Universitat Politècnica de Catalunya

Grau en Intel·ligència Artificial

Processament del Llenguatge Humà

Word Embeddings

Pau Hidalgo Pujol **Cai Selvas Sala**

5 de juny de 2024

Resum

En aquest informe s'explica el desenvolupament que s'ha fet de la pràctica sobre *Word Embeddings*, corresponent a l'assignatura de Processament del Llenguatge Humà (PLH) del Grau en Intel·ligència Artificial de la Universitat Politècnica de Catalunya (UPC).

Al llarg d'aquest document s'expliquen els passos que s'han seguit, l'estructura del codi implementat, la justificació de les decisions preses i conclusions extretes.

Índex

1 Introducció

Els word embeddings formen una part molt important del que és el processament del llenguatge natural avui en dia. Permeten obtenir representacions de les paraules sobre les quals es poden establir relacions semàntiques o altres tipus de tasques.

Aquesta pràctica tracta amb aquests de dues formes. D'entrada, es veurà el procés d'entrenament de diferents embeddings de paraules, comparant com afecten diferents mides de corpus, preprocessaments, o models (Word2Vec, CBOW...). En la segona part, s'utilitzaran aquests embeddings i, d'altres preentrenats per resoldre una tasca de similitud semàntica de frases. D'aquesta manera es podrà observar una aplicació d'aquestes representacions, així com comparar diferents models (bastats en Word2Vec, o bé en arquitectures ROBERTA...).

Addicionalment, s'han utilitzat aquests vectors per realitzar una tasca de classificació de texts, i també s'ha realitzat un finetuning propi per entendre millor aquest procés.

2 Primera part

2.1 Implementació

Tot el codi principal per tractar les dades, entrenar els models i treballar amb els vectors obtinguts es pot trobar en l'arxiu *word_vectorizer.py*, on es definex la classe `WordVectorizer`. A més, l'arxiu *part1.ipynb* conté totes les crides necessàries a aquesta classe.

Per altra banda, per l'avaluació d'aquests models s'executa l'arxiu *evaluate_models.py*, que realitza diverses avaluacions (explicades en l'apartat 2.5) i genera múltiples gràfics d'aquests resultats.

2.2 Dataset

El dataset utilitzat per l'entrenament dels diferents models és el Catalan General Crawling Corpus [1], que consta de 1016114 textos (434817705 tokens) dels 500 dominis *.cat* i *.ad* més populars durant el mes de juliol de 2020. Aquest dataset complet té una mida de 2561 MB en format *utf-8*, de manera que l'entrenament de models amb el dataset complet pot ser considerablement lent.

2.3 Preparació de dades

2.3.1 Mida del dataset

Com ja s'ha mencionat, la mida del dataset és molt gran per entrenar models ràpidament i, a més, no sempre els models entrenats amb més dades obtenen els millors resultats. Per tant, s'han definit 6 mides diferents del dataset amb les que treballar: 50MB, 100MB, 250MB, 500MB, 1000MB i 2561MB (el màxim). Aquesta mida es pot especificar en el paràmetre `size_mb` del mètode `load_data()`.

2.3.2 Tokenització

Una vegada s'ha determinat la mida de les dades, cada un dels textos s'ha de tokenitzar per paraules per tal de poder ser passat al model i entrenar-lo. S'ha valorat l'opció de tokenitzar primer per frases i després per paraules en cada frase, però s'ha considerat que això seria contra-productiu, ja que les paraules dels inicis i finals de les frases ja no tindrien les paraules del final/inici de les frases adjacents, de manera que el model tindria menys paraules en la seva finestra i segurament es perdria precisió.

Una vegada determinat que la tokenització de paraules es faria en els textos complets i no es dividiria per frases, s'han plantejat dos tokenitzadors diferents a utilitzar: el de la llibreria `spacy` i el de `nltk`. Les principals diferències entre aquests dos tokenitzadors són que NLTK és considerablement més ràpid que spaCy, però només pot tokenitzar tenint en compte les regles de l'anglès o del rus. Per altra banda, spaCy és més lent però pot tokenitzar tenint en compte la gramàtica del català. A

més, tot i ser generalment més lent, spaCy permet utilitzar CUDA amb targetes gràfiques Nvidia, de manera que es pot contrarestar part de la lentitud de la tokenització.

Per tal de tenir més models a comparar, s'ha decidit entrenar tots els models amb els dos tipus de tokenitzadors diponibles i posteriorment analitzar els resultats (apartat 2.5).

Per la tokenització, independentment del tokenitzador, s'han considerat els textos incloent-hi les majúscules corresponents, ja que sovint són indicadors que ajuden a tokenitzar millor els textos. No obstant, a l'hora de guardar les dades s'han guardat amb i sense majúscules per tal de posteriorment comprovar si hi tenen influència.

2.4 Entrenament

Tots els models a entrenar seran de tipus Word2Vec [2], obtingut mitjançant la llibreria `gensim`. Per l'entrenament, s'han considerat múltiples paràmetres:

- `sg=1` *vs* `sg=0` (Skip-gram *vs* CBOW).
- `window` (mida de finestra).
- `min_count` (nombre mínim d'aparicions per considerar una paraula en el vocabulari).
- `vector_size` (mida dels vectors).

A més, com ja s'ha mencionat, també es volen comparar els tokenitzadors, la consideració de majúscules i la mida de les dades d'entrenament.

La metodologia a seguir serà la següent:

1. Entrenar tots els models amb totes les mides de dataset, `vector_size` 100, `window=5`, `min_count=5`, tant amb Skip-gram com amb CBOW i per tots dos tipus de tokenitzador. Escollir el millor model de cada tokenitzador i aplicar-los el següent pas.
2. Comparar els millors models del pas anterior amb models iguals, però provant amb el paràmetre `window=10`. Escollir de nou el millor model per cada tokenitzador.
3. Provar si els millors models del pas anterior milloren utilitzant les dades que inclouen majúscules. Escollir de nou el millor model per cada tokenitzador.
4. Finalment, comprovar si es poden millorar els models utilitzant `vector_size=300`. Analitzar els resultats per determinar els millors models finals.

2.5 Avaluació i resultats

Per l'avaluació s'han tingut en compte els següents mètodes:

- **t-SNE**: *t-distributed Stochastic Neighbor Embedding* [3] és un mètode estadístic que permet visualitzar la projecció de vector d'alta dimensionalitat en un espai de tan sols dues o tres dimensions. D'aquesta manera, es pot comprovar si les paraules amb significats similars queden agrupades en zones similars de la projecció generada amb *t-SNE*, indicant que el embeddings del model són de bona qualitat.
- **Similituds entre paraules**: A més del *t-SNE*, també es poden comprovar manualment les paraules més similars (properes en distància euclidiana) a una certa paraula. Per tant, es comprovaran quines són les paraules més similars a certes paraules donades per evaluar si els resultats són coherents o no.
- **Correlació de Pearson, Spearman i OOV**: Aquest és el principal mètode d'avaluació que s'ha utilitzat i tracta de comparar la correlació de certes paraules en el model amb la correlació preestablerta en un fitxer de similituds, concretament les de *wordsim353* (originalment en anglès, però s'han traduït al català per aquesta pràctica mitjançant Google Translator i corregint-ho manualment), obtingut a [4]. A més, pels models que inclouen o no majúscules, s'utilitza una versió amb o sense majúscules, respectivament, per tal de que l'avaluació sigui més justa i coherent.

  El valor de la correlació de Pearson mesura la relació lineal entre la similitud del model i la preestablerta en el test mitjançant un valor en l'interval $[-1, 1]$; on valors negatius representen les correlacions negatives, valors positius les positives i 0 indica que no hi ha correlació.

  Per altra banda, el coeficient de correlació de Spearman representa la correlació monotònica entre les similituds del model i les del test, també amb valors en l'interval $[-1, 1]$. En aquest cas, es basa en rangs i no assumeix cap mena de relació lineal.

  Cal mencionar que aquestes dues correlacions també proporcionen un p-value, que es comprovarà que sempre sigui menor que 0.05 per tal d'assegurar que la correlació detectada sigui estadísticament significant.

  Addicionalment, OOV (de l'anglès *Out Of Vocabulary*) indica el percentatge de paraules que no han estat reconegudes pel model (que no apareixen en el vocabulari). Per tant, aquest és l'únic valor que és millor a com més petit sigui.

  Finalment, s'ha afegit una última mètrica anomenada Avg. Statistic, que representa la mitjana aritmètica entre la correlació de Pearson i la de Spearman. Aquesta serà la principal mètrica a considerar per avaluar els diferents models i determinar quins són millors.

2.5.1 Primer pas

Com ja s'ha mencionat prèviament, el primer pas ha consistit en entrenar tots els models amb totes les mides de dataset, vectors de mida 100, `window=5`, `min_count=5`, tant amb Skip-gram com amb CBOW i per tots dos tipus de tokenitzador. Una vegada entrenats els models, s'ha executat l'arxiu *evaluate_models.py* per obtenir totes les mètriques i generar gràfics que permetin visualitzar fàcilment els resultats.

En la taula 1 es poden veure els resultats de tots els models entrenats, que també es poden veure d'una forma més comprensible en les figures 1, 2, 3 i 4.

Cal destacar que el model entrenat amb tot el dataset i tokenitzat amb spaCy no ha pogut ser entrenat per errors de memòria després d'estar 8:30h seguides tokenitzant les dades, tot i haver-se fet per lots (batches) i en un ordinador amb CPU Intel i9 9900K, GPU Nvidia RTX 2080ti (amb CUDA) i 32GB de memòria RAM a 3600MHz.

Model	**Avg. Stat.**	**Pearson**	**Spearman**	**OOV (%)**
sg_spacy_cat_gc_1000mb	**0.5445**	**0.5314**	**0.5576**	1.3575
sg_nltk_cat_gc_1000mb	0.5233	0.5084	0.5383	**0.4525**
sg_nltk_cat_gc_maxmb	0.5092	0.4941	0.5243	**0.4525**
sg_spacy_cat_gc_250mb	0.4937	0.4786	0.5089	2.2624
sg_spacy_cat_gc_500mb	0.4928	0.4739	0.5116	1.3575
sg_nltk_cat_gc_500mb	0.4927	0.4736	0.5118	1.3575
sg_spacy_cat_gc_50mb	0.4505	0.4457	0.4552	5.8824
sg_nltk_cat_gc_250mb	0.4445	0.4300	0.4514	1.3575
sg_nltk_cat_gc_100mb	0.4172	0.4197	0.4146	2.2624
cbow_spacy_cat_gc_500mb	0.4113	0.3933	0.4292	1.3575
cbow_nltk_cat_gc_maxmb	0.4045	0.3910	0.4181	0.4525
cbow_nltk_cat_gc_1000mb	0.4024	0.3897	0.4152	0.4525
sg_nltk_cat_gc_50mb	0.3969	0.3950	0.3988	6.3348
cbow_nltk_cat_gc_500mb	0.3893	0.3758	0.4028	1.3575
cbow_spacy_cat_gc_250mb	0.3735	0.3586	0.3884	2.2624
cbow_nltk_cat_gc_250mb	0.3450	0.3335	0.3565	1.8100
cbow_spacy_cat_gc_100mb	0.3158	0.3051	0.3265	3.1674
cbow_nltk_cat_gc_100mb	0.2972	0.2885	0.3059	2.2624
cbow_nltk_cat_gc_50mb	0.2774	0.2568	0.2980	6.3348
cbow_spacy_cat_gc_50mb	0.2750	0.2593	0.2907	5.8824

Taula 1: Resultats de les avaluacions de tots els models, ordenats descendentment segons el valor d'Avg. Stat. (mitjana aritmètica entre Pearson i Spearman)

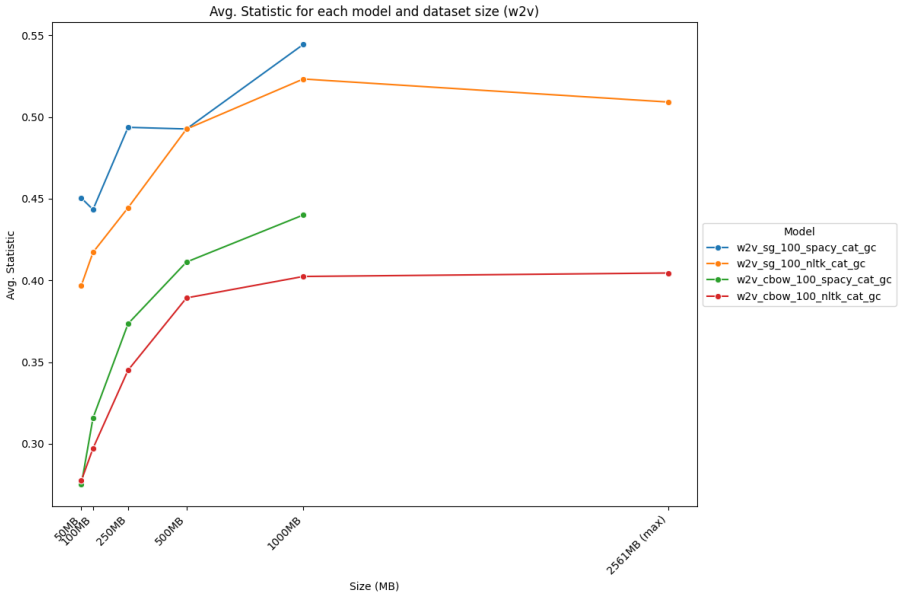

Figura 1: Evolució de la mètrica Avg. Stat. en funció del model i la mida de les dades d'entrenament

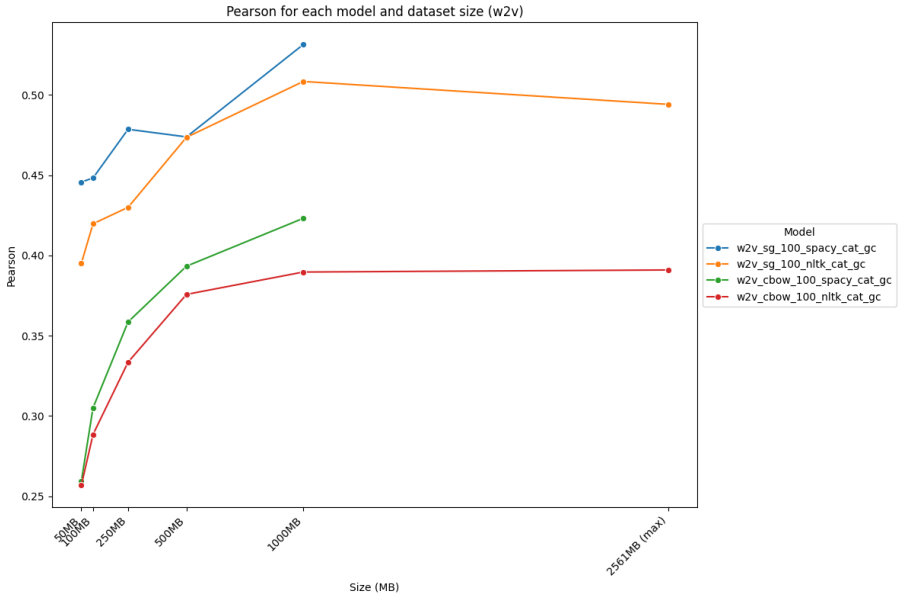

Figura 2: Evolució de la mètrica Pearson en funció del model i la mida de les dades d'entrenament

