront a képzés adott szakaszában. Tehát abban a pillanatban az adott neuron értéke nem fog változni, nem vesz részt a képzésben. Ez azért fontos a számunkra, hogy a modellel elkerüljük a

túlillesztést. Ami azt jelenti, hogy a modell a képzési adatokat, bár nagyon magas százalékban felismeri és helyesen osztályozza, viszont ezzel szemben egy még nem látott mintára már sokkal kisebb hatékonysággal működik. Bár a Dropout réteg miatt több korszakra lehet szükség a kiképzés során, viszont ezzel szemben fel is gyorsítja a korszakok futás idejét. [27]

Aktivációs függvények szerepe és a könyvtár által nyújtott lehetőségek:

Alapvetően azért van szükség az aktivációs függvényekre, mivel a bonyolultabb problémák esetében segíti a hálózatot, hogy jobban megtanulhassa a bonyolultabb mintázatokat. A legfontosabb tulajdonsága egy aktivációs függvénynek az, hogy nem linearitást ad a hálózathoz.

- Relu: A modern neurális hálózatokban főként Relu aktivációs függvényt használnak. Ez alapértelmezetten a 0-nál kisebb egyenlő számokra 0 értéket ad vissza, egyéb esetben marad az eredeti értéke.
- Sigmoid: A Sigmoid egy olyan függvény, amely, 0 vagy 1 közötti értékké alakítja a bemenetét. Amennyiben 1-nél nagyobb a bemenet 1 lesz a kimenete, 0-nál kisebb bemenet esetén pedig 0. A kettő közötti értékek esetén pedig marad a beérkezett érték. Felhasználása inkább a múltban volt jellemző.

8.2.3. PyTorch

„An open source machine learning framework that accelerates the path from research prototyping to production deployment.” [21]

Vagyis a PyTorch szintén egy nyílt forráskódú gépi tanulási könyvtár. A facebook AI-t kutató laboratóriuma fejlesztette ki. Rendelkezik python és c++ felülettel is. Sokan használják természetes nyelvi feldolgozásra és számitógépes látással kapcsolatos feladatokra. A PyTorch legfőképpen abban különbözik a többi mély tanulási keretrendszertől, hogy dinamikus számítási grafikonokat használ. Ez lényegében azt jelenti, hogy még a statikus számítási grafikonokat a futási idő előtt definiáljuk, addig a dinamikusakat futásidő közben számítjuk.
A PyTorch ezek mellett könnyen és gyorsan megtanulható köszönhetően az egyszerű szintaxisának. [12]

8.2.4. Választott keretrendszer

A feladatra a korábban felsorolt könyvtárak közül lényegében bármelyik megfelelne. Én a Keras keretrendszert választottam ki, mivel ez a TensorFlow tetején fut ezért lényegében Tensorflow-t használok, viszont magasabb szinten.
Ez leegyszerűsíti a modell elkészítését és gyorsabb tesztelést tesz lehetővé.

Ehhez kapcsolódik, hogy python nyelven fogom elkészíteni a szoftveremet, mivel ez az a nyelv, amelyet az egyik legszélesebb körben alkalmaznak gépi tanulás témakörében, illetve ezen a nyelven használható az összes általam kiválasztott könyvtár és segédeszköz, amelyek segítségével elkészíthetem a projektem.

8.3. TensorBoard (modell monitorozásához)

A TensorBoard egy olyan nyílt forráskódú szoftver, amely segítségével vizualizálni tudjuk a modellünket, hibakeresésre van benn lehetőség, és segít az optimalizálásban. Segítségével nyomon tudunk követni olyan fontos mutatókat, mint a pontosság és a veszteség, a modell grafikájának megjelenítése.
Lehetőség van benne nyomon követni az idő múlásával a súlyokban bekövetkező változásokat is. Nagyon jól együtt tud működni a Tensorflow-al, hiszen ahhoz készítették, illetve ezáltal a Keras-al is. [18]

8.4. Rendszerterv

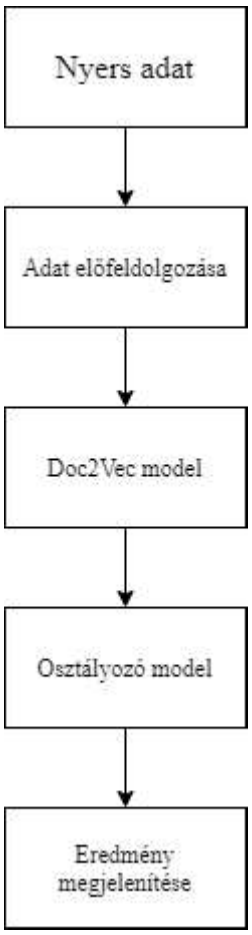
A feladat tényleges megvalósítása két nagyobb részre bontható. Az első rész a dokumentumok átalakítására szolgáló modell felépítése és paramétereinek beállítása. A második egység magának az osztályozó modellnek a felépítése. A kész szoftver már képes lesz valamilyen, az adott osztályokba besorolható nyers jogi dokumentumokat feldolgozni, és ezeket a feldolgozott, megtisztított dokumentumokat átalakítani szám vektorokká, majd ezeket a vektorokat adja tovább a modellnek, amely a már tanult minták alapján megpróbálja eldönteni melyik osztályba sorolhatók.

A dokumentumokat ketté választom. Az egyik része, azaz a rendelkezésemre álló tanító adatok 80%-át használom fel tanítási adatnak, a maradék 20%-ot pedig validációs adatnak fogom felhasználni.
Ezen kívül szükségem van még egy teszt halmazra is, amellyel a már betanított hálózat működését fogom tesztelni. Ezeknek olyan adatoknak kell lenniük, amelyeket még nem látott eddig a hálózat, ezért ezeket az adatokat előre el fogom különíteni.

A tanítási adathalmazzal elkezdem első körben a Doc2Vec betanítását, amely alapvető feltétele annak, hogy használható bemenetet tudjak átadni az osztályozómnak.

Ezután az elkészített Doc2Vec modellt tudjuk felhasználni, hogy ténylegesen átalakítsuk a bemenetünket. Ez egy időigényes folyamat, mivel, ha minél pontosabban szeretnénk ábrázolni a dokumentumainkat annál több korszakot kell használnunk, ami ennyi adatnál már jelentős időt vesz igénybe. Azért jelentene problémát a túl kevés korszak

alkalmazása, mivel így jóval nagyobb eséllyel kaphatunk egymástól akár teljesen eltérő dokumentum ábrázolásokat is.
Tehát a rendszert az alábbi képen jelzett folyamatábra alapján fogom megvalósítani.



2. ábra: Rendszerterv

A rendszert két különböző irányból kell megvizsgálni. Egyik esetben a még be nem tanított rendszert, másikban pedig amikor már a lehető legjobb eredményt adja vissza. A nagyobb részt a rendszer elkészítésében a betanítás folyamata és a paraméterek megfelelő beállítása veszi igénybe.
Mindkét esetben szükséges az adatok feldolgozása. Lényegében sima txt fájlból beolvasott adatokkal fog dolgozni a rendszer. Miután elvégzi a felesleges részek eltávolítását a bemeneti adatból, azt átküldi a Doc2Vec modellen és kimenetként egy valamilyen előre beállított hosszúságú számvektor ad vissza, amelyet átad az osztályozó modellnek. Ez a modell egy becslést készít arra, hogy az adott dokumentumot hány százalékosan sorolja be valamelyik osztályba.
Ezek közül az osztályok közül a 3 legnagyobb százalékkal rendelkezőt kiemeljük és ezek lesznek azok, amelyeket a modell a leginkább hozzáillőnek gondolt.

Mivel a rendszert rengetegszer kell majd letesztelni, nem csak azért, hogy működőképes vagy sem. Ez a kisebb része. Nagyobb részben a paraméterek jó beállítását kell megtalálni. Ez azt jelenti, hogy többször is ki kell edzeni ugyan azzal az adattal egymástól eltérő paraméterekkel rendelkező modelleket.
Egy ilyen modell elkésztése, például a Doc2Vec esetében az adatok felének felhasználásával körülbelül 6-7 órát vesz igénybe. Mivel nagyon magas az időigénye egy ilyen tesztnek, érdemes lehet erre valamilyen automatizmust építeni bele, ami helyettünk elvégzi a teszteket. Itt arra gondolok, hogy minden egyes paraméterhez különböző tömböket veszünk fel, és ezekbe a legnagyobb potenciállal rendelkező értékeket rakjuk. Ezeket az értékeket különféle képen összepárosítjuk és kieddzük vele a modellt.

Fontos, hogy nem érdemes az egész adatkészletet felhasználni ezekre a tesztekre, legalábbis az első pár körben amikor a legrosszabbakat szeretnénk kiszórni, hiszen ez rengeteg időt venne igénybe. Ezért csak a már kifejezetten nagy potenciállal rendelkező paraméter listákat érdemes kipróbálni ténylegesen nagy adatkészleten. Én úgy képzelem el ezt a folyamatot, ahogy már korábban a Doc2Vec esetében leírtam, hogy vannak különböző módszerek arra, hogy miként tudjuk kiértékelni az elkészült modell teljesítményét. Ezeket a kiértékelt adatokat egy külön fájlban, valamilyen struktúrában eltároljuk. Itt érdemes csv fájlt alkalmazni a könnyebb feldolgozhatóság érdekében.
Tehát miután lefutott minden kezdeti teszt, az összegyűjtött adatok alapján a paraméterek közül kiválasztjuk a legjobbakat és egy következő körben ezeket már nagyobb adatkészlettel teszteljük le újra.
Ezáltal nagy valószínűséggel közel a legjobban felparaméterezett modellt tudom majd elérni.

Az osztályozó modell esetében kicsit másképp zajlik majd a tesztelés, mivel itt már kevésbé a paraméterektől fog függeni a hálózat, itt már egy teljesen általam feléptett hálózatról van szó.
Nagyban befolyásolja a teljesítményt, hogy milyen és mennyi réteget fogok majd felhasználni. A bemeneti réteg lényegében majd attól függ, hogy milyen nagyságú vektorok esetében adja vissza a Doc2Vec a legjobb dokumentum beágyazást.
A hálózat edzése közben folyamatosan adatokat gyűjtök arról, hogyan változik a pontosság és a veszteség függvény értéke. Ezt a matplotlib könyvtár segítségével tudom majd kirajzolni diagrammokra.
Emellett a TensorBoard lesz még nagy segítségemre, amivel az edzések közben kinyert adatokat, mint a paraméterek, hálózat, futásidő és a teljesítmény elmentem egy fájlba, hogy a későbbiekben össze tudjam hasonlítani a modelleket.

Mindkét esetében nagyon lényeges kérdés az idő, hiszen nem áll rendelkezésünkre végtelen idő, és sajnos egy túl nagy hálózat kiedzése akár napokat is igénybe vehet.

Ezért amellett, hogy a lehető legnagyobb pontosságú hálózatot szeretném elérni, a kiedzési időt is figyelembe kell vennem majd.

8.5. Kiértékelési technikák

Egy kiképzett neurális hálózat esetében az utolsó lépés az, hogy meg kell tudnunk állapítani, milyen teljesítményen tud működni az adott hálózat. Több módon is meg kell vizsgálni egy modellt annak érdekében, hogy minden tekintetben kielégítő választ kapjunk arra mennyire pontos a modellünk.
A következő részben az erre alkalmas technikákat fogom leírni, és a későbbiekben a modell tesztelésénél fel is fogom őket használni.

8.5.1. Modell pontosság függvény

A modell pontosságát alap esetben egyszerűen meghatározhatjuk azzal, hogy megnézzük egy még nem látott felcímkézett minta halmazra, milyen pontossággal adja vissza az adott címkéket. Ebben az esetben csak annyit fogunk látni, hogy a modell az osztályok összességében hányszor tévedett, de egyéb információ nem fog a rendelkezésünkre állni, tehát az is megtörténhet, hogy csak 1 osztály esetében lesz kimagaslóan magas tévedés, ezért önmagában ez az egy szám nem írja le egy modell valódi használhatóságát.
A Tensorboard nevű rendszer el fog készíteni számomra egy grafikont, amely leírja, hogy a tanítási és a validációs halmazon milyen eredményeket tudott felmutatni a modell az adott képzési korszakban.
Emellé lesz még szükség egy független adathalmazon végzett tesztre is, amely a már kész modellen mutatja majd meg a pontosságát.

8.5.2. Logaritmikus veszteség függvény

Ez a függvény a hamis besorolások büntetésével működik. Nagyon jól használható több osztályos besorolás esetén, mivel ebben az esetben minden osztályhoz egy valószínűséget rendelünk.
A függvény a [0, ∞[tartományban van értelmezve, ebben a tartományon belül akkor ér el általában nagyobb pontosságot a modell, ha ez az érték minél közelebb van a 0-hoz, ezért ennek az értéknek a minimalizálására kell törekedni. [24]

8.5.3. Confusion matrix

Valamilyen osztályozási modell kiértékelésére szolgáló technika, amely megmutatja, hogy a modell mely osztályok tekintetében követ el inkább hibákat, és amennyiben egy hibát elkövet, akkor az a hiba milyen irányba vagy osztály felé történik.

Erre azért van szükség, mivel gyakran egy eredmény bár nagyon jónak tűnhet, de meg kell vizsgálni azt is, hogy a tévedések miként és milyen irányban történtek, hogy pontosabb következtetéseket tudjunk levonni. Itt nem mindegy, hogy egy pár százalékos tévedés például orvosi értelemben véve pozitívból negatív irányba vagy éppen fordítva történik.
Ez az én esetemben azért is fontos, mivel nem tudok teljesen kiegyensúlyozott tanítási halmazt beszerezni, ezért így nagyobb rátekintést fogok kapni az osztályok eloszlására. [23]

Ez az én esetemben egy 20*20-as mátrix lesz, amelynek ideális esetben az átlójában elhelyezkedő mezőbe beírt számok összege megegyezne a teszt halmaz elemeinek számával, vagyis ebben az esetben lenne a modell 100%-os teljesítményű.
A következőt lesz érdemes vizsgálni, hogy ha téved a modell akkor mely osztály irányába téved főként.
Esetünkben van rá lehetőség, hogy 2 osztály között nagy a hasonlóság, ezáltal az elemek egymáshoz közel helyezkednek el, vagyis hasonló vektorok írják le őket, így a tévedés mértéke kisebb, mint két egymástól nagy mértékben különböző osztály esetében.

8.5.4. F1 pontszám

Az F1 pontszám a mintán mért besorolások alapján kapott pontosság és a visszahívás közötti harmonikus átlag.
Ebben az esetben a pontosságot a következőképpen tudjuk leírni: A megfelelő osztályba besorolt elemek száma osztva az összes olyan minta számával, amelyeket az osztályozó az adott osztályba próbált beosztani akár helyesen vagy akár helytelenül.
A visszahívás pedig: A megfelelő osztályba besorolt elemek száma osztva minden olyan elem számával, amelyet a modell tökéletes működése esetén az adott osztályba kellett volna besorolnia.
Az F1 pontszámot 0-tól 1-ig terjedő skálán lehet ábrázolni, és 100-zal felszorozva százalékos értéket kaphatunk, amely minél közelebb van a 100%-hoz annál jobb a modellünk. [24]

9. Implementálás

Ebben a fejezetben magáról a megvalósításról, a véglegesen felhasznált technológiákról és a közben felmerült akadályokról, valamint azok megoldásáról fogok írni.

9.1. Adatok beszerzése, feldolgozása és elemzése

Első körben írtam egy email-t az EUR-LEX Helpdesk-nek, hogy miként lehetne beszerezni ezeket a dokumentumokat, vagy esetleg el tudnák-e küldeni nekem valamilyen formában. Ebben így nem tudtak segíteni, viszont kaptam pár linket amelyek segítségével végül megtaláltam a megoldást a problémámra és sikerült a következő módon beszereznem a forrás adatokat:

Ahhoz, hogy a projekt megvalósítható legyen, szükségem volt a dokumentumokra, amelyek az eur-lex.europa.eu oldalon elérhetőek. Ezen a weboldalon volt arra lehetőség, hogy 100 MB adatmennyiségig kérjek le információt a dokumentumokról maximálisan. Ezért éves bontásban kikértem az összes olyan magyar nyelvű dokumentumot, amely már tartalmazta a fejezetkódot, mint címkéket. Ezeket igénylés után csv formátumban kaptam meg Email-ben, viszont itt csak a dokumentum címe, celex kódja és a címke volt látható. Ezután találtam egy link sémát, amelyet ezekből az adatokból ki tudtam generálni egy újabb csv fájlba, így minden dokumentumhoz létrejött egy link.