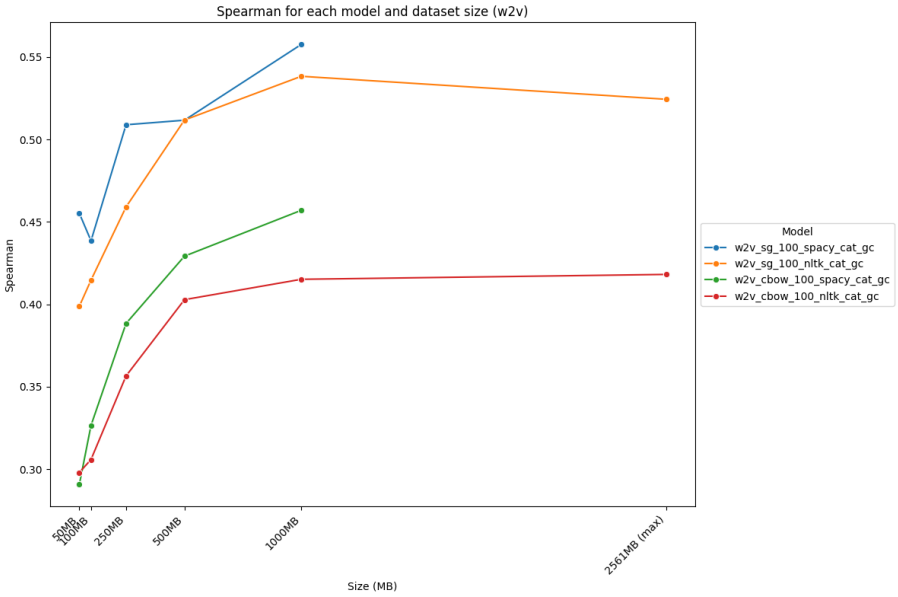

Figura 3: Evolució de la mètrica Spearman en funció del model i la mida de les dades d'entrenament

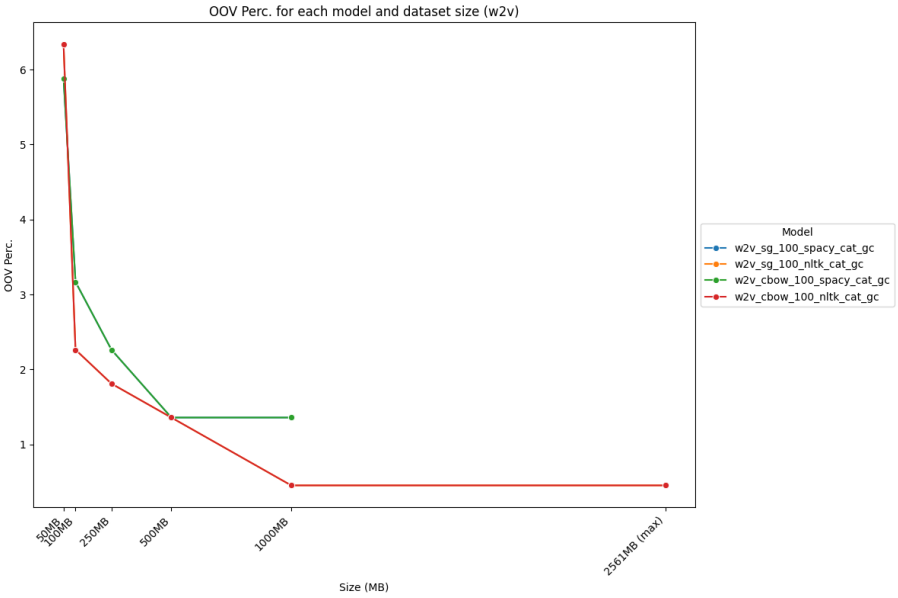

Figura 4: Evolució de la mètrica OOV en funció del model i la mida de les dades d'entrenament. Els resultats queden dividits entre els de spaCy (verd) i NLTK (vermell), ja que OOV només depèn del tokenitzador utilitzat en les dades d'entrenament

Tot i que en les prèvies figures 1, 2, 3 i 4 ja es pot apreciar clarament com spaCy és generalment millor que NLTK i Skip-gram supera àmpliament a CBOW, en les figures 5 i 6 es poden veure millor les diferències.

Per tant, basant-nos en les prèvies figures i en les figures 7, 8, 9 i 10 (que mostren ordenadament els resultats de tots els models), s'han escollit els següents models per realitzar el següent pas:

- **SpaCy**: Model Word2Vec amb Skip-gram i entrenat amb 1000MB de dades. Evidentment, amb paràmetres `vector_size=100`, `window=5` i `min_count=5`.
- **NLTK**: El mateix model que per spaCy; és a dir, el model Word2Vec amb Skip-gram i entrenat amb 1000MB de dades. Evidentment, amb paràmetres `vector_size=100`, `window=5` i `min_count=5`.

Cal destacar com el millor model de NLTK ha estat el de 1000MB i no pas el de tot el dataset (2561MB). Això pot ser que sigui degut a que pel model complet caldria un `min_count` per eliminar soroll,

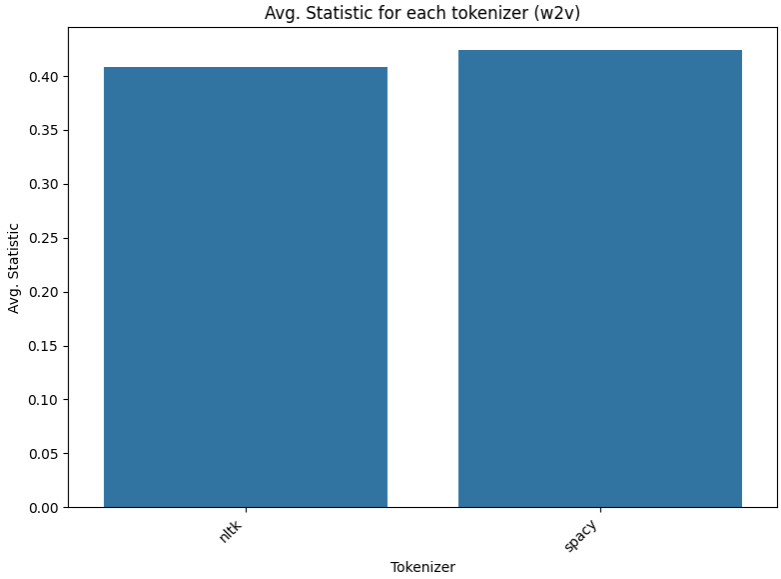

Figura 5: Mitjana de la mètrica Avg. Stat. dels models amb NLTK *vs* amb spaCy

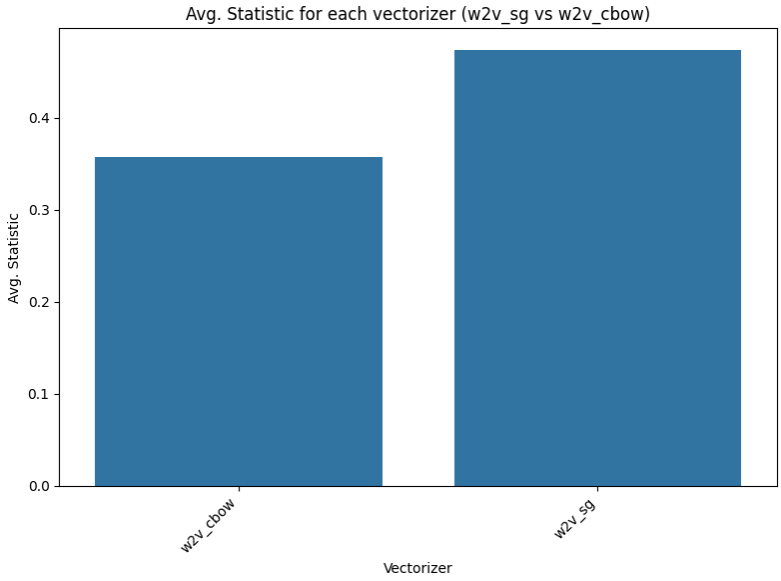

Figura 6: Mitjana de la mètrica Avg. Stat. dels models amb CBOW *vs* amb Skip-gram

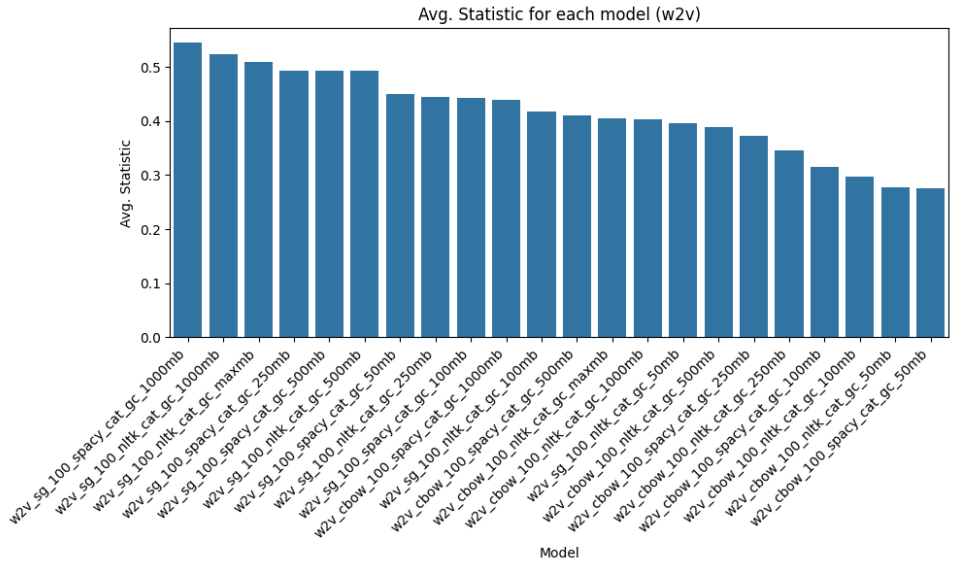

Figura 7: Mètrica Avg. Stat. per tots els models entrenats

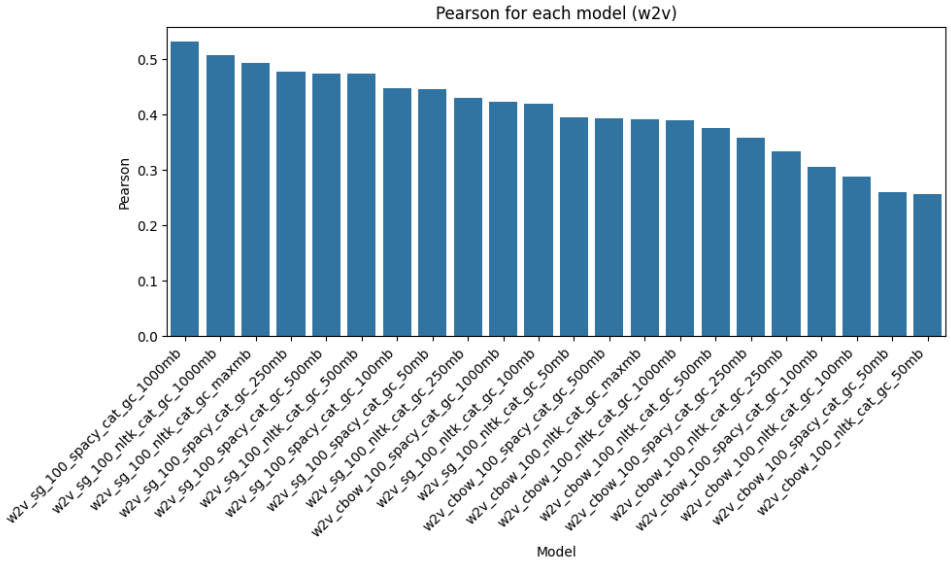

Figura 8: Mètrica Pearson per tots els models entrenats

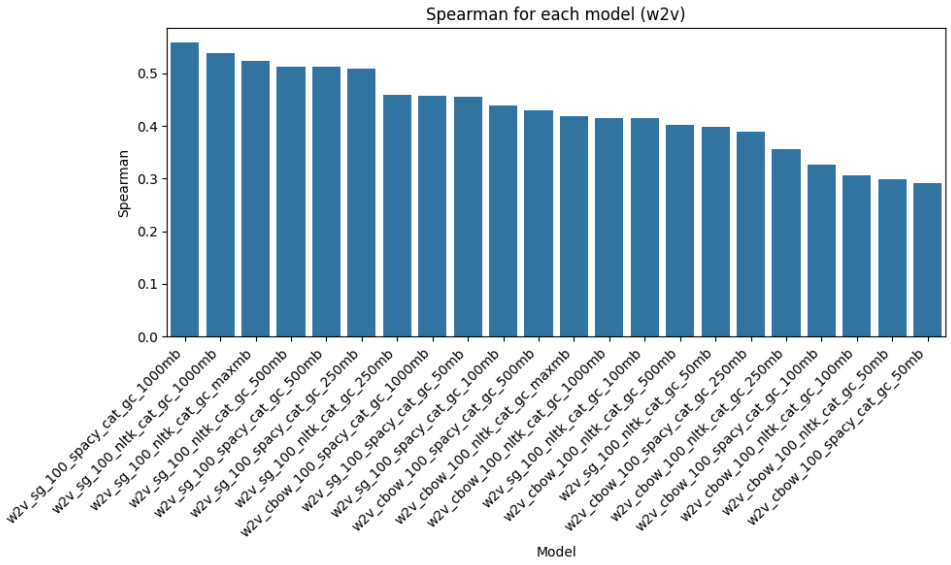

Figura 9: Mètrica Spearman per tots els models entrenats

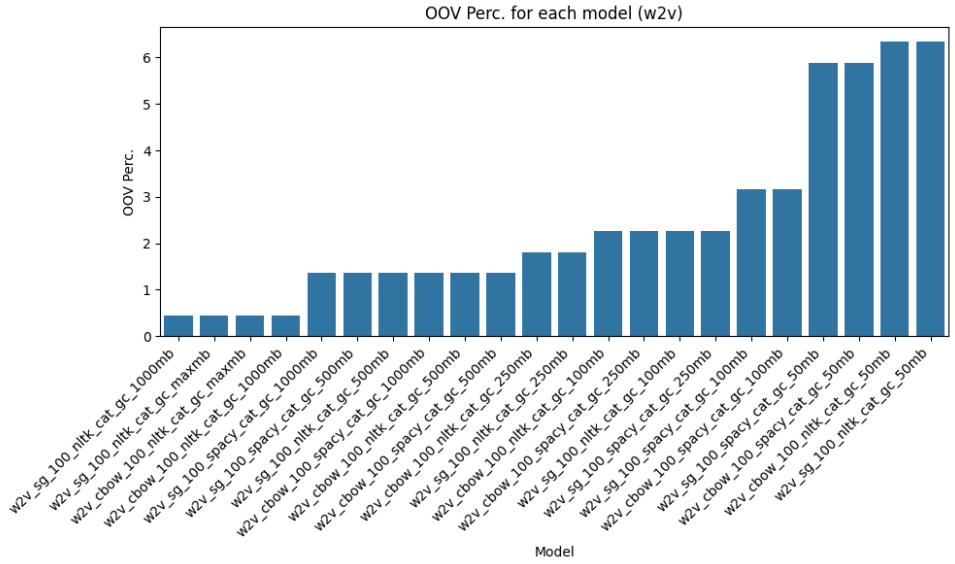

Figura 10: Mètrica OOV per tots els models entrenats

2.5.2 Segon pas

Una vegada escollits els millors models per cada un dels tokenitzadors, s'han entrenat models iguals als escollits, però amb el paràmetre `window=10`. La comparació dels nous resultats canvis amb els previs es poden veure en la taula 2.