celex	classid	classname	link
22017A1222(01)	11	Külkapcsolatok	https://eur-lex.europa.eu/legal-content/HU/TXT/HTML/?uri=CELEX:22017A1222(01)
22016A1203(02)	11	Külkapcsolatok	https://eur-lex.europa.eu/legal-content/HU/TXT/HTML/?uri=CELEX:22016A1203(02)
52012IP0478	11	Külkapcsolatok	https://eur-lex.europa.eu/legal-content/HU/TXT/HTML/?uri=CELEX:52012IP0478
52012IP0469	17	A vállalkozásokra vonatkozó jogszabályok	https://eur-lex.europa.eu/legal-content/HU/TXT/HTML/?uri=CELEX:52012IP0469
52012IP0492	5	A munkavállalók szabad mozgása és szociálpolitika	https://eur-lex.europa.eu/legal-content/HU/TXT/HTML/?uri=CELEX:52012IP0492
52012IP0505	11	Külkapcsolatok	https://eur-lex.europa.eu/legal-content/HU/TXT/HTML/?uri=CELEX:52012IP0505
52012IP0482	15	Környezet, fogyasztók és egészségvédelem	https://eur-lex.europa.eu/legal-content/HU/TXT/HTML/?uri=CELEX:52012IP0482
52012BP0485	1	Általános, pénzügyi és intézményi ügyek	https://eur-lex.europa.eu/legal-content/HU/TXT/HTML/?uri=CELEX:52012BP0485
52012IP0488	5	A munkavállalók szabad mozgása és szociálpolitika	https://eur-lex.europa.eu/legal-content/HU/TXT/HTML/?uri=CELEX:52012IP0488

3. ábra: Dokumentum linkek eltárolása

Ezeken a linkeken elérhetővé váltak a dokumentumok, és innen tudtam letölteni őket. Erre készítettem egy letöltő metódust, amely végigment az összes linken és letöltötte az oldal tartalmát, viszont ezek a letöltött fájlok html kódot is tartalmaztak ezért ezeket ki kellett vonni belőlük.
Ezután minden dokumentumból elkészült egy txt fájl, amelyet kimentettem egy könyvtárba, és későbbiekben ezeket már fel tudtam használni.

Ezek mellett szükség volt még egy olyan fájlra, amely tartalmazza az osztályokat a dokumentumokhoz kapcsolva. Ezeket úgy oldottam meg, hogy generáltam egy csv fájlt, amely tartalmazza a celex kódot és mellé az osztálynak az azonosító számát.
Ez látható a következő ábrán:

celex	classid	classname
22017A1222(01)	11	Külkapcsolatok
22016A1203(02)	11	Külkapcsolatok
52012IP0478	11	Külkapcsolatok
52012IP0469	17	A vállalkozásokra vonatkozó jogszabályok
52012IP0492	5	A munkavállalók szabad mozgása és szociálpolitika
52012IP0505	11	Külkapcsolatok
52012IP0482	15	Környezet, fogyasztók és egészségvédelem
52012BP0485	1	Általános, pénzügyi és intézményi ügyek
52012IP0488	5	A munkavállalók szabad mozgása és szociálpolitika

4. ábra: Dokumentum osztályok eltárolása

Ezek után a dokumentumokból eltávolításra kerültek a stop szavak, mivel ezek olyan szavak, amelyek a magyar nyelvben nagyon magas számban fordulnak elő, így érdemi információt nem hordoznak a címkék viszonylatában, viszont nagyban megnöveli jelenlétük a dokumentumok hosszát, ezáltal a képzési időt is.

```
from nltk.corpus import stopwords


def deleteStopWords(word_tokens):
    stop_words = stopwords.words('hungarian')
    return [w for w in word_tokens if not w.lower() in stop_words]
```

5. ábra: Stop szavak törlése

A képen lévő metódus kiszűri az összes olyan szót, amely az adott nyelvben, jelen esetben a magyar nyelvben, gyakran fordulnak elő, így nem hordoznak magukban egy osztályra specifikus információkat. A metódusnál lényeges lépés a szavak kisbetűssé alakítása, hiszen ellenkező esetben a mondatok kezdő szavait különböző szavaknak érzékelné a nagy kezdőbetű miatt, és nem kerülnének eltávolításra.
word_tokens: Egy adott dokumentumnak a szavai, paraméterként
return: A stop szavak nélküli dokumentum visszaadása

Ez a metódus minden esetben lefut, akkor is, ha egy dokumentummal szeretnénk tesztelni a modell működését, abból a dokumentumból is ki kell vonni minden olyan szót és zavaró tényezőt, amelyet az edzés folyamatában nem használtunk fel.

9.2. Dokumentumok átalakításának technológiája (Doc2Vec)

A dokumentumok szám vektorokká való átalakítására a Doc2Vec technológia segítségével került sor. Ezen belül is nem egy már előre kiedzett modellt kerestem, hanem

a rendelkezésemre álló adatokat használtam fel a modell kiképzéséhez, így a saját speciális szókincsemet tudtam megtanítani a modellel.
Az első körös tesztekben még kevésbé szűrtem meg a dokumentumokat, így a kiedzés futás ideje több napot vett volna igénybe, ezért arra a következtetésre jutottam, hogy kivonom a dokumentumokból a legtöbbet használt szavakat így a képzési idő fél napra csökkent.
A következő lépés a paraméterek megválasztása volt ennek az eldöntésében az internet, illetve a saját magam által végzett tesztelgetés segített.

A modell beállításainál öt szóban határoztam meg azt, hogy mennyi legyen az a legkevesebb előfordulás, amelyben a modell figyelembe vegye az adott szót. Ez azért lényeges, mivel így sok értelmetlen szót ki tud szűrni, amelyek a dokumentumok beszerzése közben esett hibák következtében kerültek bele.
Azért tartom megfelelőnek ezt a számot mivel így nem esik nagyon vissza a korpusz, és emellett nem lesz sok olyan szó, amelyet a modell még sosem látott, ami majd a későbbi új dokumentumok esetén lesz lényeges. Persze ez nem véd minket az új dokumentumok esetében, hogy a kihagyott szavak közül végül később mégis bekerüljenek az osztályozás folyamatába.

A modell vektor méretét 300-ban határoztam meg, az internetes keresések alapján ezt a méretet ajánlották a legtöbb helyen.

9.3. Az osztályozó modell felépítése és kiképzése (Keras)

Kezdésként meg kell teremteni a megfelelő feltételeket, hogy az adatokkal dolgozni tudjon a modell, ez három fő részből áll.
A dokumentumokat ugyan olyan módon kell feldolgozni, mint ahogy a doc2vec modell betanítottuk és ehhez kapcsolódik a második lépés is, hogy elkészítjük a numerikus vektorokat. Ezt viszonylag innentől egyszerűen meg tudjuk tenni, csak átadjuk a doc2vec modellnek és az visszaadja számunkra a már előre meghatározott hosszúságú, a mi esetünkben 300 számból álló vektort.
A másik lényeges lépése a modell kiképzésének elkezdéséhez, az osztályok meghatározása. Itt a modell nem tud mit kezdeni a szöveggel leírt osztályokat, így ezeket számszerűsítenünk kell, viszont ez sem megfelelő a modell számára. Itt kell alkalmazni a One-Hot kódolású vektorokat.

Jelen esetben ezek 20 hosszúságú számvektorok lesznek melyek a következőképpen fognak kinézni:

```
Label_encoder = LabelEncoder()
Label_encoder.fit(train_labels)
train_y = np_utils.to_categorical(Label_encoder.transform(train_labels))
```

6. ábra: Osztály címkék One-Hot vektorokká alakítása

A következő képen pedig az adott osztályok címkéjéhez tartozó számok, és az azokból kialakított One-Hot vektorok láthatóak:

```
11 = [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
18 = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
9 = [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
17 = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
5 = [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

7. ábra: Osztályok címkéje One-Hot vektorrá alakítva

A következő lényeges lépés maga a modell elkészítése. Itt a felhasznált rétegek, azok méretei, az aktivációs függvények és még számos egyéb beállítás tudja befolyásolni mind a modell képzési idejét, mind pedig annak hatékonyságát.
Az aktivációs függvény szempontjából megközelítve a problémát két alap esetet néztem meg. Az egyik a sigmoid, a másik pedig a relu, illetve annak különböző változatai.

A sigmoid függvény esetében sokkal szebben, nagyobb ugrálások nélkül nőtt a pontosság, viszont minden esetben elmaradt azoktól az esetektől, amelyekben a relu aktivációs függvényt használtam. Így tapasztalati úton arra a következtetésre jutottam, hogy az én problémám esetében a legjobb választás a relu aktivációs függvény.
Itt a különböző fajtái között már látványosan nagy eltérést nem tapasztaltam. Fontos része a modellnek az utolsó rétegben alkalmazott softmax aktivációs függvény, mivel ennek segítségével alakul ki a modell végleges kimenete.
A Dense rétegek használatát részesítettem előnyben, egyik számomra nagyon fontos érv, ami mellette szólt, hogy minimális időt vett igénybe ilyen rétegekkel a modell kiedzése, illetve a tapasztalataim alapján így sikerült a lehető legjobb eredményeket elérnem.
A modellemben fontos szerepe van még a Dropout rétegeknek, amelyek segítségével növelni tudtam a modell hatékonyságát, emellett a túlillesztés ellen is bizonyos mértékű védelmet nyújt.
Itt a legjobb értéknek a 20%-kos esélyt találtam arra, hogy mennyi legyen a valószínűsége annak, hogy kihagy egy neuront.

```
def buildModel():
    model = Sequential()
    model.add(Dense(512, input_dim=300, activation='relu'))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(20, activation='softmax'))
    return model
```

8. ábra: Osztályozó modell

9.4. Modell kiképzés gyorsítása

Mindkét esetben problémát okozott a lassú kiképzés, és mivel én nem rendelkezek korlátlan erőforrással, ezért szükség volt arra, hogy valahogy egyszerűbben és gyorsabban tudjam a modelleket kiképezni és tesztelni is közben.
Az osztályozó modell esetében első körben minden egyes dokumentumot át kell alakítani szám vektorrá, amely 36.000 dokumentum esetében már nagyon jelentős időt vesz igénybe. Ez függ a Doc2Vec modell méretétől és a korszakok számától amíg próbálja beállítani a megfelelő számokat a Doc2Vec. Ezzel szemben az egyszerűsített Doc2Vec modellem esetén is 5-6 órát vett igénybe, amely minden osztályozó modell kiképzés előtt meg kell történnie.
Ennek a kikerülésére találtam ki azt a megoldást, hogy egyszer generálom le és el tárolom a 300 számból álló vektorokat egy txt fájlban, és minden modell edzés előtt ezt olvasom be, ez pedig alig pár másodpercet vesz igénybe.

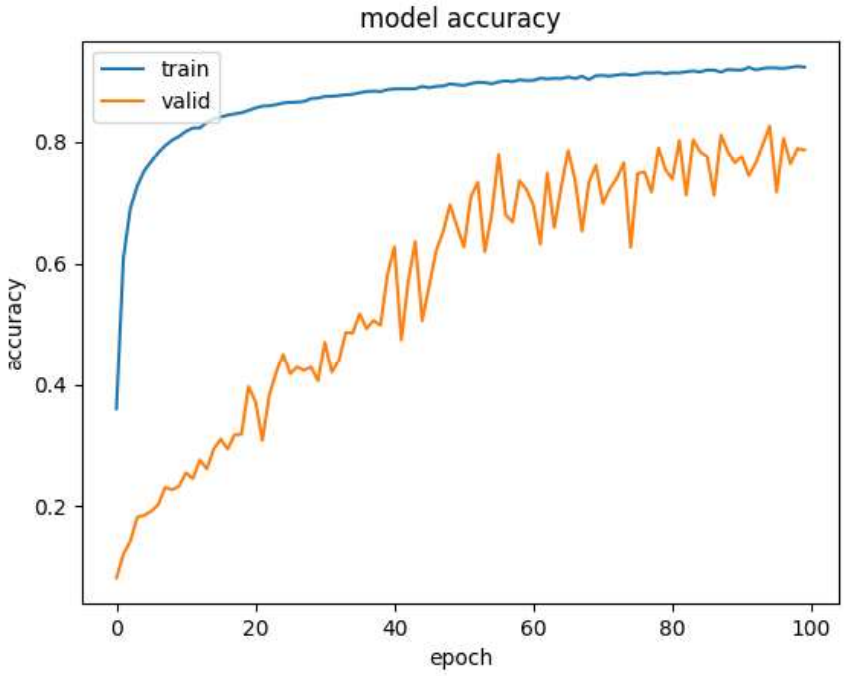
9.5. Modell pontosítási lehetőségek

A SMOTE implementációja alapjában véve nagyon egyszerű, mivel csak át kell adni neki a már megfelelően feldolgozott adatokat. Ezalatt egyik részről magukra a numerikus dokumentum vektorokra gondolok, másik részről pedig a hozzájuk tartozó One-Hot vektorokká alakított címkékre. Ezekből képes a SMOTE generálni új adatokat bármilyen mennyiségben.
Az első esetben csak erre a módszerre támaszkodva, minden osztály adatainak számát azonos mértékre hoztam. Ez azt jelenti, hogy minden osztályban pont annyi adat lett végül, mint a legtöbbet tartalmazó osztályban.

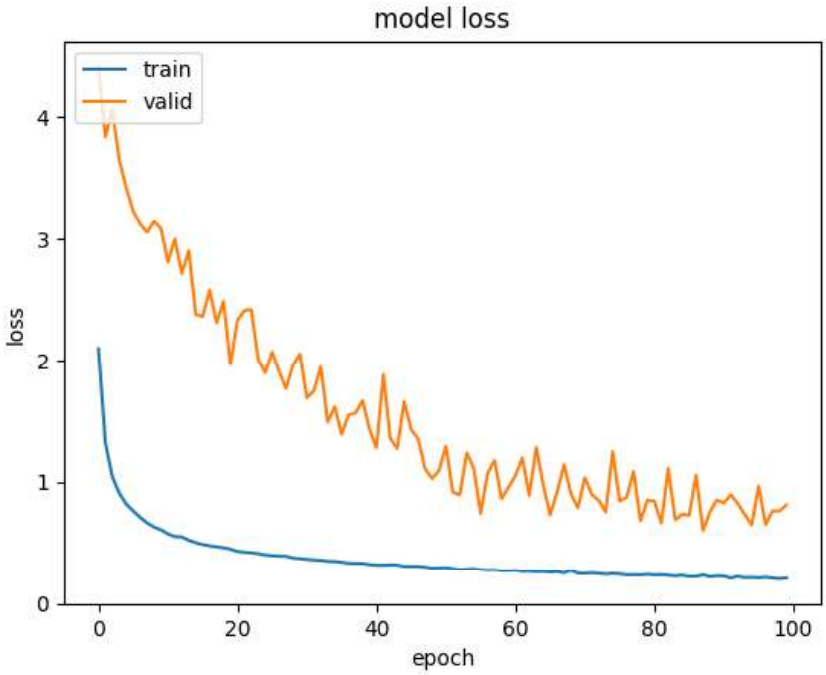
A következőkben, azonos neurális hálózat mellett próbáltam ki mind a két esetet, és a következőket tapasztaltam a modell kiedzése során:

Az osztályok közül a legkevesebb elemet tartalmazó 107, a legtöbbet tartalmazó 5673 egymástól különböző dokumentumból áll.
Alap verzióban a SMOTE minden mintából a már meglévők alapján annyit generál, amennyi hiányzik az 5673-hoz képest. Ez azt jelenti, hogy ebben az esetben rendelkezésemre már kicsivel több mint 110.000 adat áll.
Itt alapból nem lehet megállapítani, hogy a modell teljesítménye ettől nőni fog vagy sem ezért csak tényleges teszteléssel tudtam megállapítani, és a következő eredményeket tudtam elérni.

9.5.1. SMOTE használata az adathalmazon



9. ábra: Pontosság diagram (SMOTE)



10. ábra: Veszteség diagram (SMOTE)

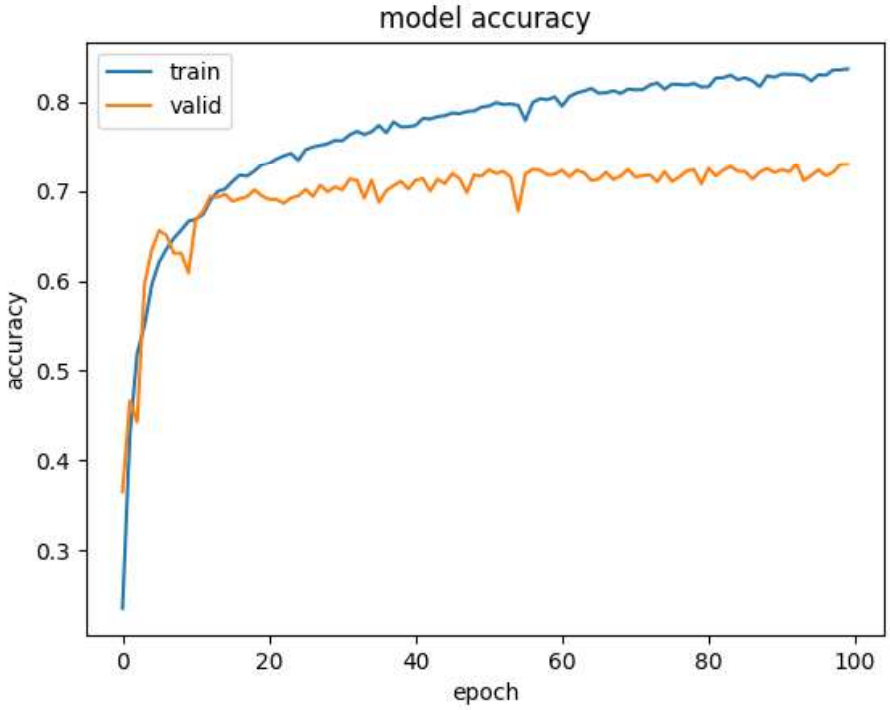
Ebben az esetben SMOTE-ot használtam az edzési adathalmazon és az alábbi végeredményeket kaptam:

- A modell 100 korszakon keresztül lett kiképezve

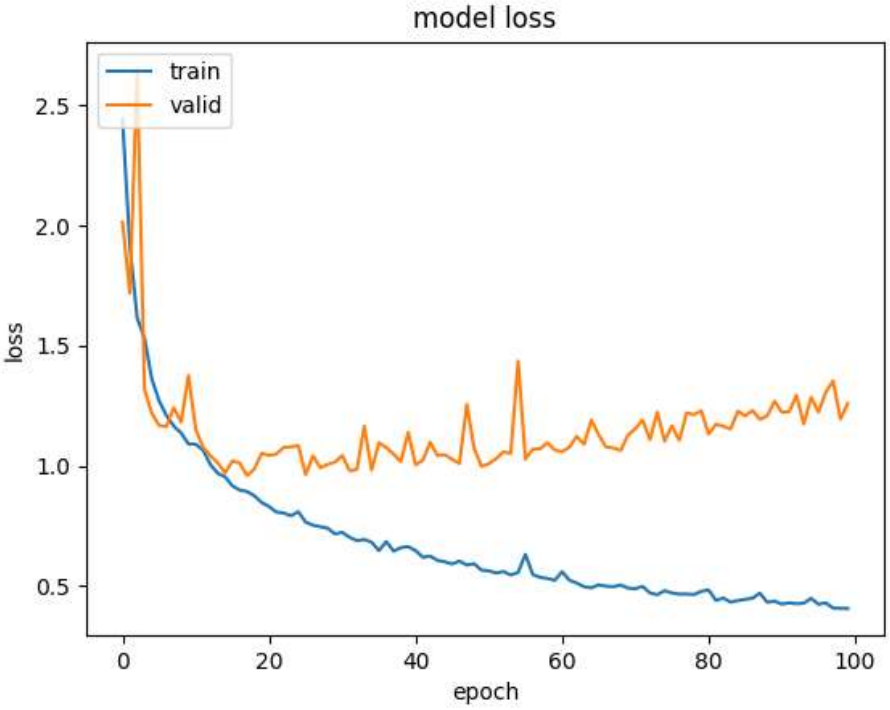
- Az edzési halmazon a végső pontosság: 92,22%
- Az edzési halmazon a végső veszteség értéke: 0,2159

- A validációs halmazon a végső pontosság: 74,09%
- A validációs halmazon a végső veszteség értéke: 0.8986

9.5.2. SMOTE használata nélküli eredmények



11. ábra: Pontosság diagram (SMOTE nélkül)



12. ábra: Veszteség diagram (SMOTE nélkül)

Az alábbi esetben nem használtam a SMOTE-ot az edzési adathalmazon és az alábbi végeredményeket kaptam:

- A modell 100 korszakon keresztül lett kiképezve

- Az edzési halmazon a végső pontosság: 83,63%
- Az edzési halmazon a végső veszteség értéke: 0,4060

- A validációs halmazon a végső pontosság: 73,03%
- A validációs halmazon a végső veszteség értéke: 0,7303

9.5.3. SMOTE használata és használata nélküli eredmények összehasonlítása

	SMOTE használatával	SMOTE használata nélkül
korszakok száma	100	
edzési halmazon a pontosság	**92,22%**	83,63%
edzési halmazon a veszteség	**0,2159**	0,4060
validációs halmazon a pontosság	**74,09%**	73,03%
validációs halmazon a veszteség	0.8986	**0,7303**
teszt halmazon a pontosság	**46%**	42%

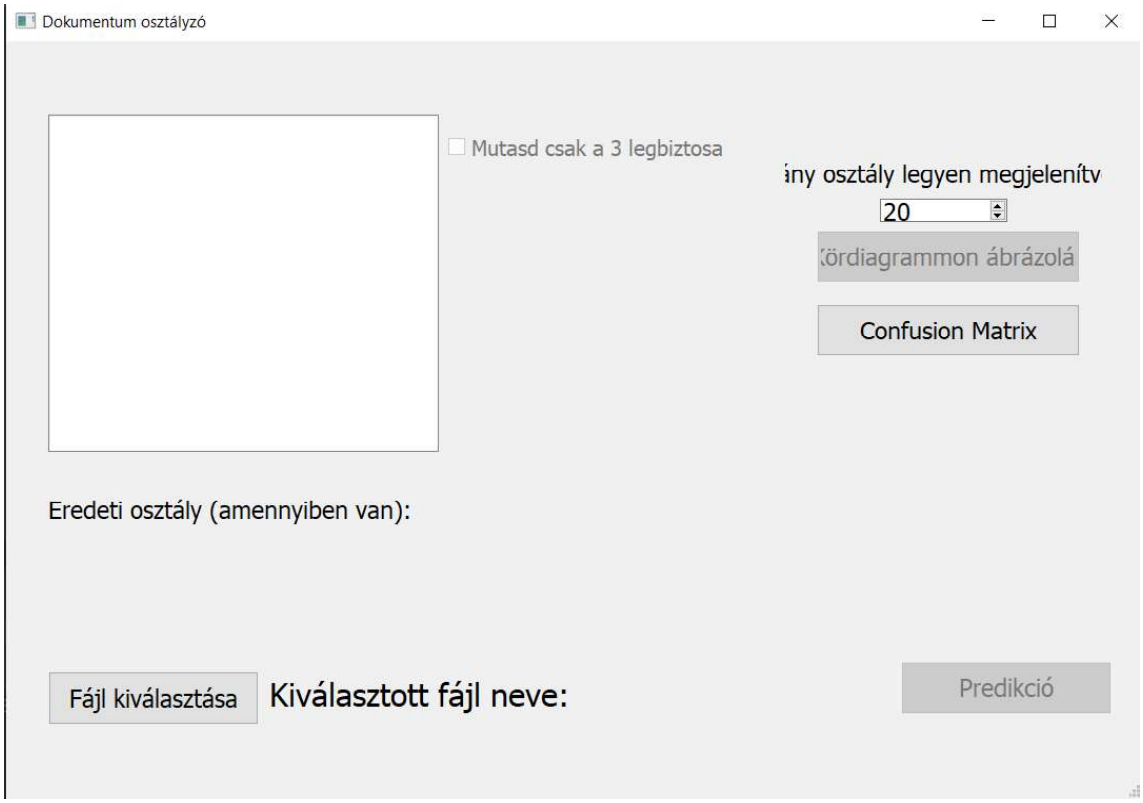
1. táblázat: SMOTE használata és használata nélküli eredmények összehasonlítása

9.6. Segéd metódusok

- **getRootFolderPath():** Visszaadja a projekt mappájának a könyvtárát, ez a különböző elemek elérése miatt szükséges.
- **getClassCsv():** Beolvassa az osztályokat tartalmazó csv fájl-t.
- **getTestClassCsv():** Beolvassa a teszt adatokhoz tartozó osztályokat tartalmazó csv fájlt.
- **loadDoc2VecModel():** Betölti a már kiedzett Doc2Vec modellt.
- **loadLegalPredictModel():** Betölti a már kiedzett osztályozó modellt.
- **getAllTrainDataPath():** Visszaadja az össze tanító minta elérési útját.
- **readAllTrainingData():** Beolvassa az összes tanító dokumentumot.
- **readDataByCount(count):** Egy megadott számú tanító dokumentumot olvas be, erre a kisebb számú tesztadattal történő tesztek esetén volt szükség.