Model	Avg. Stat.	Pearson	Spearman	OOV (%)
sg_win10_nltk_cat_gc_1000mb	**0.5643**	**0.5523**	**0.5763**	**0.4525**
sg_win10_spacy_cat_gc_1000mb	0.5601	0.5491	0.5711	1.3575
sg_spacy_cat_gc_1000mb	0.5445	0.5314	0.5576	1.3575
sg_nltk_cat_gc_1000mb	0.5233	0.5084	0.5383	**0.4525**

Taula 2: Resultats de les avaluacions dels previs millors models i dels nous models (amb `window=10`), ordenats descendentment segons el valor d'Avg. Stat. (mitjana aritmètica entre Pearson i Spearman)

En tots dos casos es pot veure com el paràmetre `window=10` ha millorat els resultats previs (amb `window=5`). Això era d'esperar, ja que ara cada paraula té molt més "context" sobre les paraules que es troben en el seu entorn.

Per altra banda, tot i que els resultats són molt similars per tots dos tokenitzadors, cal destacar com ara el millor model és és el que utilitza dades tokenitzades amb NLTK i no pas amb spaCy. Per veure si això passa de forma general o es tracta només d'aquest cap en concret, en la figura 2

es pot veure la diferència d'Avg. Stat. entre els models de Skip-gram amb spaCy i NLTK en funció de les diferents mides del dataset i per `window=5` (per defecte) i `window=10`.

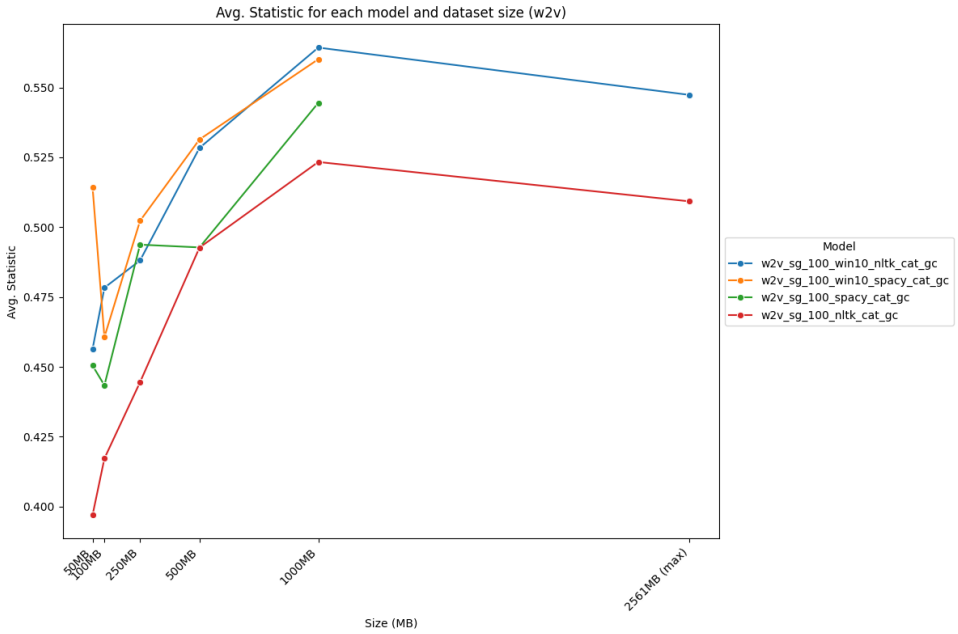

Figura 11: Mètrica Avg. Stat. per tots els models en funció de la mida del dataset d'entrenament

Ara està més clar que justament el cas mencionat anteriorment és dels pocs casos on NLTK supera a spaCy (i per molt poca diferència). Això també es dona pel dataset de 100MB, on la diferència és més clara.

A més, també es pot mencionar com, per NLTK, el model amb 1000MB segueix donant millors resultats que el de mida màxima (2561MB), segurament degut a que no s'ha augmentat el paràmetre `min_count`.

Una vegada analitzats aquests nous resultats, és clar que les nous millors models pel següent pas són les que tenen `window=10`, de manera que s'han escollit aquests.

2.5.3 Tercer pas

En aquest pas s'han provat els models escollits en el pas anterior, però amb dades que contenen majúscules. A l'hora d'avaluar aquestes dades, s'ha utilitzat el test que també conté les paraules originals (que no s'han convertit a minúscules).

Model	Avg. Stat.	Pearson	Spearman	OOV (%)
sg_win10_nltk_cat_gc_1000mb	**0.5643**	**0.5523**	**0.5763**	**0.4525**
sg_win10_spacy_cat_gc_1000mb	0.5601	0.5491	0.5711	1.3575
sg_win10_nltk_caps_cat_gc_1000mb	0.5225	0.5131	0.5320	0.9050
sg_win10_spacy_caps_cat_gc_1000mb	0.5222	0.5123	0.5322	1.3575

Taula 3: Resultats de les avaluacions dels previs millors models i dels nous models (incloent majúscules), ordenats descendentment segons el valor d'Avg. Stat. (mitjana aritmètica entre Pearson i Spearman)

En aquest cas es segueix veient com NLTK supera a spaCy en la mida 1000MB, a més de que les majúscules baixen el rendiment del model. A més, es pot veure que en spaCy s'ha obtingut el mateix resultat de OOV utilitzant majúscules que no utilitzant-les, però en NLTK s'ha vist un empitjorament considerable (del 100%). Per tant, pel següent pas ens quedem amb els mateixos models que hem determinat prèviament.

2.5.4 Quart pas

Per intentar millorar encara més els models, s'ha provat d'entrenar els millors models del pas prèvi, però utilitzant `vector_size=300`. En la taula 4 es pot veure la comparació dels nous models amb els anteriors.

Model	Avg. Stat.	Pearson	Spearman	OOV (%)
sg_300_win10_nltk_cat_gc_1000mb	**0.5884**	**0.5733**	**0.6036**	**0.4525**
sg_300_win10_spacy_cat_gc_1000mb	0.5844	0.5693	0.5995	1.3575
sg_win10_nltk_cat_gc_1000mb	0.5643	0.5523	0.5763	**0.4525**
sg_win10_spacy_cat_gc_1000mb	0.5601	0.5491	0.5711	1.3575

Taula 4: Resultats de les avaluacions dels previs millors models i dels nous models (amb `vector_size=300`), ordenats descendentment segons el valor d'Avg. Stat. (mitjana aritmètica entre Pearson i Spearman)

Els resultats mostren que la mida de vector ha millorat bastant els resultats previs, arribant per primera vegada a tenir una mètrica superior a 0.6 (Spearman en aquest cas). També es pot apreciar com NLTK segueix sent millor que spaCy en aquesta mida de dataset (1000MB), de manera que el millor model final és Word2Vec amb Skip-gram, tokenització amb NLTK, mida de dataset de 1000MB, `vector_size=300`, `window=10` i `min_count=5`.

2.5.5 Model final

Una vegada ja s'ha determinat el millor model, també s'ha fet un anàlisi amb *t-SNE* i un anàlisi de similituds amb certs exemples de paraules per tal de comprovar que realment té un bon rendiment.

2.5.5.1 t-SNE

Per realitzar l'anàlisi amb *t-SNE* s'ha realitzat amb el paràmetre `perplexity=5` per tal de que es representin millor les estructures locals, mostrant de manera més agrupada les paraules similars. Inicialment, s'ha provat amb les 100 paraules més freqüents del vocabulari i els resultats es poden veure en la figura 12.

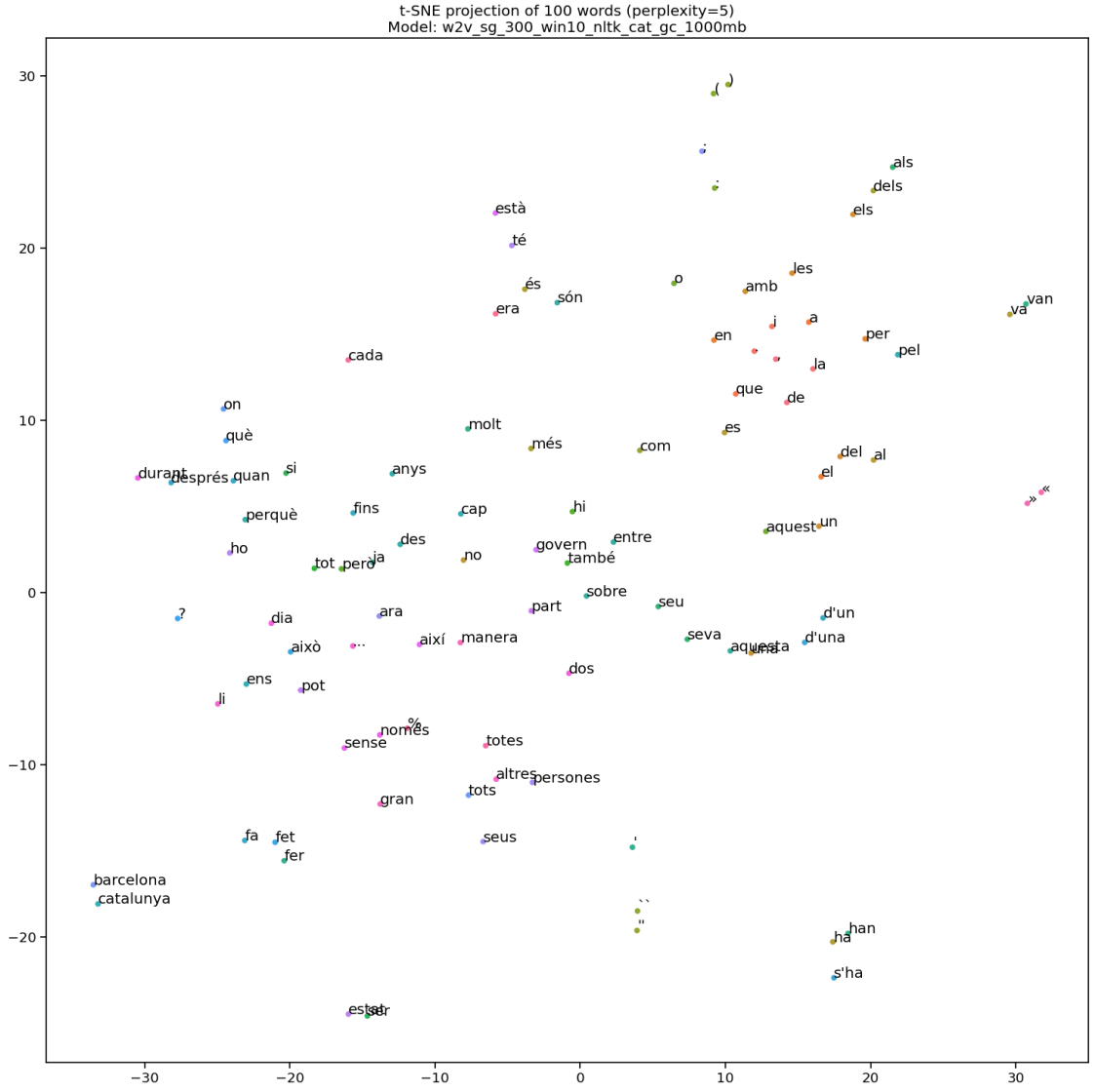

Figura 12: Projecció mitjançant *t-SNE* dels vectors de les 100 paraules més freqüents en el dataset

Es pot apreciar com certs grups de caràcters o paraules, com ara "(" amb ")", "catalunya" amb "barcelona", "va" amb "van", "¡¡" amb "¿¿", etc. Aquests grups semblen ser correctes, però la majoria de paraules d'aquesta projecció no tenen gaire significat o són directament stopwords.

Per tal de realitzar un *t-SNE* amb millors paraules, s'ha creat un vocabulari concret amb diversos grups de paraules que, a priori, és de sentit comú pensar que haurien d'estar agrupades. Aquests grups són els següents:

- **Sports**: futbol, tenis, hoquei, rugbi, atletisme, ciclisme, basquet, handbol, volei, esquí.
- **Politics**: parlament, govern, president, alcalde, política, eleccions, partit, democràcia.
- **Economy**: economia, empresa, mercat, comerç, indústria, finances, negoci, capital, consum, vendes.
- **Nature**: muntanya, riu, arbres, flors, paisatge, pedra, pirineus.
- **Tech**: ordinador, programari, software, hardware, mòbil, xarxa, internet, informàtica, ciberseguretat, ciberatac, dades, bigdata, digital.
- **Music**: piano, guitarra, violí, trompeta, orquestra, concert, òpera, flauta.
- **Food**: pizza, hamburguesa, pasta, paella, carn, amanida, canelons, postres, gelat, arròs.
- **Animals**: gos, gat, elefant, girafa, hipopòtam, ocell, lleó, ratolí, tigre, cavall, vaca, gallina.

Seguidament, s'han provat de fer projeccions d'alguns d'aquests grups de paraules mitjançant *t-SNE*. Per exemple, en la figura 13 es poden veure els resultats pels grups sports, politics, nature, tech i food. En aquest cas es veu clarament com les agrupacions són molt coherents i mostren també les similituds entre paraules de diferents grups. Per exemple, "esquí" és clarament l'esport més proper a "muntanya" i altres elements del grup nature. També, "partit" (referint-se a partit polític) és la paraula del grup politics més propera al grup sports, ja que també existeix el significat de "partit" de tenis, futbol, etc.

Per altra banda, en la figura 14 s'hi pot veure la projeccció amb la resta de grups que no s'havien inclòs en la figura 13. És interessant notar com "gallina" o "vaca", tot i pertànyer al grup d'animals, es troben molt propers a food, ja que són els únics animals que es consumeixen a Catalunya, mentre que els animals més salvatges es troben més propers al grup nature. També es pot notar com el grup politics és molt porper al grup economy, especialment a les paraules "finances" i "economia", que sovint es mencionen en política.

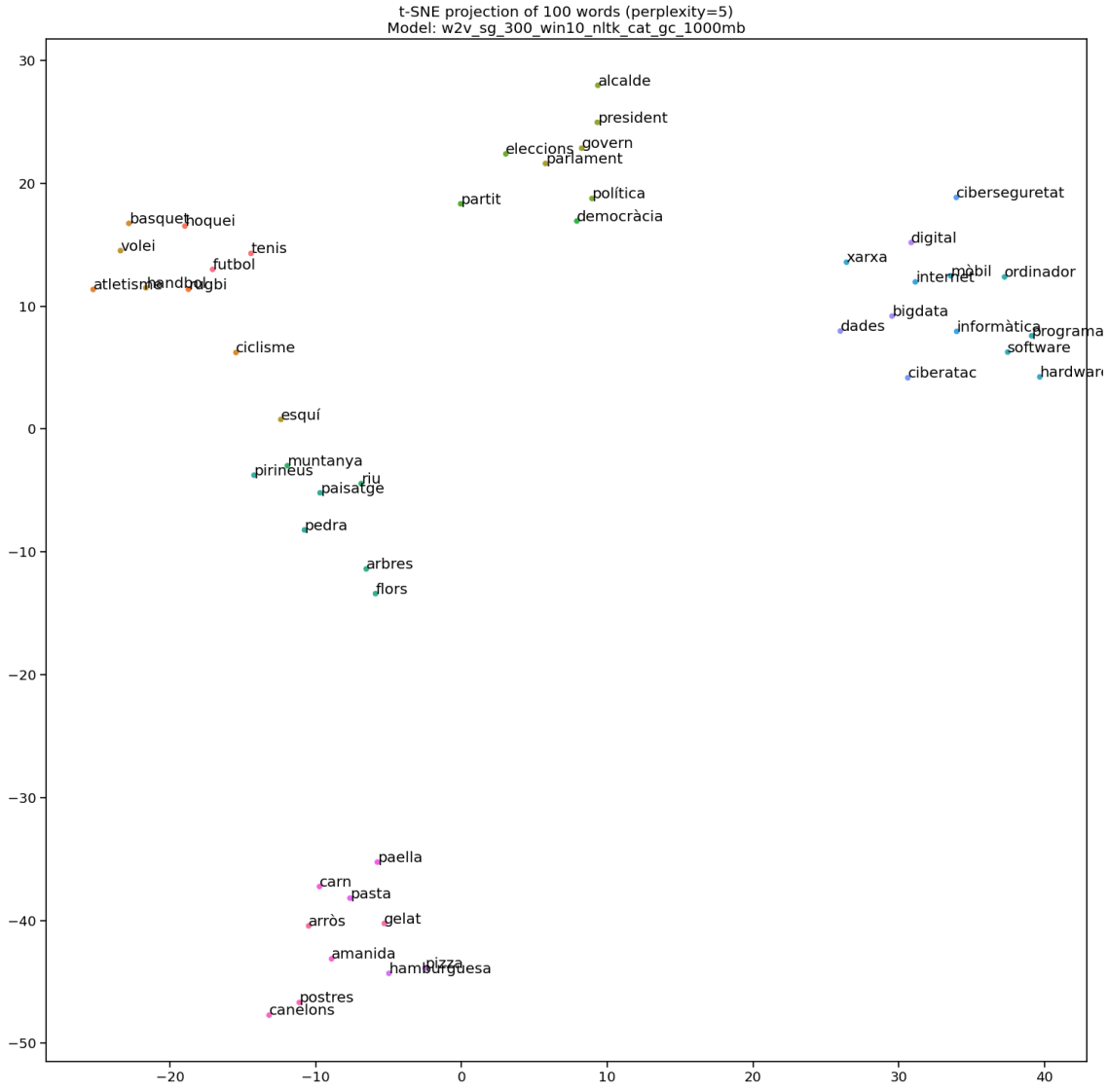

Figura 13: Projecció amb *t-SNE* dels vectors de les paraules dels grups sports, politics, nature, tech i food

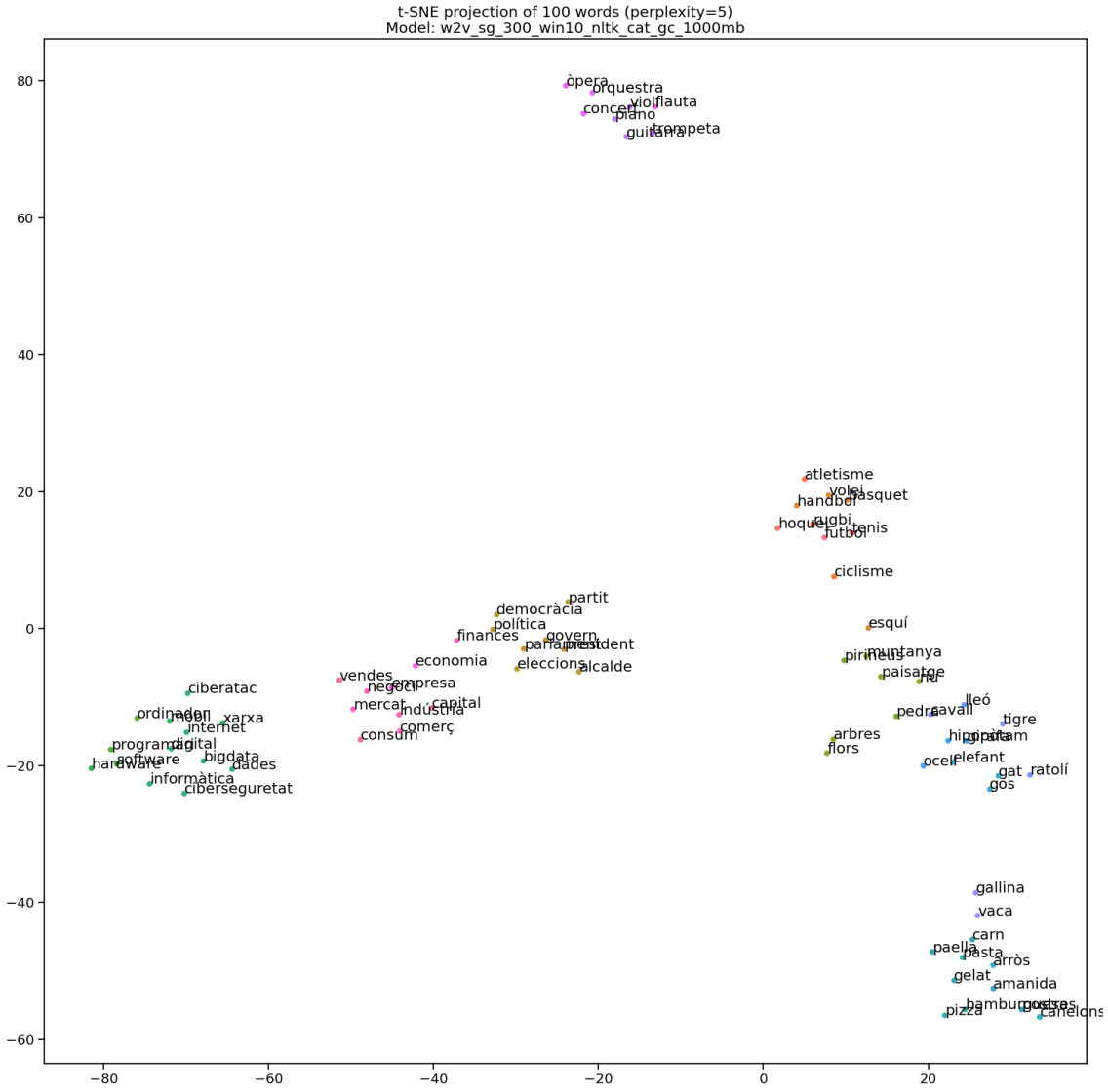

Figura 14: Projecció amb *t-SNE* dels vectors de les paraules de tots els grups

Tot això indica que els embeddings d'aquest model final són bastant bons i han aconseguir captar la similitud entre certes paraules.

Finalment, s'han realitzat les projeccions de les paraules "rei", "reina", "home", "dona", "gat", "gata", per tal de veure si hi ha certes direccions que indiquen el gènere i altres que indiquin si la paraula està més realacionada amb la naturalesa i els animals o amb els humans i la societat. Per fer-ho, s'ha establert `perplexity=4`, ja que només hi ha un total de 6 paraules, de manera

que es captaran patrons bastant generals/globals (valor de `perplexity` molt proper al nombre de paraules projectades). En la figura 15 es pot veure com clarament s'ha aconseguit trobar la projecció desitjada, deixant "gat" i "gata" en un costat, "home" i dona al mig, i "rei" i "reina" a l'altre costat. A més, "gat", "home" i "rei" es troben també en el costat oposat a "gata", "dona" i "reina". Per tant, aproximadament, la direcció vertical indica el gènere, mentre que la horitzontal (i també una mica el vertical) indica la pertanyença a la natura - societat.

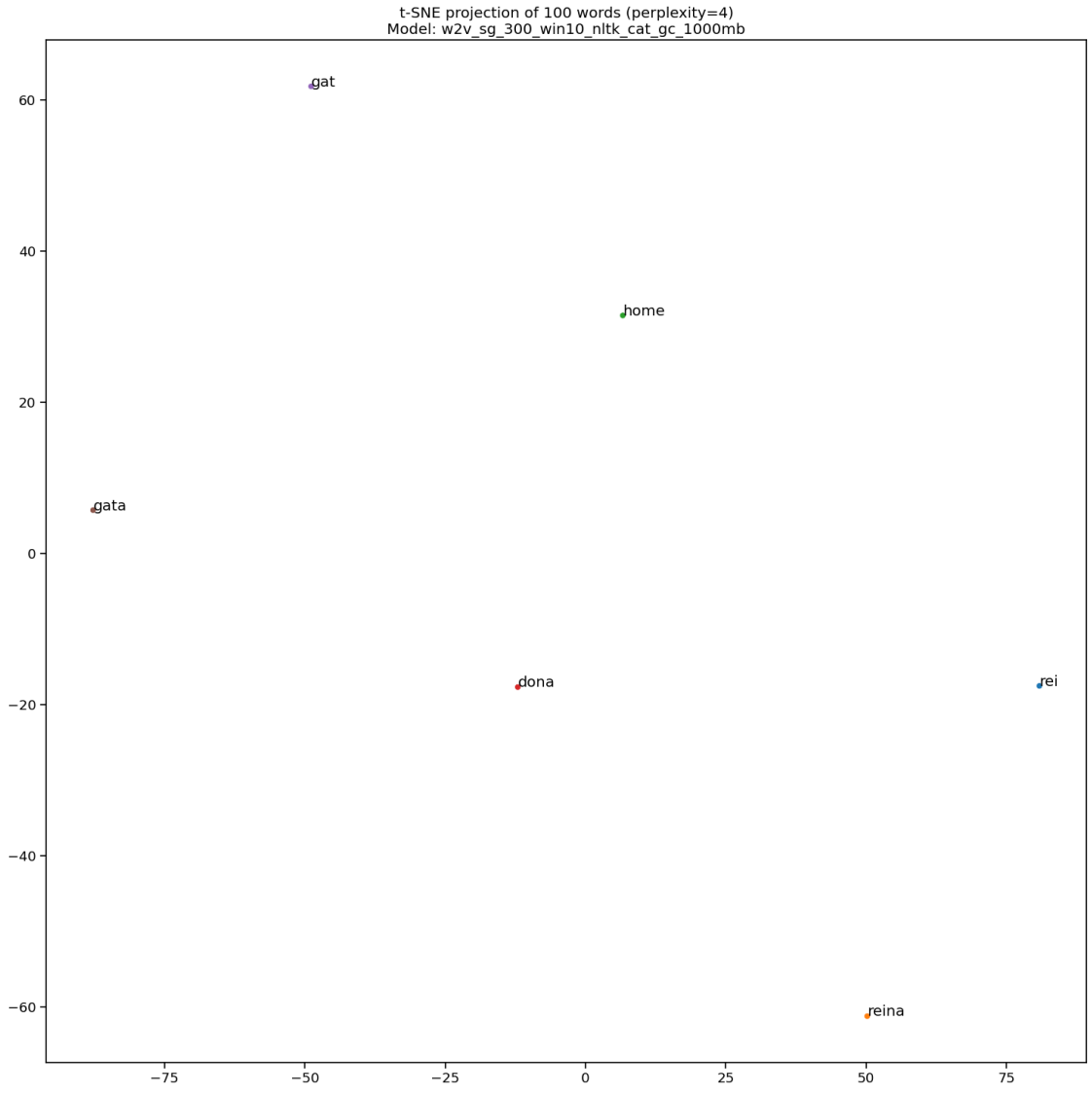

Figura 15: Projecció amb *t-SNE* dels vectors de les paraules "rei", "reina", "home", "dona", "gat", "gata"

2.5.5.2 Similituds entre paraules

Finalment, per acabar de veure si el model capta les similituds entre paraules correctament, s'han provat diversos exemples de paraules i s'han analitzat les 5 paraules més similars segons el model (amb el paràmetre `most_similar()`). Els resultats es poden veure en la taula 5 es mostren múltiples paraules amb les seves 5 paraules més similars, indicant la seva similitud de cosinus respecte la paraula original.

Paraula	Similar	Similitud (cosinus)
parlament	diputats	0.6935
	senat	0.6740
	parlamentària	0.6701
	diputat	0.6523
	parlamentari	0.6502
carretera	l'autopista	0.7641
	rotonda	0.7444
	c-35	0.7405
	b-124	0.7370
	n-152	0.7353
upc	politècnica	0.7596
	l'eseiaat	0.6730
	fib	0.6513
	d'enginyeria	0.6425
	upc-barcelonatech	0.6417
roba	sabates	0.7271
	sabatilles	0.6671
	calçat	0.6508
	mitjons	0.6407
	llenceria	0.6231
programari	software	0.8249
	maquinari	0.7641
	client/servidor	0.7197
	hardware	0.7099
	ofimàtic	0.6987

Taula 5: Taula de similituds de cosinus entre paraules amb el model final

Com s'ha pogut veure, els resultats són bastant positius. No obstant, es pot notar que algunes paraules no han estat correctament tokenizades (p. e.: "d'enginyeria" hauria de ser "enginyeria" i "l'autopista" hauria de ser "autopista"). Això és degut a que aquest model, com ja s'ha mencionat, utilitza NLTK en la tokenització, de manera que segueix les regles de l'anglès i no les del català.

Com que spaCy si que considera la gramàtica del català, també s'han comprovat les similituds de paraules per el seu model (el segon "millor" model, segons s'ha vist en l'apartat 2.5.4). En la taula 6 es poden veure els nous resultats i sembla que la qualitat general és similar, però en aquest cas no hi ha errors de tokenització.