- **writeVectorisedDocumentsToTXT(vectorList, classIdxList, isTrain):** Elmenti a már Doc2Vec modell által vektorrá alakított dokumentumokat leíró vektorokat egy txt fájlba.
- **loadVectorisedDocuments(isTrain):** Betölti a már korábban létrehozott dokumentum vektorokat.
- **getDocVectorsCSVPath():**
- **getClassIndexCSVPath():** Ez a két metódus elérési utat ad vissza.
- **IsTestDocVectorFilesExists():**
- **IsDocVectorFilesExists():** Megvizsgálja, hogy létezik-e a dokumentum vektorokat tartalmazó fájl, és amennyiben nem akkor kell csak újra átalakítani a dokumentumokat.
- **laodStopWordsIfNeed():** Amennyiben szükséges letölti a Magyar nyelvben legtöbbet használt szavakat.
- **deleteStopWords(word_tokens):** Az átadott szavakat tartalmazó listából kitörli azokat a szavakat, amelyek benne vannak a magyar nyelvben legtöbbet használt szavak között
- **documentNumberInClasses():** Visszaadja, hogy egy adott osztályon belül hány darab tanító mintával rendelkezünk

9.7. Kliens alkalmazás

A modellhez készítettem egy egyszerű kliens alkalmazást, amely alapvetően a teszteléshez szükséges alap funkciókat tartalmazza. Úgy, mint dokumentum kiválasztása, Dokumentum kiértékelése, és kisebb statisztikai elemek megmutatása a visszakapott adatok alapján, illetve még előállítja a confusion mátrixot is, amelyről a következő fejezetben fogok bővebben írni.



13. ábra: Kliens alkalmazás

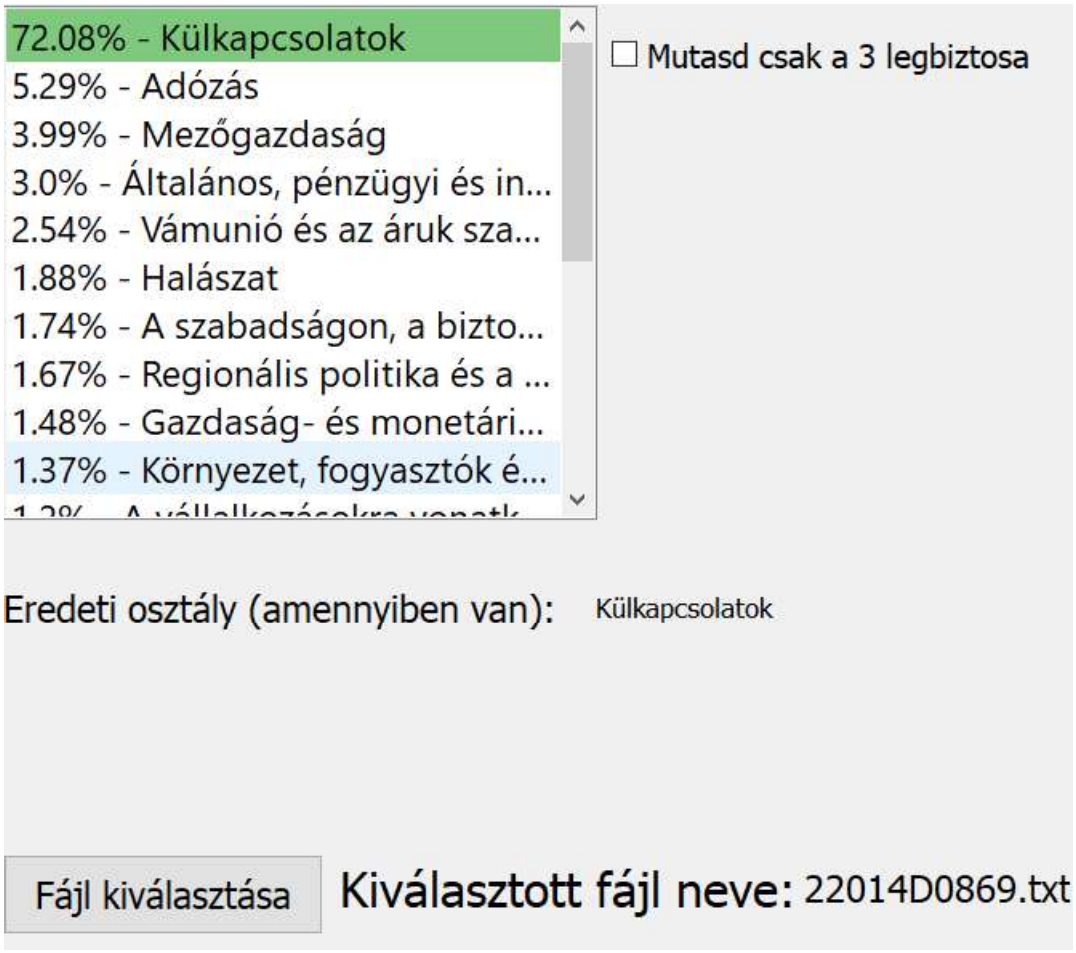
Funkciók:

- **Fájl kiválasztása:** Jogi dokumentum kiválasztása txt formátumban.
- **Predikció:** A gomb egy dokumentum kiválasztása után válik aktívvá, ekkor a gombra kattintva a modell kiértékeli a dokumentumot.
- **Confusion Mátrix:** A kigyűjtött tesztadatok alapján elkészít egy Confusion Mátrixot
- **ListBox:** Itt jelenik meg százalékos arányban, csökkenő sorrendben rendezve mind a 20 osztály és a hozzájuk rendelt valószínűség, hogy melyik osztályba tartozhat a dokumentum.
- **Kördiagramm:** Itt is az eredményt lehet megjeleníteni, csak itt összevonhatók az osztályok egy körcikkbe.

10. Eredmények értékelése

10.1. Modell működése

Ahogy már az előző fejezetben bemutattam, készítettem a modellhez egy egyszerű kliens alkalmazást, amely a teszteléshez szükséges alap funkciókat tartalmazza, valamint előállítja a confusion mátrixot is, amelyről a következő pontban fogok bővebben írni.



14. ábra: Tesztelési felület

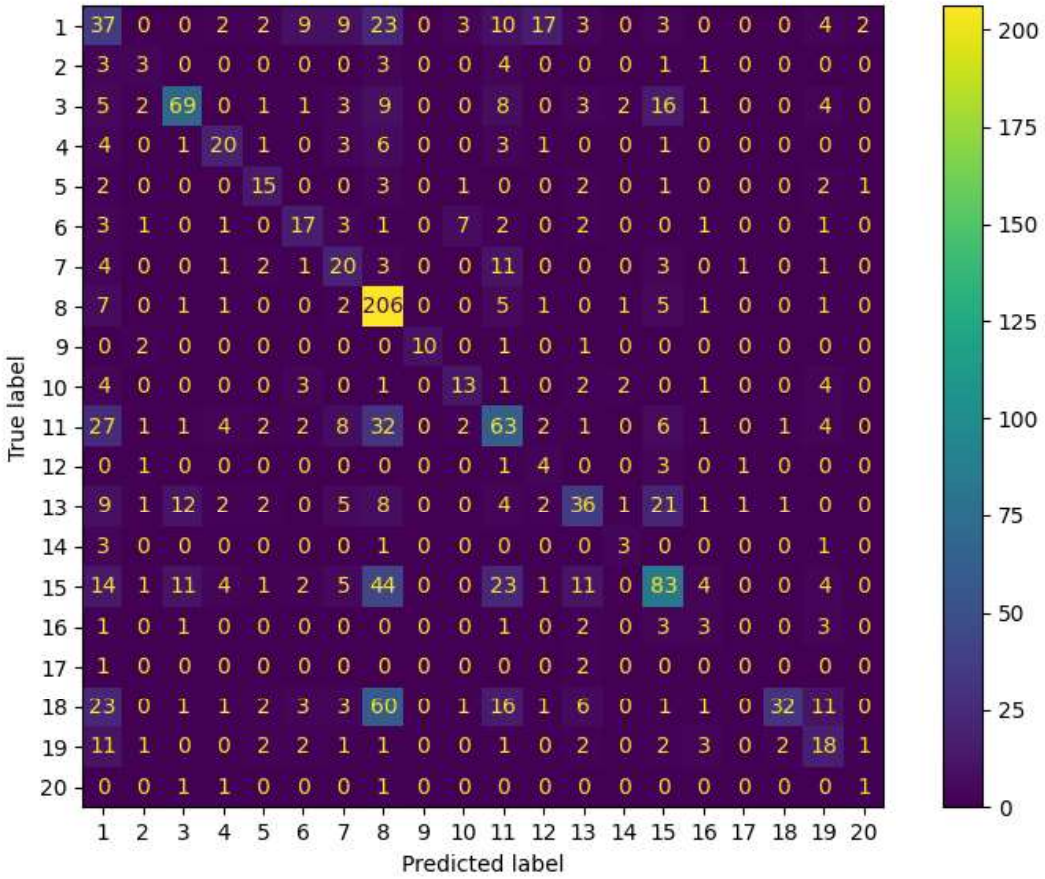
A fenti képen egy egyszerű felület látható, amelyet a modell tesztelésére hoztam létre. Itt egyszerűen kiválasztunk egy txt fájlt, amely lehetőleg ténylegesen egy valódi Európai Uniós jogi dokumentum, hiszen csak ebben az esetben lesz az eredmény releváns. Amennyiben egy megfelelős dokumentumot választunk ki és annak van címkéje, akkor azt kijelzi. Ez jelen esetben a ’Külkapcsolatok’.
A képen látható még egy lista, amelyben, zölddel van kiemelve az az osztály, amely ténylegesen a címke szerint is az, és ideális esetben ez szerepel a legnagyobb valószínűséggel.

Jelen esetben 72,08% eséllyel sorolta a Külkapcsolatok közé az átadott dokumentumot. Így van egy viszonylag kicsi bizonytalanság a modell részéről, de ez nem feltétlen jelent problémát amiatt, hogy a dokumentum tartalmazhat olyan elemeket, amely más témakörök részét képezik, ezért nem feltétlen mindig egyértelműen besorolható minden dokumentum minden osztályba.

10.2. Confusion mátrix

A confusion mátrix kigenerálásánál 1411 darab, a modell által eddig még sosem látott dokumentumot használtam fel. Ezek a dokumentumok hasonló eloszlásban tartalmazzák az egyes osztályokat, mint a tanító adathalmaz.
A mátrixon jól látható, hogy osztályonként eltérő a modell teljesítménye, ez azt jelenti, hogy vannak olyan esetek amikor jóval nagyobb pontossággal tudja az adott osztály dokumentumait besorolni a többihez képest, amit a tanító minta egyenlőtlensége okoz.

Ezt a problémát egy ilyen szakterületen, ahol nem függ tőlünk a rendelkezésünkre álló minta összetétele, elég nehéz kijavítani úgy, hogy egyéb problémák ne lépjenek fel.



15. ábra: Confusion Mátrix

A mátrix átlójában látható, hogy hány esetben sikerült a modellnek megfelelően felcímkéznie az eddig még sosem látott dokumentumokat. A sorokban a tévesen az adott osztályba sorolt dokumentumok láthatóak, ezáltal meg lehet azt állapítani, hogy a modell esetleg tömegesen sorolt-e egy adott címkével rendelkező dokumentumot tévesen egy másik osztályba. A mátrixon megfigyelhető, hogy elég gyakori a modell azon tévedése, hogy tévesen sorol dokumentumokat a 'Versenypolitika' osztályába. Legnagyobb mértékben a 'Közös kül- és biztonságpolitika' címke helyére kerül tévesen az előbb említett címke. Ez két dolgot is jelenhet, egyik az, hogy a két témakörben jellemzően hasonló szavakat használnak. A másik eset pedig, hogy valójában mind a két címke reális lehet a dokumentum tartalmát megvizsgálva.