Per tant, tot i que el model de NLTK donava resultats una mica millora (amb molt poca superioritat respecte a spaCy), la tokenització és bastant millor amb spaCy, de manera que els dos models es poden considerar com els "millors" i obtenen resultats bastant satisfactoris.

Paraula	**Similar**	**Similitud (cosinus)**
parlament	diputats	0.7041
	senat	0.6811
	diputat	0.6675
	parlamentària	0.6575
	parlamentari	0.6380
carretera	autopista	0.7431
	n-ii	0.7394
	c-35	0.7393
	b-124	0.7382
	n-2	0.7332
upc	politècnica	0.7511
	eseiaat	0.6368
	barcelonatech	0.6368
	catalunya·barcelonatech	0.6277
	enginyeria	0.6227
roba	sabates	0.7181
	calçat	0.6473
	vestir	0.6329
	jaqueta	0.6311
	sabatilles	0.6284
programari	software	0.8285
	maquinari	0.7652
	hardware	0.7371
	s.o.	0.7168
	linux	0.7091

Taula 6: Taula de similituds de cosinus entre paraules amb el model que utilitza tokenització amb spaCy

3 Segona part

3.1 Implementació

En aquest apartat, el codi principal es troba al fitxer *textsim_class.py*. Conté la implementació de la classe `TextSimilarity`, encarregada tant de preprocessar les dades com d'entrenar i avaluar els diferents models. L'arxiu *part2.ipynb* conté, doncs, les crides a objectes d'aquesta classe necessàries per dur a terme tots els experiments.

Els models de tensorflow, per temes de llegibilitat, estan definits en un tercer fitxer, *tensorflow_models.py*, amb una funció encarregada de construir i compilar cada un dels provats.

Per acabar de realitzar els gràfics, existeix el notebook *plot_textsim.ipynb*, on es defineixen les funcions usades pels plots que es comentaran més endavant.

3.2 Dataset

El conjunt de dades usat en aquest apartat és l'STS-ca [5], del projecte AINA. Conté 2073 parelles de frases d'entrenament, 500 de validació i 500 de test, anotades manualment. Per carregar-lo s'ha utilitzat el mètode load_dataset de la llibreria datasets de HuggingFace.

3.3 Models d'embeddings escollits

El primer pas era definir quins models d'incrustació de paraules (Word-Embeddings) calia utilitzar. Se'ns demanava provar One-Hot, models Word2Vec preeentrenats, spaCy (ca_core_news_md) [6], RoBERTa (tant CLS com la mitjana) [7] i finalment comparar amb RoBERTa finetuned [8].

En el cas del RoBERTa, Spacy no tenia una documentació clara sobre quin era el CLS, quin no o exactament com obtenir els embeddings. Per tant, s'ha optat per provar tres versions: un considerant el CLS com el primer token, un com l'últim, i finalment un llegint el model de HuggingFace.

El model One-Hot, per evitar tenir vectors de mida massa llarga, es va optar per quedar-se amb tan sols les 1000 paraules més importants (usant el mètode filter_extremes).

Per als models Word2Vec, vam optar per provar-ne diferents (d'aquesta manera també els podíem comparar): word2vec entrenat a l'apartat anterior amb tot el corpus (i vector size 300), així com els de 1000 , 500 i 100 que havien donat millors resultats; un altre word2vec de l'nlpl vector repository [9], concretament l'id 34; i el model de fasttext *cc.ca.300*.

Addicionalment, alguns models també requerien d'embeddings entrenables. Aquests s'han inicialitzat de manera aleatòria o bé utilitzant el nostre model complet o ccca300.

3.4 Models de tensorflow

Per tal de realitzar la tasca, també calia un model de regressió de similitud. Per poder comparar els resultats, s'ha optat per implementar-ne un total de 7 de diferents. D'entrada però, cal diferenciar entre els que utilitzen embeddings directament i els que tenen embeddings entrenables.

3.4.1 Models predefinits

Aquests reben com a entrada directament els embeddings del model. N'hi ha dos grups: uns que directament realitzen una concatenació dels vectors i uns altres que realitzen unes projeccions i calculen una distància (normalment de cosinus, o de cosinus normalitzada).

Dins el primer grup trobem el model 1, el 4 i el 7. En principi, estan ordenats per ordre de complexitat. El primer conté tan sols una capa de 64 neurones abans de la sortida. El següent, afegeix un parell de capes més: una de 128, i una de 200 abans, per tal de poder captar patrons més complexos. A més, incorpora regularització l2 per evitar realitzar overfitting (un dels problemes que vam trobar amb els models). Finalment, el set té moltes més neurones (2048, 1024, 256, 64) i també canvia la sortida, ja que en aquest cas realitza una sigmoide i la multiplica per 5 (per tal d'escalar els valors entre 0 i 5, ja que són els que tenim). També incorpora dropout com a tècnica de regularització.

El segon conjunt de models incorpora la similitud de cosinus d'alguna forma. D'entrada, les projeccions serveixen per donar més importància a determinades posicions dels vectors d'embeddings. El primer model (el 2), calcula aquest cosinus normalitzat amb l2, i escalat entre 0 i 5, i el retorna directament. El seu optimitzer és Adamax.

El número 3, afegeix després del càlcul de la distància una capa amb 16 neurones i la sigmoide que hem vist en el model 7. El 5 afegeix més neurones, així com Batch Normalization i el 6 és pràcticament idèntic al 5 però amb més dropout i regularització. Finalment, el 8 és lleugerament similar a aquests últims, però amb menys neurones ja que es va observar que obtenia millors resultats.

Comentar que als resultats (a *similarity_results.csv*) els nombres dels models són 1 menys.

3.4.2 Embeddings entrenables

Els models amb embeddings entrenables funcionen lleugerament diferent. D'entrada només hi ha 4 arquitectures diferents, i els altres consisteixen simplement en realitzar un condicional o un altre.

El primer model rep dos inputs (vectors d'índexs de paraules de les dues frases) i inicialitza la capa Embedding de keras com correspongui (amb pesos preentrenats o bé sense, de manera aleatòria). A continuació, realitza un GlobalAveragePooling1D, i ens trobem el condicional. En cas d'usar cosinus, simplement el calcula directament i el retorna. En cas contrari, abans de retornar-lo passa per una xarxa de 16 neurones. El segon model és pràcticament idèntic però incorpora un mecanisme d'atenció, on calcula vectors contextuals. Després d'això, el model és com l'anterior. Així, el model

3 parteix d'aquest 2 però després de la capa d'atenció incorpora un parell de Dense. Finalment, el 4 funciona totalment diferent i enlloc de calcular cosinus o altres mètriques concatena els vectors i els passa per una xarxa neuronal. Finalment, el model 8 és un intent d'imitar el model 8 de l'apartat anterior amb aquests inputs.

3.5 Preparació de les dades

La preparació de les dades en aquest apartat és més complexa. D'entrada cal simplement llegir el dataset, i es crea l'objecte diccionari i tfidf. A continuació, ve la part més complexa: cal passar les frases llegides a vectors. Com que cada una de les families de models funciona de forma diferent, existeix un mètode diferent que s'escull en funció de quin model d'embeddings li hem passat (així com la tècnica per ajuntar-los).

Els models word2vec tenen dues tècniques possibles: o utiltizar la mitjana o utilitzar tfidf. En el primer simplement s'agafen els embeddings de cada paraula i es realitza un numpy mean, mentre que en el segon es realitza un numpy average amb weights.

El model one-hot filtra el diccionari per evitar tenir vectors massa llargs, quedant-nos tan sols amb 1000 paraules. Llavors, el procés consisteix simplement en convertir un bag of words a 0 i 1. Per al model de Spacy mitjà, simplement li passem la frase i obtenim el .vector (de la frase completa).

Els models de RoBERTa són un pèl més complexos. D'entrada, es poden llegir de dos llocs: o de Spacy o bé de hugging face directament. En el primer cas, realitzem com en l'apartat anterior de Spacy però aquest cop obtenim `output._.trf_data.last_hidden_layer_state.data`. En teoria, en la posició -1 tenim el CLS i de la resta podem fer la mitjana. Com que no teniem massa clar si el CLS estava a la última posició o a la primera, ja que la documentació no és gens clara, hi ha un altre mètode per obtenir la posició 0.

En cas d'haver carregat RoBERTa de Huggingface, el CLS està a `last_hidden_state[:, 0, :]`, i la mitjana la realitzem directament amb un .mean.

Per últim, el model finetuned, del qual encara no hem parlat, es podria llegir com un model de RoBERTa, però aconsellem utiltizar-lo de manera independent als altres, realitzant un pipe per preparar les seqüències de frases i obtinguent els prediction["score] per analitzar-ne els resultats.

Comentar que, com que els models de RoBERTa tardàven bastant a realitzar aquesta preparació de les dades, hi ha l'opció de carregar-ne de preentrenades que permet estalviar-nos uns 8 minuts.

3.6 Entrenament

Per tal d'entrenar el model, un cop mapejades les dades, cal convertir-les a x i y (un mètode senzill), escollir un dels models plantejats anteriorment (tinguent en compte que pels d'embeddings entrenables, 0 és el model 1 sense cosinus, 1 és el mateix amb cosinus; 2 és el model 2 sense cosinus, 3 és el mateix amb cosinus...).

Aleshores, un cop s'ha definit el model (amb el mètode `define_model`) es pot entrenar. En el

nostre cas, hem optat per entrenar-los durant un màxim de 256 epochs, però amb EarlyStopping i ReduceOnPlateau, dos callbacks que ens permetran o bé millorar quan el model s'estanqui o bé parar-lo abans quan estigui moltes iteracions sense millorar.

Un cop entrenat el model, els resultats són simplement la correlació de pearson entre les prediccions i els valors reals.

Per provar tots els models, en el notebook *part2.ipynb* s'ha realitzat un bucle que itera per cada un d'ells i guarda els resultats en un csv.

En total hi ha 28 models d'embeddings diferents (els comentats anteriorment i les seves combinacions com *mean* i *tfidf*, o *cls* o mijtana en el cas dels models roberta). En el cas de l'inicialitzat de manera aleatòria, s'han realitzat dues execucions per tal de poder contrastar millor els resultats (i fora del bucle també se n'han realitzat més).

3.7 Resultats

Les mètriques d'avaluació escollides han sigut principalment, la correlació de pearson, tot i que també s'ha calculat la de spearman per si hi havia alguna relació no linear entre les prediccions i els valors reals.

Com que tenim molts resultats, és complicat valorar-los tots. Per aquest motiu, parlarem tan sols dels millors i estudiarem altres comportaments de manera més general (agrupant per famílies, models...)

D'entrada, podem analitzar els resultats de mitjana de tots els models per cada un dels embeddings diferents 16.

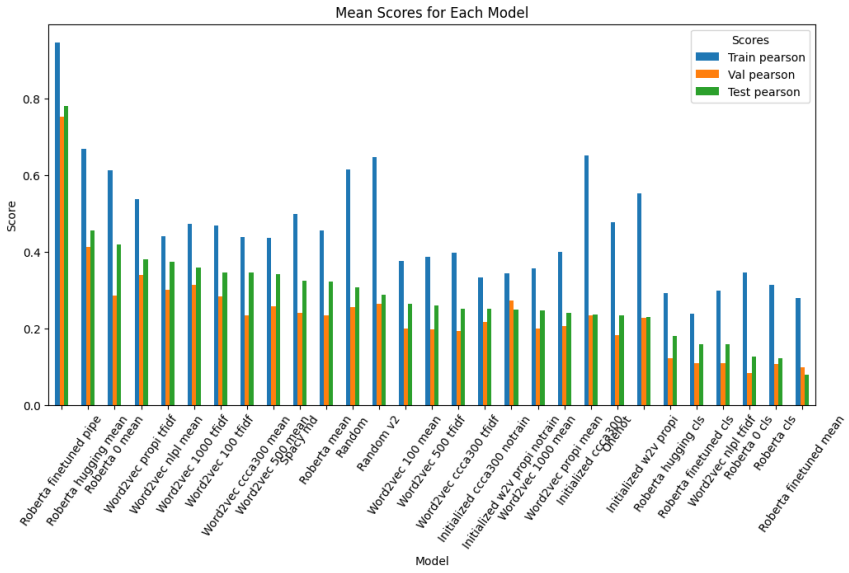

Figura 16: Mitjana dels models de tensorflow per cada un dels embeddings

Observem ja d'entrada com clarament, utilitzar el pipe del RoBERTa finetuned obté els millors resultats amb diferència. A continuació, trobem els models de RoBERTa utilitzant la mitjana i el CLS, llegint-los des de Huggingface. El següent és bastant sorprenent, ja que és un dels models Word2vec entrenats en l'apartat anterior, i precisament és el que contenia tan sols 100mB. A més, utilitza directament la mitjana, sense el tfidf. Per tancar el top 5 tenim un altre dels nostres models, aquest cop el que entrenava amb totes les dades, també usant la mitjana.

El model que obté pitjors resultats és llegir el RoBERTa finetuned com l'altre Roberta (és probable que no estiguem llegint el que toca realment), o bé utilitzar el CLS del roberta amb Spacy.

Tot i això, aquests resultats són la mitjana de tots els models de Tensorflow. Pot ser que en algun cas, uns embeddings siguin millors en un model concrets però molt dolents en un altre. Per evitar aquest problema, podem agafar el millor model de cada embedding 17.

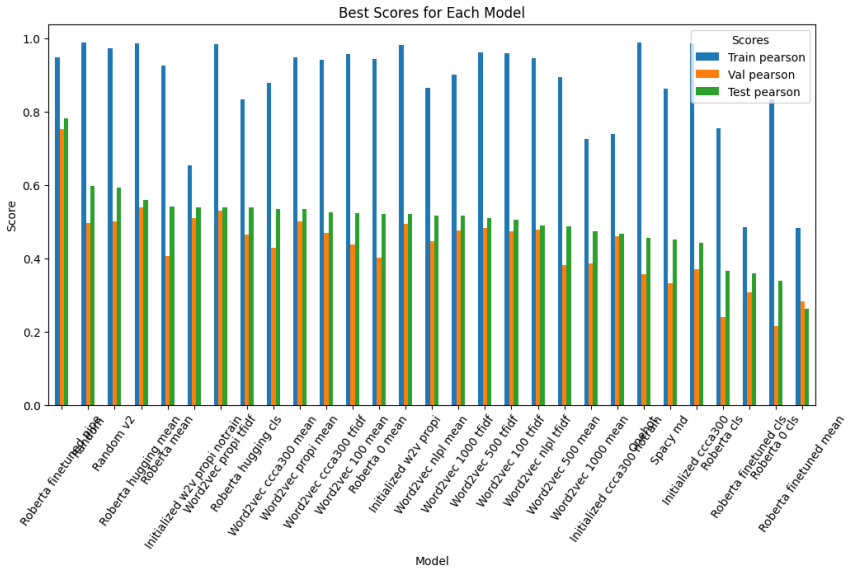

Figura 17: Millor model per cada un dels embeddings

Els resultats en aquest cas canvien lleugerament. El millor model continua sent el RoBERTa finetuned, però el segon és el d'embeddings entrenables inicialitzat amb Random. A més, aquest cop el word2vec propi és millor amb tfidf (i el de 100 perd posicions). Podem observar, enlloc dels plots, la taula directament 7.