10.3. Osztályonkénti eredmények

Az alábbi képen a modell pontosságának kiértékelése látható, a korábban már említett adathalmazon.

	precision	recall	f1-score	support
0	0.23	0.30	0.26	124
1	0.23	0.20	0.21	15
2	0.70	0.56	0.62	124
3	0.54	0.50	0.52	40
4	0.50	0.56	0.53	27
5	0.42	0.44	0.43	39
6	0.32	0.43	0.37	47
7	0.51	0.89	0.65	231
8	1.00	0.71	0.83	14
9	0.48	0.42	0.45	31
10	0.41	0.40	0.41	157
11	0.14	0.40	0.21	10
12	0.49	0.34	0.40	106
13	0.33	0.38	0.35	8
14	0.56	0.40	0.46	208
15	0.17	0.21	0.19	14
16	0.00	0.00	0.00	3
17	0.89	0.20	0.32	162
18	0.31	0.38	0.34	47
19	0.20	0.25	0.22	4
accuracy			0.46	1411
macro avg	0.42	0.40	0.39	1411
weighted avg	0.52	0.46	0.45	1411

16. ábra: Részletes kiértékelés

Ezen a képen jól lehet látni pontosan milyen osztályból hány darab állt a rendelkezésemre, és az adott osztályokból külön-külön milyen valószínűséggel sikerült a modellnek helyes címkét visszaadnia.
Az osztályok eloszlása is látható, hogy nem egyenletes, viszont a tanító mintában is hasonló volt a dokumentumok eloszlása, lévén, hogy ilyen arányban keletkeznek ezek a dokumentumok.
A képről leolvasható, hogy bizonyos esetekben a modell magasan fölülmúlta az átlagos teljesítményt, ehhez képest más esetekben rosszabb teljesítményt nyújtott.

A legkiemelkedőbb eset a 'Közös kül- és biztonságpolitika', amelynél 162 egymástól különböző dokumentumról 89%-kos valószínűséggel állapította meg helyesen, hogy az előbb említett osztályba tartoznak.

A modell teljesítménye ezen az adatkészleten 46%. Ez abban az értelemben kissé félrevezető, hogy vannak olyan osztályok, amelyekből alig pár darab áll rendelkezésre az adatkészletben. Ezen osztályok tekintetében nem lehet azt kijelenteni, hogy a modell teljesítménye ezekre az osztályokra is hasonló eredményt érne el. Ezért érdemes megnéznünk a súlyozott átlagot is, amely kiemeli a modell azon teljesítményét, amely a többségben lévő osztályok esetében mutatkozott meg, mivel itt nagyobb súlyt fektet a modell az átlag kiszámításánál a többségben lévő osztályokra. A súlyozott átlag elérte az 52%-kos teljesítményt. Ezáltal az is megfigyelhető, hogy nagyobb biztossággal ad vissza helyes eredményt a modell azon osztályok esetében, melyek nagyon mennyiségben fordultak elő a képzési halmazban.

11. Továbbfejlesztési lehetőségek

Egy neurális hálózat esetében mindenképp az egyik legfontosabb dolog, hogy az milyen pontosságot képes elérni, ezért a nagyobb pontosság elérése mindenképp egy fontos továbbfejlesztési lehetőség. Itt főként a mikénẗje a kérdéses és ennek lehet több megközelítése is.

Minden ilyen projekt alapja az adat, és egy olyan projektben, ahol neurális hálózatot használnak sose lehet elég adat, ezért mindenképp az adatok mennyiségének növelésével lehetne javítani a modell pontosságán, ezáltal felhasználhatóságán. Itt érdemes lehet valahogy megpróbálni beleépíteni a nem felcímkézett adatokat is, akár egy jobb Doc2Vec modell elérése érdekében, így jobban rátanítva az általunk használt nyelvezetre. Persze bizonyos mennyiség fölött már jóval nagyobb erőforrásra van szükségünk.

Másik nagy fejlődési lehetőség, ha az adatokat jobb minőségben próbáljuk meg beszerezni, vagy valamilyen bonyolultabb szövegfeldolgozást alkalmazunk rá, hogy minél több fölösleges szöveget távolítsunk el belőle, viszont ehhez már szükséges speciális tudás is, amely segítségével megítélhető, hogy mely részek nem befolyásolják a dokumentumok osztályait, hiszen hasznos információt kivonni a dokumentumokból nem válna előnyünkre.

A harmadik lényeges rész, ahol javítani lehetne, az a modell rétegeinek kialakítása. Ezesetben egy esetleges bonyolultabb modell lehet, hogy jobb eredményekkel szolgálhatna, viszont ehhez nagy számú tesztelésre lenne szükség, amelyhez szintén jelentős mennyiségű erőforrás szükséges.
Így akár fejlesztési lehetőségként jelenik meg a felhő technológia használata, ez azt jelenti, hogy a megírt algoritmusokat nem a saját gépemen futtatnám le, hanem egy sokkal nagyobb erőforrással rendelkező, speciálisan ilyen algoritmusok futtatására készített távoli szerveren.

A fejlődés egy másik ága lehet a felhasználói felhasználatra való elérhetőség megvalósítása. Ez például megoldható lenne akár egy felhős alkalmazás formájában, ahol egy API-n keresztül bárki küldhetne be dokumentumokat, valamilyen webes felületen keresztül és a rendszer ezt a dokumentumot feldolgozás után átadná a modellnek és a végső eredménnyel térne vissza.

12. Összefoglalás

A projektem célja egy olyan rendszer létrehozása volt, amely képes a lehető legkisebb emberi erő befektetéssel és emellett a legnagyobb pontossággal szinte elhanyagolható időmennyiség alatt eldönteni egy Európai Uniós jogi dokumentumról, hogy melyik osztályba lehet besorolni azt fejezetkód szerint.