Model	**Exec model**	**Pearson train**	**Pearson val**	**Pearson test**
Roberta finetuned pipe	7	0.9474	0.7523	0.7820
Random	1	0.9278	0.4956	0.5985
Random v2	1	0.9259	0.5005	0.5932
Roberta hugging mean	1	0.9628	0.4975	0.5597
Roberta hugging mean	5	0.5670	0.5394	0.5504
Roberta hugging mean	Baseline	0.5583	0.5336	0.5449
Roberta mean	1	0.9256	0.4071	0.5419
Initialized w2v propi notrain	0	0.6546	0.5099	0.5396
Word2vec propi tfidf	1	0.8311	0.4746	0.5395
Roberta hugging cls	Baseline	0.4670	0.4657	0.5391

Taula 7: Taula dels 10 millors model en funció dels resultats de test

És interessant veure també com dos dels models que apareixen ni tan sols entrenen, sinó que són el model baseline directament (distàncies de cosinus), encara que no sigui un model d'embeddings entrenat amb aquests (ja que és el RoBERTa base). Pot ser que, degut a l'alta dimensionalitat que tenen els seus embeddings (768), calcular les distàncies directament ja sigui efectiu. Tot i això, el model 1 sembla ser lleugerament millor.

Podem observar també l'altra mètrica, la de Spearman, que sol ser lleugerament més elevada 18.

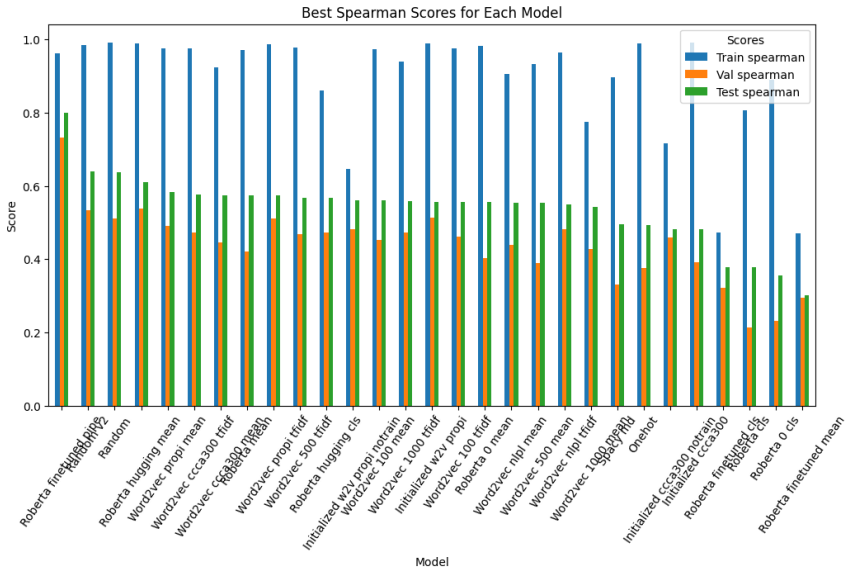

Figura 18: Millors models utilitzant Spearman

El top 10 continua sent bastant similar 8:

Model	Exec model	Spearman train	Spearman val	Spearman test
Roberta finetuned pipe	7	0.9614	0.7319	0.7993
Random v2	1	0.9470	0.5337	0.6403
Random	1	0.9478	0.5121	0.6367
Roberta hugging mean	5	0.5904	0.5378	0.6102
Roberta hugging mean	Baseline	0.5799	0.5329	0.6074
Roberta hugging mean	1	0.9853	0.4976	0.5936
Roberta hugging mean	4	0.5268	0.5283	0.5935
Word2vec propi mean	1	0.8497	0.4920	0.5831
Word2vec ccca300 tfidf	1	0.8256	0.4723	0.5771
Word2vec ccca300 mean	1	0.7823	0.4452	0.5748

Taula 8: Taula dels 10 millors model en funció dels Spearman de test

En aquest cas, els models de RoBERTta encara milloren més, situant-se sobre tots els altres.

Cal mencionar, però, que els resultats per la partició de validació canvien lleugerament. Els observem a la figura 19.

Aquest cop, els models de Word2Vec són bastant millors (excepte els de Roberta) que, per exemple,

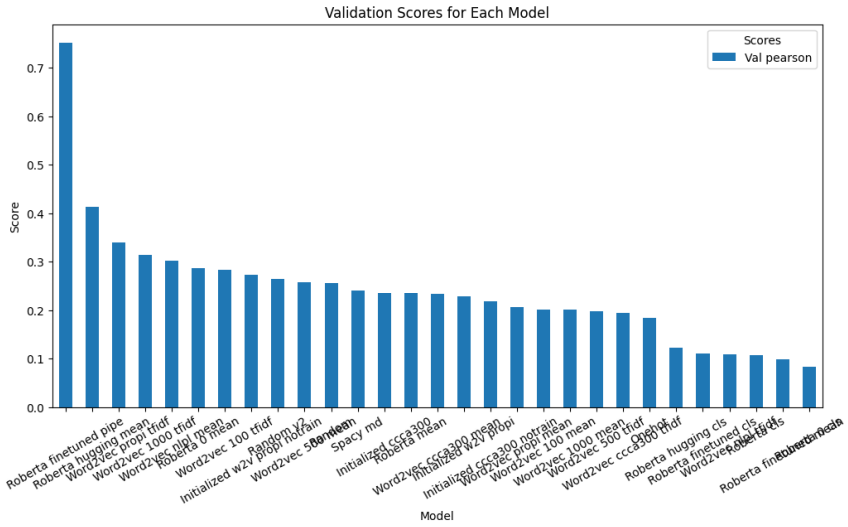

Figura 19: Models ordenats en funció dels resultats de validation

els d'embeddings entrenables. Els inicialitzats aleatòriament, són bastant inferiors.

Analitzant els diferents models de Tensorflow, detectem diversos patrons 20:

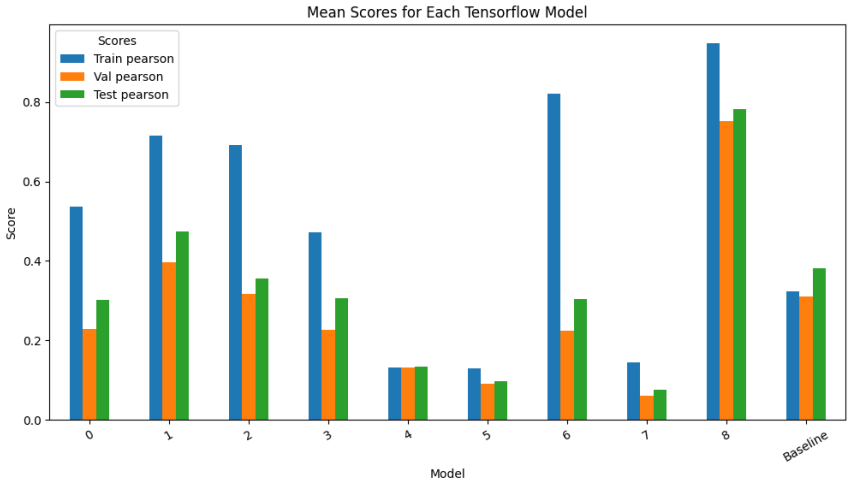

Figura 20: Mitjana dels valors de Pearson per cada model de Tensorflow

El model 8, no definit, és el pipe del RoBERTa finetuned, i ens serveix de punt de comparació. Dels altres, clarament el model 1 (explicat abans com a model 2) és el millor, seguit del model2. Sorprèn per exemple que el càlcul utilitzant el baseline directament sigui tan bo, situant-se en el tercer model. A més, realitza menys overfitting que els altres.

El model 6 té clarament un problema d'overfitting, mentre que el 4, 5 i 7 tenen resultats molt pobres. Aquest fet, si analitzem les dades, es deu a que en alguns casos, amb els models word2vec, les correlacions acaben sent negatives (de fins a -0.5). És el cas del model de roberta de HuggingFace, on s'observa que utiltizant la mitjana assoleix 0.55 i utilitzant el CLS -0.53.

Ens podem centrar en famílies de models d'embeddings. Per exemple, podem comparar els models de word2vec a la figura 21 utilitzant els resultats de Spearman.

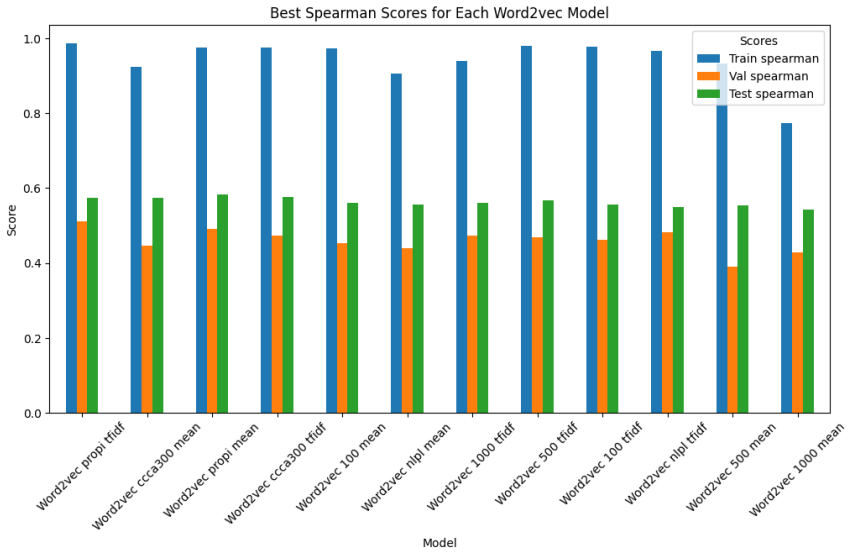

Figura 21: Millors scores de Spearman per cada model de Word2Vec

Observem com el millor model és el que hem creat, a l'apartat anterior, amb totes les dades i mida d'embedding 300. A més, el segon també té mida 300, sent aquest el de cc.ca.300. En canvi, l'altre preentrenat d'internet, del de nlpl, és dels pitjors models.

Sembla ser que els embeddings amb una dimensionalitat més elevada són millors, ja que precisament són els dos de 300 els que han obtingut uns resultats més elevats.

També ens podem centrar en els que utilitzaven embeddings d'una altra forma, en alguns casos entrenant-los. En total en tenim 6: dos de randoms, i quatre inicialitzats amb word2vec entrenats o no. Observem els seus resultats de Pearson .

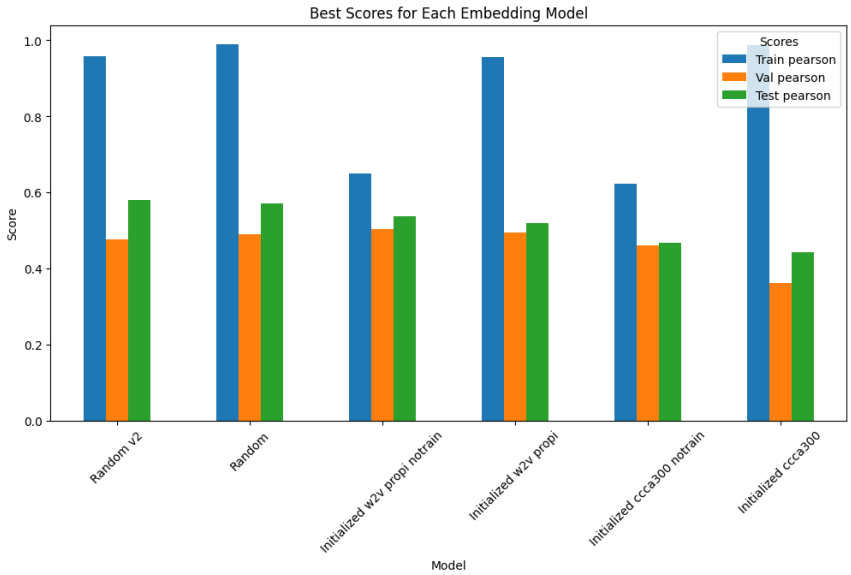

Figura 22: Millors resultats de Pearson pels models amb capa Embedding

Observem a 22 com els models inicialtizats aleatòriament obtenen millors resultats que els altres. Els que entrenen la capa d'Embedding tots solen realitzar un cert overfitting, però en el cas dels inicialitzats això provoca que el seu rendiment (tant en el test com en la validació) sigui menor. Sembla ser que, començar des d'uns pesos preentrenats, limita al model en certa manera i el fa estancar-se en un òptim local.

Els models complets es poden trobar a Github.

3.7.1 Interpretació

En general, analitzant els resultats, podem extreure algunes conclusions. D'entrada, sembla ser que els embeddings, per si sols, són bastant bons a l'hora de realitzar aquesta tasca, i realment no cal un xarxa neuronal molt complexa per obtenir uns bons resultats (de fet, com més senzilla pràcticament millor).

Addicionalment, amb word2vec hem vist com augmentar la dimensionalitat dels vectors permet captar les relacions semàntiques entre les frases millor. En general, però, sembla ser que els models que utilitzen encoders com RoBERTa, són més consistents i obtenen millors resultats, sobretot si es realitza finetuning.

Els embeddings entrenables també són una molt bona solució al problema, tot i que depenen bastant de la inicialització. De fet, hem observat com inicialitzar-los de forma aleatòria permet acabar obtenint resultats fins i tot millors a fer-ho amb un model preentrenat amb word2vec (tot i que evidentment, les primeres iteracions dels model són pitjors).

El model de Spacy mitjà no ha aparegut en cap comparativa, ja que és dels pitjors models. El mateix passa amb OneHot encoding, tot i que probablement amb una mida de vector més gran els resultats millorarien.

Hem observat com, en la majoria de casos, els models realitzen overfitting: és a dir, s'adapten molt bé a les dades d'entrenament, però no generalitzen gaire. A més, tot i que en alguns models teòricament realitzar la distància de cosinus directamnet no té gaire sentit (com en el cas del roberta no finetunejat amb CLS, el de spacy, o el one-hot encoding, ja que tècnicament no han estat entrenats amb aquesta mètrica i són embeddings contextuals), els resultats del baseline són bastant bons.

Tot i això, la correlació (excepte el model finetunejat) tant de Pearson com de Spearman no són gaire altres, quedant-se en els models bons entorn al 0.5 i en els dolents a 0.3. Spearman, no lineal, obté uns resultats lleugerament millors (0.64) tot i que molt similars.

3.8 Finetuning

Veient els bons resultats del RoBERTa finetuned, vam optar per intentar realitzar un finetuning propi. Per fer-ho, vam aprofitar els recursos de la llibreria sentence-transformers [10].

Vam optar per intentar realtizar fine-tunning en dos models: un amb una altra arquitectura, MPNet [11], que en principi és millor que RoBERTa (però del qual no disposàvem de cap model entrenat en català); i el model de roberta-base-ca-v2, que és en el que es basa l'altre model finetunejat que tenim. Aquestes arquitectures, en el fons estan basades les dues en BERT.

El dataset utilitzat és el mateix que hem fet servir fins aquest apartat.

El primer model ja està adaptat directament per fer-se servir com a sequence transformer. Al estar originalment en anglès, vam considerar que necessitaria més epochs per adaptar-se al català, i el vam entrenar amb 40. Com a mètrica d'avaluació, vam utilitzar la partició de validació de la base de dades, amb similitud de cosinus. La llibreria incorpora directament un EmbeddingSimilarityEvaluator, que facilita el procés.

Pel que fa al model de RoBERTa, per poder fer-lo servir com a sequence transformer primer el carregarem com a Transformer normal, i a continuació realitzarem un mean Pooling. La llibreria permet utilitzar també max Pooling, o el CLS token, però hem considerat que la mitjana (el valor per defecte) era una mètrica prou bona. Evidentment, es podria crear un altre model utiltizant per exemple el token CLS.

S'ha optat per entrenar-lo amb un learning-rate més baix i menys epochs, ja que en principi de per si ja s'adapta millor a les nostres dades.

Podem analitzar els resultats d'aquests dos models 23:

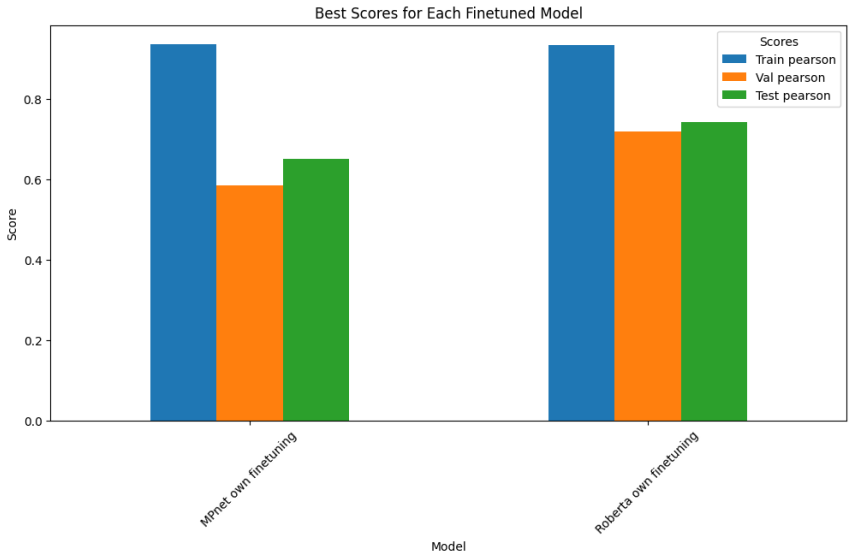

Figura 23: Resultats dels nostres models finetunejats

Observem com el model derivat de RoBERTa té pitjors resultats al l'entrenament, però tot i així obté millors resultats. La taula 9 ens mostra els valors exactes.

Model	**P train**	**P val**	**P test**	**S train**	**S val**	**S test**
Roberta finetuned	0.9474	0.7523	0.7820	0.9614	0.7319	0.7993
Roberta own	0.9350	0.7185	0.7429	0.9899	0.7312	0.7714
MPnet own	0.9370	0.5856	0.6501	0.9918	0.5855	0.6820

Taula 9: Taula comparativa dels models finetunejats

Aquests dos models nous, es situen en el top 2 i 3 dels models anterior. Aquest fet ens permet observar la gran millora en les capacitats del model que proporciona el finetuning, ja que fins i tot un model que no està entrenat pel català com és el MPnet és capaç de superar, per exemple, els embeddings de RoBERTa base. Comparant-los amb els resultats dels models anteriors, són molt superiors, i per tant molt millors en aquesta tasca.

Mencionar que el mètode de fine-tunning és diferent que el roberta-finetuned del projecte AINA, ja que en el nostre cas el que hem fet és acabar modificant els embeddings de la última capa (per carregar-lo amb transformers, el pipe de text classification no funcionarà correctament per aquesta tasca: caldrà agafar la mitjana dels embeddings de la última capa o bé el CLS). S'han entrenat utilitzant el mean Pooling (el default) dels embeddings, llavors tot i que el CLS funciona prou bé (0.7147 al test de RoBERTa) la idea és utilitzar la mitjana. També dir que l'avantatge de realitzar nosaltres mateixos el finetuning és que sabem exactament amb quins mètodes s'ha entrenat, amb quines dades, i podriem arribar a ajustar-lo en funció de què vulguem fer amb ell.

Els models estan disponibles a HuggingFace, concretament a finetuned-mpnet i finetuned-roberta.

4 Part opcional

En aquesta part opcional de la pràctica s'ha entrenat un model de classificació per predir la temàtica d'un text. Concretament, utilitzant el dataset de TeCla [12], on es troben 113376 articles classificats segons la temàtica de la que parlen. Aquest dataset permet realitzar dos prediccions diferents: una més general, amb 4 grans classes (temes), i una de molt més específica amb 53 classes diferents. A l'hora d'utilitzar aquest dataset, s'haura de considerar que, tan per 4 classes com per 53 classes, hi ha desbalanceig ens els valors de la variable objectiu.

La implementació d'aquest model s'ha fet a la classe `ClassificationWord2Vec`, definida en l'arxiu *classification_word2vec.py*. Totes les crides necessàries a aquesta classe per tal d'entrenar i avaluar el model es troben en l'arxiu *optional.ipynb*.

Totes les frases del dataset es tokenitzen mitjançant el mateix tokenitzador que el model d'embeddings ha utilitzat en les seves dades (NLTK o spaCy), ja que així es minimitzen les paraules que no es trobaran en el vocabulari del model.

Per l'entrenament d'aquest model cal un model d'embeddings per tal de representar les frases mitjançant vectors. El càlcul de la representació de les frases en vectors es fa mitjançant la mitjana dels vectors dels tokens que conformen la frase, ignorant els tokens que no apareixen en el vocabulari del model d'embeddings. Aquest model pot ser canviat mitjançant el paràmetre `w2v_model_name`.

Una vegada es tenen totes les dades perparades, el mètode `build_model()` permet definir un model de tipus xarxa neuronal amb diferentes capes, mides de capa, funcions d'activació, optimitzadors i learning rates. Aquesta xarxa neuronal rep com a input el vector que representa la frase a partir de la que s'ha de predir la seva temàtica, adaptant-se automàticament a la mida dels vectors del model d'embeddings.

4.1 Model de classificació amb 4 classes

Per ser coherents amb la resta d'aquesta pràctica, s'ha escollit utilitzar el millor model de la part 1 d'aquest projecte (definit en l'apartat 2.5.4 i avaluat en l'apartat 2.5.5). És a dir, amb `vector_size=300`, `window=10`, `min_count=5`, tokenitzador NLTK i 1000MB de dades d'entrenament del dataset Catalan General Crawling [1].

Pel model a entrenar s'ha decidir fixar l'optimitzador a Adam (amb `learning_rate=0.001`) i s'ha volgut provar inicialment sense capes ocultes; és a dir, només amb la capa d'entrada (mida 300, degut a la mida dels vectors de les frases) i la capa de sortida (4 neurones amb funció d'activació *softmax*). A més, per contrarestar el desbalanceig de les 4 classes de la variable objectiu, s'han assignat pesos (*weights*) a les diferents classes. Amb això, el model té un total de 1204 paràmetres.

Una vegada realitzat l'entrenament, en la figura 24 es poden veure les corbes d'aprenentatge en la partició d'entrenament i de validació. Sorprenentment, aquest simple model ha obtingut uns resultats molt bons, arribant a 0.9246 d'accuracy en la partició d'entrenament i a 0.9294 en la de validació.

Tot i que aquests resultats són molt bons, s'ha provat d'afegir alguna capa oculta en el model,

però la millora era ínfima (arribant com a molt a 0.94 d'accuracy en la validació) i els temps d'entrenament augmentaven. Per tant, es pot considerar que el model base plantejat és ja molt bo i no és necessari afegir-hi capes ocultes.

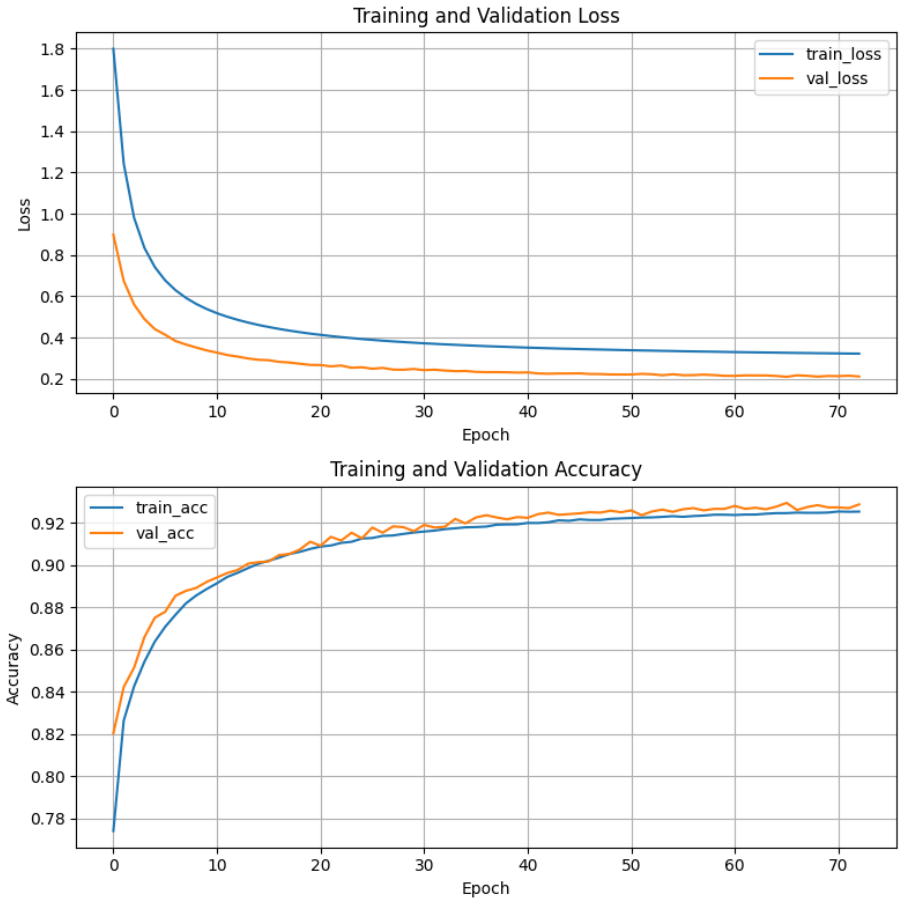

Figura 24: Corbes d'error (loss) i precisió (accuracy) en les particions de train i validation

Finalment, en l'avaluació en la partició de test s'ha obtingut un valor de 0.9277 d'accuracy. Tenint en compte que les classes es troben desbalancejades, també podem considerar que ha obtingut un valor de F1-Score (macro) que ha estat de 0.9277 en la partició de test.

En la figura 25 es pot veure la matriu de confusió de les prediccions del model en la partició de test, on els resultats són clarament molt bons.

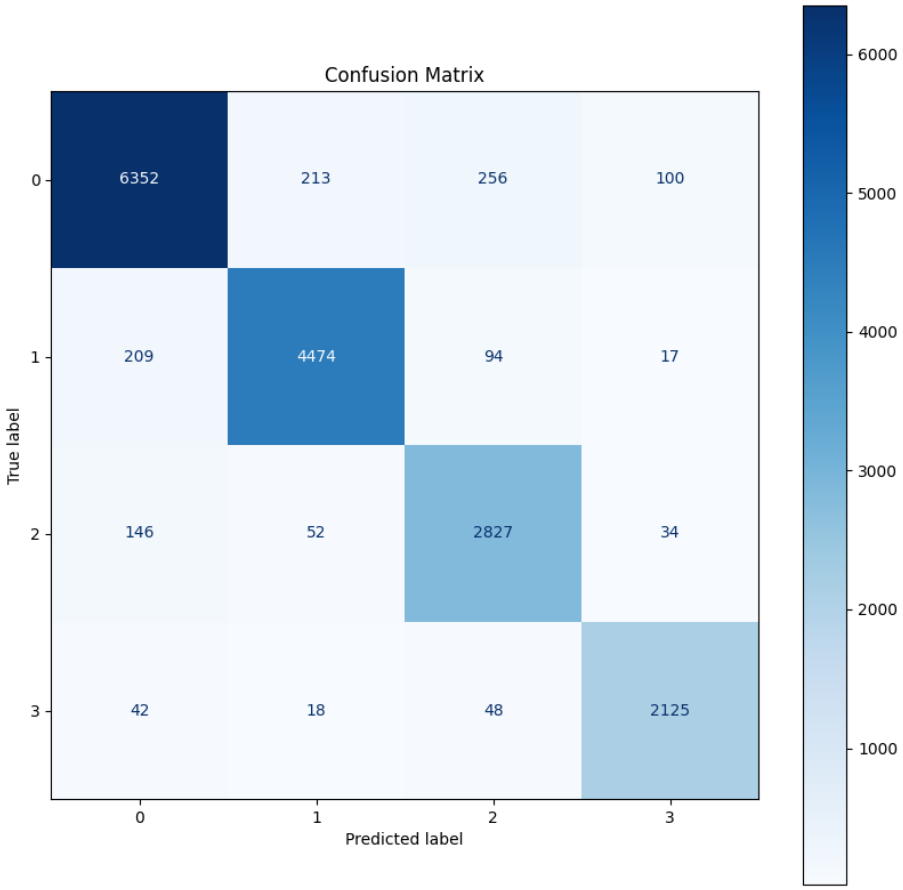

Figura 25: Matriu de confusió del model de classificació per a 4 classes

Amb tot això, es pot concloure que aquest model és molt senzill i efectiu, amb grans capacitats de generalització. Això segurament es deu a que les 4 classes són bastant genèriques i el model capta fàcilment els tipus de frases que s'utilitzen en cada un dels àmbits.

4.2 Model de classificació amb 53 classes

Per aquesta segona part la classificació passa a tenir 53 classes. Això provoca que les classes ja no siguin gaire genèriques i segurament els tenen una major dificulatat per aprendre els patrons de les frases de cada temàtica.

En aquest cas, el model utilitzat també utilitza optimitzador Adam amb `learning_rate=0.001` i també s'ha començat provant sense capes ocultes. De la mateixa manera que abans, per contrarestar el desbalanceig també s'han assignat diferents pesos a les 53 classes.