Megvizsgáltam több olyan technikát, amely szóba jöhetett egy ilyen neurális hálózaton alapuló osztályozó rendszer kialakításánál. Ezeket összehasonlítottam és kiválasztottam a feladat szempontjából legideálisabb könyvtárakat, technikákat.
Elkészítettem több metódust, melyek segítségével be tudtam szerezni a témához tartozó tanító dokumentumokat és hozzájuk tudtam kötni a megfelelő osztályokat.
Kiválasztottam a rendelkezésemre álló olyan dokumentum áttranszformáló technikát, amely megfelelő hatékonysággal és az én rendelkezésemre álló adatkészlet szempontjából is megfelelően képes volt áttranszformálni a dokumentumaimat numerikus vektorokká.
Ezek után célul tűztem ki magamnak, hogy minél jobb pontosságot és teljesítményt érjek el a modellel, ezeknek a kiértékelését pedig több különböző neurális hálózatoknál használt technikával is elvégeztem. Ehhez készítettem még egy egyszerű kliens alkalmazást, amely a modell tesztelésére és valódi működésének kipróbálására, bemutatására szolgál.

A modell legnagyobb pontossága elérte a 46%-ot a 20 osztály esetében. Bizonyos osztályoknál, amelyek esetében sikerült a modellt pontosabban ráilleszteni, az osztályokra jellemző tulajdonságokra ott ez a szám közel a dupláját is el tudta érni. Ha mi véletlenszerűen szeretnénk eldönteni, vagyis megtippeljük melyik osztályba sorolnánk be egy adott dokumentumot, arra a matematikai valószínűség szerint 5% az esélyünk, hogy helyesen eltaláljuk az adott osztályt, vagyis a modellünk jobban teljesít.

Ebből arra következtetek, hogy a modell és maga a projekt működőképes. Ugyanakkor egyéb felmerülő nehézségek nagy mértékben tudták befolyásolni a pontosságot negatív irányba. Ilyen az adatok egyenlőtlensége és a minősége, amiken elég nehéz érdemben változtatni, de mint jövőbeli cél mindenképpen lehetséges.

A modell betöltési ideje minimális, körülbelül 10 másodperces időt vesz igénybe, ami persze csak akkor számít, amikor szeretnénk elindítani az applikációt. Így a rendszer a gyakorlatban is jól alkalmazható. Amennyiben már fut az applikáció, onnantól egy dokumentumról anélkül, hogy mi csak egy pillanatra is belenéznénk, a rendszer a saját jóslatát szinte teljesen elhanyagolható, 1 másodpercen belül tudja visszaadni. Gondoljunk bele, hogy mennyi ez a 10 avagy 1 másodperc ahhoz képest, ami ahhoz szükséges, hogy egy átlag ember tüzetesen végig olvassa az adott dokumentumot, értelmezze annak tartalmát és döntést hozzon arról, hogy ő melyik osztályba sorolná azt. Mind emellett azt

is feltételezve, hogy az alanyunk ért a jogi szövegekhez, tehát így a legkisebb valószínűséggel hibázik.

Összességében kijelenthető, hogy a specifikációba foglaltakat sikerült megvalósítanom, és a projektem működőképesnek bizonyult.

13. Summary

The aim of my project was to create a system that can decide on a European Union legal document that in which class it can be classified, according to a chapter code, with the greatest precision while investing the least possible human effort.
I have examined several techniques that may have been considered for developing a neural network-based classification system, like the one I wanted to build. I compared these and selected the most ideal libraries and techniques for the task.
I developed several methods so that I could use them to obtain the teaching documents related to the topic and to connect the appropriate classes to them.
I chose a document transformation technique that was able to transform my documents into numerical vectors with sufficient efficiency and in terms of the data set available to me.
After that, I set myself the goal to achieve the best possible accuracy and performance with the model. I also evaluated these using several different techniques used in neural networks. For this purpose, I created a simple client application, which is used to test the model and also to test and demonstrate its real operation.

The highest accuracy of the model reached 46% for the 20 given classes. For some classes, where the model was fitted more precisely to the class-specific properties, this number could be almost doubled. If we want to decide randomly, that is, to guess which class to classify a given document, we have a mathematical probability of 5% chance that we will hit that particular class correctly, so it is obvious that our model performs better that.

From this, I conclude that the model and the project itself are functional and could be used. However, other difficulties encountered could greatly affect the accuracy in a negative direction. For example you can take the inequality and quality of the data, which is quite difficult to change substantially, but as a future goal it is certainly possible to work on.