Una vegada entrenat el model, els resultats no són gaire bons: 0.6457 d'accuracy en la partició de train i 0.6410 en la de validació. En la figura 26 es poden veure les corbes d'aprenentatge d'aquest entrenament.

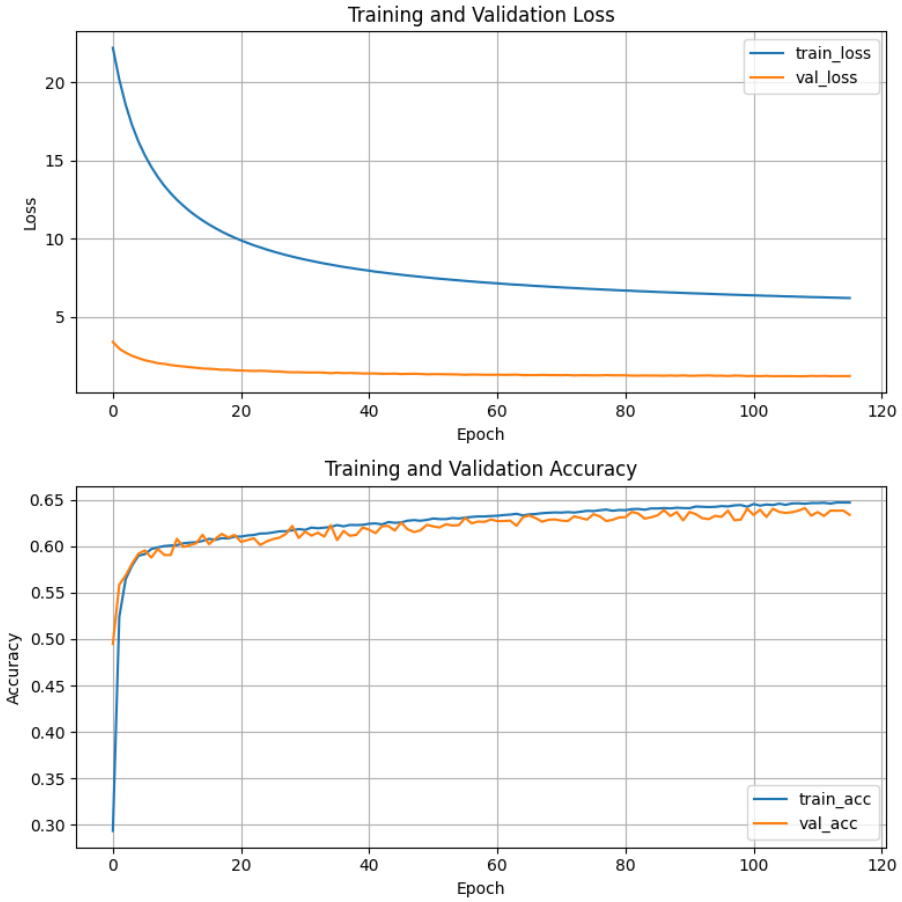

Figura 26: Corbes d'error (loss) i precisió (accuracy) en les particions de train i validation

Com que el model no sembla poder aprendre correctament les dades, s'ha provat d'afegir mesures dràstiques: tres capes ocultes amb 2048, 2048 i 1024 neurones, respectivament, i funció d'activació ReLU.

Després d'entrenar aquest nou model, els resultats no han estat gaire millor. Concretament, s'ha obtingut un accuracy de 0.6743 en el train i de 0.6671 en el validation, però el temps d'entrenament ha estat molt superior. En la figura 27 es poden veure les corbes d'entrenament d'aquest nou model. Veient aquestes corbes, sembla que el model podria haver millorat una mica més si l'`EarlyStopping` tingués una paciència superior, però seria a costa d'overfitting i sense una gran millora.

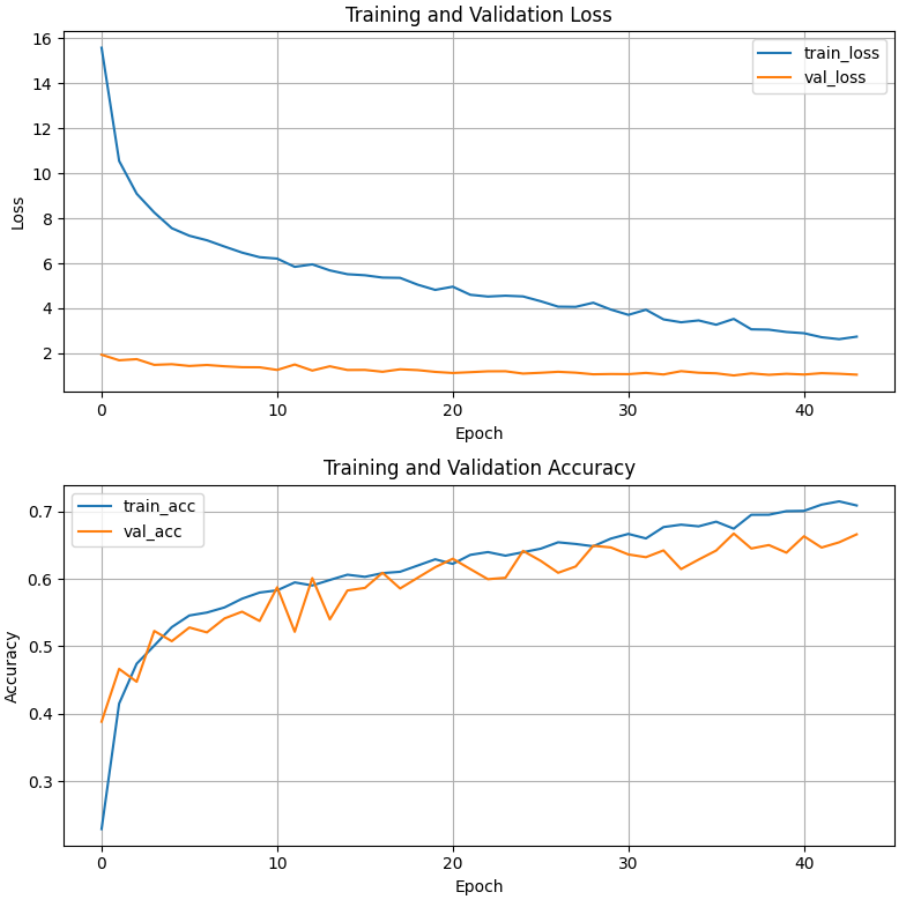

Figura 27: Corbes d'error (loss) i precisió (accuracy) en les particions de train i validation

En la partició de test, l'accuracy obtingut ha estat de 0.6671 i l'F1-Score (macro) tan sols ha arribat fins a 0.5805. Aquests resultats són bastant més dolents que els anteriors, probablement degut a que tenir 53 classes fa que aquestes siguin molt específicques i sigui difícil determinal el tema d'un text mitjançant la mitjana dels vectors de les seves paraules.

En la figura 28 es poden veure els resultats de les prediccions per les 53 classes diferents amb aquest últim model. És destacable com hi ha certes classes que es confonen bastant, probablement pel fet de que tenen temàtiques que utilitzen paraules similars.

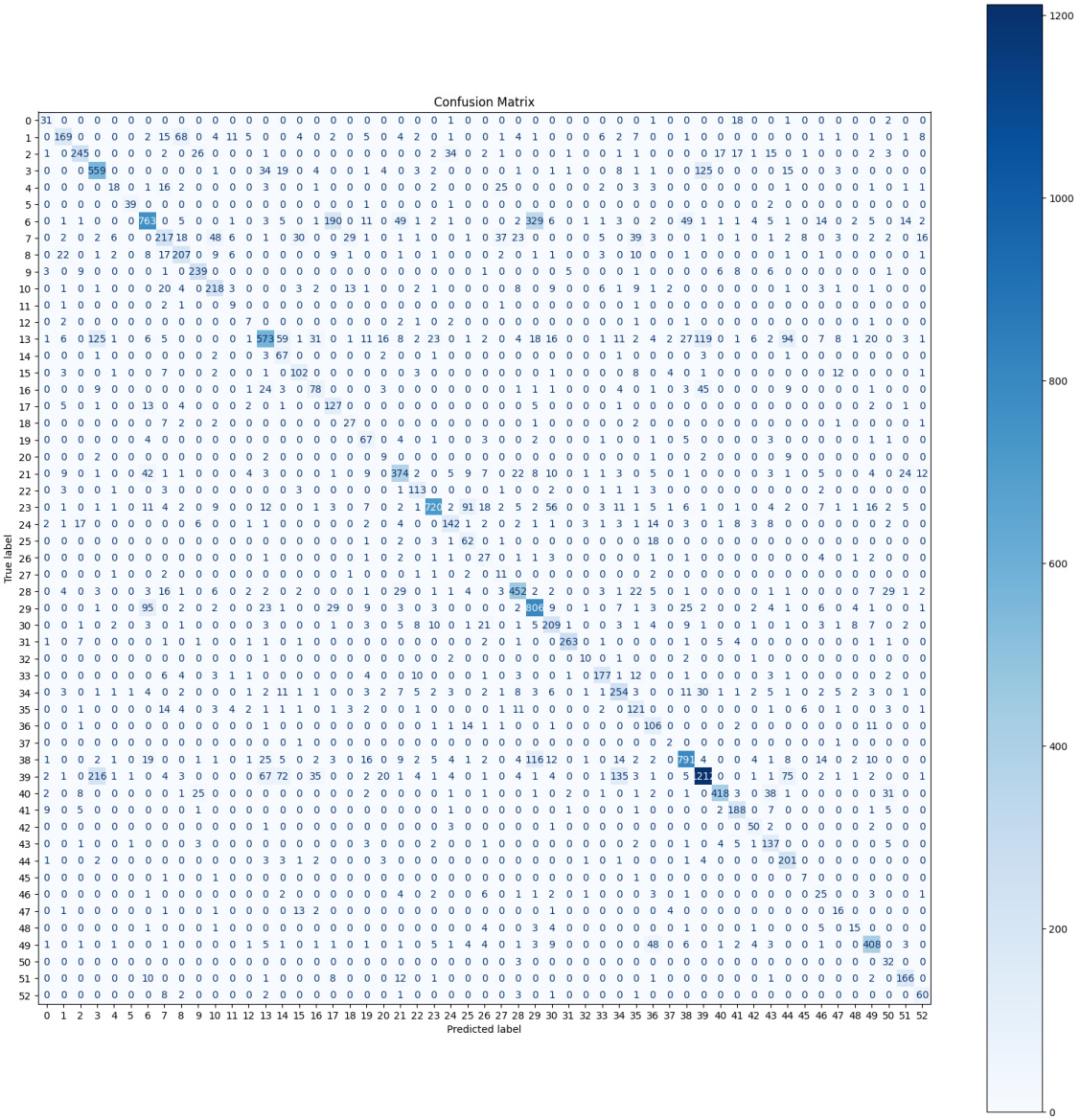

Figura 28: Matriu de confusió del model de classificació per a 4 classes

Sembla que, encara que el model sigui altament complex, és molt difícil obtenir bons resultats en aquesta classificació. Pot ser que altres models, com ara Random Forest, proporcionessin millores en la classificció, però segurament la millora no seria gaire notable. Per altra banda, segurament la única manera d'obtenir una millora substancial és mitjançant un model d'embeddings més complet, que utilitzés vectors de mida superior, el paràmetre `window` també superior, o que hagi estat entrenat amb més epochs, ...

5 Conclusions

Després de realitzar múltiples experiments en les diferents seccions d'aquesta pràctica, s'ha pogut veure que, de forma general, els models de tipus Word2Vec són útils però els hi costa arribar als resultats que obtenen altres models basats en transformers, com ara Bert o RoBERTa. A més, també, s'ha pogut veure que els vectors de mida més gran, o l'augment d'altres paràmetres com ara `window`, proporcionen millores notables en tots els tipus de models.

Aquest projecte ha permès demostrar com els models basats en transformers han suposat una gran revolució en l'àmbit del Processament del Llenguatge Humà i ha permès entendre els motius pels que els Large Language Models (LLMs) són tan bons amb múltiples tasques, més enllà de només la generació de text.

A més, també s'ha pogut comprovar la importància del hardware en l'entrenament de models d'aquest tipus, ja que equips bàsics com ara ordinadors portàtils poden derivar en temps d'entrenament molt alts o capacitat d'entrenar models relativament senzills. En canvi, amb hardware d'altes prestacions es poden entrenar models amb mola més informació, mida de vectors i altres paràmetres que ajuden a que els resultats obtinguts siguin substancialment millors, i amb temps d'entrenament més raonables.

6 Referències

[1] Hugging Face. projecte-aina/catalan_general_crawling. `https://huggingface.co/datasets/projecte-aina/catalan_general_crawling`, 2020.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[3] Wikipedia contributors. T-distributed stochastic neighbor embedding — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=T-distributed_stochastic_neighbor_embedding&oldid=1205815221`, 2024. [Online; accessed 3-June-2024].

[4] M. Radenković, R. Řehůřek, and P. Sojka. Gensim test data: wordsim353.tsv. `https://github.com/piskvorky/gensim/blob/develop/gensim/test/test_data/wordsim353.tsv`, 2021.

[5] Jordi Armengol-Estapé, Casimiro Pio Carrino, Carlos Rodriguez-Penagos, Ona de Gibert Bonet, Carme Armentano-Oller, Aitor Gonzalez-Agirre, Maite Melero, and Marta Villegas. Are multilingual models the best choice for moderately under-resourced languages? A comprehensive assessment for Catalan. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4933–4946, Online, August 2021. Association for Computational Linguistics.

[6] Explosion AI. spacy ca_core_news_md, 2023. Version 3.5.0.

[7] Jordi Armengol-Estapé, Casimiro Pio Carrino, Carlos Rodriguez-Penagos, Ona de Gibert Bonet, Carme Armentano-Oller, Aitor Gonzalez-Agirre, Maite Melero, and Marta Villegas. Are multilingual models the best choice for moderately under-resourced languages? A comprehensive assessment for Catalan. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4933–4946, Online, August 2021. Association for Computational Linguistics.

[8] Jordi Armengol-Estapé, Casimiro Pio Carrino, Carlos Rodriguez-Penagos, Ona de Gibert Bonet, Carme Armentano-Oller, Aitor Gonzalez-Agirre, Maite Melero, and Marta Villegas. Are multilingual models the best choice for moderately under-resourced languages? A comprehensive assessment for Catalan. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4933–4946, Online, August 2021. Association for Computational Linguistics.

[9] The Nordic Language Processing Laboratory (NLPL). The nlpl vector repository, 2024. Accessed: 2024-06-05.

[10] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[11] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding. *arXiv preprint arXiv:2004.09297*, 2020.

[12] Hugging Face. projecte-aina/tecla. `https://huggingface.co/datasets/projecte-aina/tecla`, 2020.