The load time of the model is minimal, as it takes about 10 seconds, which of course only counts when you want to start the application. Thus, the system can be used well in practice. If the application is already running, then from there on the system can return its own prediction on a document without even looking at it for a moment, almost completely negligible, within 1 second. Think about how much this 10 or 1 second is compared to what it takes for an average person to read through a given document, interpret its content, and make a decision about which class to classify that document to. In addition, we also assume that our subject has an understanding of the legal texts, so that they are the least likely to make an error.

Overall it can be stated, that I managed to implement everything I stated in the specifications and my project proved to work well.

14. Irodalomjegyzék

[1] Gépi tanulás történelme
https://en.wikipedia.org/wiki/Timeline_of_machine_learning
Utoljára megtekintve: 2021.04.26

[2] Természetes nyelvi feldolgozás
https://surface.syr.edu/cgi/viewcontent.cgi?article=1043&context=istpub
Utoljára megtekintve: 2022.04.06

[3] Dokumentumok beágyazása
https://sybrandt.com/posts/document_embedding/
Utoljára megtekintve: 2021.04.22

[4] Doc2Vec működése
https://shuzhanfan.github.io/2018/08/understanding-word2vec-and-doc2vec/
Utoljára megtekintve: 2021.04.26

[5] Tanítási adatok problémái és megoldások
https://towardsdatascience.com/problems-in-machine-learning-models-check-your-data-first-f6c2c88c5ec2
Utoljára megtekintve: 2021.04.26

[6] Undersampling leírása
https://machinelearningmastery.com/undersampling-algorithms-for-imbalanced-classification/
Utoljára megtekintve: 2021.04.27

[7] Kiegyensúlyozatlan minta kezelési módjai, és SMOTE működése
https://rikunert.com/SMOTE_explained
Utoljára megtekintve: 2021.04.27

[8] Tanítási formák
https://en.wikipedia.org/wiki/Document_classification
Utoljára megtekintve: 2021.04.28

[9] TensorFlow és Keras bemutatás, jellemzői
https://hu.csstricks.net/8226560-keras-vs-tensorflow-must-know-differences
Utoljára megtekintve: 2021.05.01

[10] SMOTE bemutatása
https://towardsdatascience.com/how-to-effortlessly-handle-class-imbalance-with-python-and-smote-9b715ca8e5a7
Utoljára megtekintve: 2021.05.07

[11] fejezetkód
https://op.europa.eu/hu/web/eu-vocabularies/dataset/-/resource?uri=http://publications.europa.eu/resource/dataset/dir-eu-legal-act
Utoljára megtekintve: 2021.05.07

[12] PyTorch jellemzői
https://blog.paperspace.com/why-use-pytorch-deep-learning-framework/
Utoljára megtekintve: 2021.05.07

[13] Hosszú jogi dokumentumok osztályozása Doc2Vec használatával
https://arxiv.org/pdf/1912.06905.pdf
Utoljára megtekintve: 2021.05.07

[14] Adatok forrása
https://eur-lex.europa.eu/homepage.html?locale=hu
Utoljára megtekintve: 2021.05.07

[15] Doc2Vec modell tesztelés
https://radimrehurek.com/gensim/auto_examples/tutorials/run_doc2vec_lee.html
Utoljára megtekintve: 2021.05.07

[16] BERT jellemzői
https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/
Utoljára megtekintve: 2021.05.07

[17] BERT használata dokumentumokon
https://arxiv.org/pdf/1904.08398.pdf
Utoljára megtekintve: 2021.05.07

[18] TensorBoard bemutatás
https://www.guru99.com/tensorboard-tutorial.html
Utoljára megtekintve: 2021.05.07

[19] TensorFlow jellemzői
https://www.tensorflow.org/
Utoljára megtekintve: 2022.04.04.

[20] Keras jellemzői
https://keras.io/about/
Utoljára megtekintve: 2022.04.04.

[21] PyTorch jellemzői
https://pytorch.org/
Utoljára megtekintve: 2022.04.04.

[22] BERT jellemzői
https://cloud.google.com/ai-platform/training/docs/algorithms/bert-start
Utoljára megtekintve: 2022.04.04.

[23] Cofusion matrix
https://machinelearningmastery.com/confusion-matrix-machine-learning/
Utoljára megtekintve: 2022.04.04.

[24] Neural Network Evaluate metrics
https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234
Utoljára megtekintve: 2022.04.04.

[25] Doc2Vec jellemzői
https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e
Utoljára megtekintve: 2022.04.04.

[26] Doc2Vec, CBOW, Skip-gram jellemzői
https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314
Utoljára megtekintve: 2022.04.04.

[27] Dropout réteg
https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5
Utoljára megtekintve: 2022.04.05.

[28] Dense réteg bemutatása
https://towardsdatascience.com/introduction-to-convolutional-neural-network-cnn-de73f69c5b83
Utoljára megtekintve: 2022.04.05.

[29] Aktivációs függvények bemutatása
https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253
Utoljára megtekintve: 2022.04.05.

[30] „*A New Hybrid Based on Long Short-Term Memory Network with Spotted Hyena Optimization Algorithm for Multi-Label Text Classification*”
https://www.mdpi.com/2227-7390/10/3/488?fbclid=IwAR0ZImquQZ5XoZ8wl83hUeRfMCW67YLlQ3PKmwDBBUURVfA5bbIL-TZvnsM
PDF: *mathematics-10-00488-v2.pdf*
Utoljára megtekintve: 2022.04.06.

15. Ábrajegyzék

16. Táblázatjegyzék