

Word-Embeddings

**Pràctica 4**

**Processament del llenguatge humà**

*Intel·ligència artificial*

Autors

Llum Fuster Palà

Artur Aubach Altes

Grup 11

Salvador Medina Herrera

Quadrimestre Primavera 2022/2023

CONTINGUTS

[**1. INTRODUCCIÓ 2**](#_2et92p0)

[**2. WORD-EMBEDDING MODELS 3**](#_bjltiijexp81)

[2.1 WORD2VEC 3](#_x0my8ltvy8ud)

[2.1.1 100 MB 3](#_mgz9rt4hkvhy)

[2.1.2 500 MB 3](#_dh8bw3f09so5)

[2.2 FASTTEXT 4](#_mx4ifzfp5ihx)

[2.3 ANÀLISI DE RESULTATS 4](#_d6rs2hph8g9m)

[**3. TEXT SIMILARITY: SINGLE VECTOR 6**](#_uup4xxyyfdz1)

[3.1 ONE-HOT-ENCODER 6](#_kqvvmda8no68)

[3.2 WORD2VEC 7](#_baah5pcefq3)

[3.2.1 AMB MEAN 8](#_xwm3whl2oibj)

[3.2.2 AMB TFIDF 8](#_yrn2svj8cc47)

[3.3 SPACY 9](#_bork8lc7e4ik)

[3.4 ROBERTA 9](#_a0sk9qgxhp55)

[3.5 ROBERTA FINE-TUNED 9](#_yhh0rjh6lwk6)

[3.6 COMPARACIÓ DE RESULTATS 10](#_62o2mkgcgskm)

[**4. TEXT SIMILARITY: SEQUENCE VECTOR 12**](#_qc7x2ibm6uhs)

[4.1 MODELS DE MLP 12](#_9ajkj44mqf3w)

[4.2 COMPARACIÓ D’EMBEDDINGS 13](#_4q9f1zainnz)

[**5. MODEL DE CLASSIFICACIÓ 18**](#_s4qqven2vvb2)

[5.1 FUNCIONS 18](#_etm6ra50qboq)

[5.3 DADES 20](#_xim604s9c8sq)

[5.3 RESULTATS 22](#_x9lp28bou7hk)

[**6. CONCLUSIONS 24**](#_b3hwwtpzh7ow)

# 

# 1. INTRODUCCIÓ

El present treball s'enfoca en l'exploració i la implementació de diferents tècniques de modelatge de llenguatge i anàlisi de similitud de text en llengua catalana. Aquesta recerca no només contribuirà a la comprensió de com els models de llenguatge processen i entenen el text, sinó que també proporcionarà una base per a l'aplicació pràctica d'aquests models en tasques com ara la classificació de textos.

En primer lloc, ens embarcarem en el desenvolupament de models d'incrustació de paraules, concretament el model Word2Vec, utilitzant diferents mides de conjunts de dades. La nostra font principal serà el corpus Catalan General Crawling, però considerarem també l'addició d'altres fonts de dades si és necessari. Opcionalment, explorarem també el model FastText.

Seguidament, procedirem a entrenar i implementar un model de similitud de text semàntic. Això ens permetrà comparar com diferents models d'incrustació de paraules, com ara One-Hot-Encoder, Word2Vec, spaCy, RoBERTa, i RoBERTa fine-tuned, poden afectar la capacitat del nostre model per a discernir la similitud semàntica entre textos.

A més, aprofundirem en la similitud de text amb vectors de seqüència, utilitzant embeddings entrenables inicialitzats amb Random Embeddings i Word2Vec. L'objectiu és analitzar com aquesta diferent aproximació afecta els resultats obtinguts en comparació amb l'ús de vectors únics.

Finalment, aplicarem aquests coneixements a la construcció d'un model de classificació de text, amb l'objectiu d'avaluar la seva eficàcia en una tasca pràctica.

A través d'aquesta recerca, esperem augmentar la comprensió sobre les diferents tècniques de modelatge de llenguatge i la seva aplicabilitat a diferents tasques de processament del llenguatge natural en català.

# 

# 2. WORD-EMBEDDING MODELS

Aquesta secció del treball està dedicada a l'estudi i la implementació de models d'incrustació de paraules, en concret Word2Vec i FastText. Començarem amb el model Word2Vec, amb el qual entrenarem dos conjunts de dades de diferents mides. Posteriorment, experimentarem amb el model FastText de manera opcional. Un cop completats aquests passos, procedirem a analitzar els resultats obtinguts. Els models d'incrustació de paraules són una part fonamental en el processament del llenguatge natural, per això dedicarem esforç en aquesta fase per assegurar-nos de la qualitat de les bases per a les etapes següents del projecte.

## 2.1 WORD2VEC

El model Word2Vec es distingeix com una de les metodologies més apreciades en l'àmbit del processament del llenguatge natural. Aquesta tècnica proporciona una forma eficient i intuïtiva de representar les paraules com a vectors multidimensionals, capturant la seva semàntica i les relacions semàntiques amb altres paraules en un espai vectorial dens. L'objectiu d'aquesta secció és entrenar aquest model amb conjunts de dades de diferents mides per a analitzar l'impacte de la mida del dataset en la qualitat de les incrustacions generades.

### 2.1.1 100 MB

En primer lloc, ens hem dedicat a entrenar un model Word2Vec utilitzant un conjunt de dades de 100MB. Per a fer-ho, hem seleccionat el corpus "Catalan General Crawling" disponible en la plataforma Hugging Face (https://huggingface.co/datasets/projecte-aina/catalan\_general\_crawling). A partir d'aquest corpus, hem extret un subconjunt que concorda amb la mida requerida. Aquest model, tot i treballar amb una quantitat limitada de dades, pretén capturar l'essència del llenguatge català i representar les paraules en un espai vectorial.

### 2.1.2 500 MB

Paral·lelament, hem entrenat un segon model Word2Vec, aquest cop utilitzant un conjunt de dades de 500MB. De la mateixa manera que amb el primer model, hem utilitzat el corpus "Catalan General Crawling", però en aquesta ocasió hem seleccionat un subconjunt més gran. L'objectiu d'aquest model és no només capturar la semàntica del català, sinó també entendre com l'augment de la mida del conjunt de dades pot afectar la qualitat i la precisió de les incrustacions generades.

## 2.2 FASTTEXT

Seguint amb la nostra exploració dels models d'incrustació de paraules, hem implementat FastText. FastText, a diferència de Word2Vec, considera el context dels caràcters dins d'una paraula, la qual cosa pot millorar la representació de les paraules infreqüents i permetre la representació de paraules fora de vocabulari. Per a aquesta part del treball, hem utilitzat el model pre-entrenat cc.ca.300.bin.gz. Aquest model ha estat entrenat en un gran corpus de text català, assegurant així una representació de qualitat de la semàntica del llenguatge.

## 2.3 ANÀLISI DE RESULTATS

Per tal d'avaluar el rendiment dels nostres models d'incrustació de paraules, hem dut a terme una sèrie de proves per examinar com cada model captura les relacions semàntiques dins del llenguatge català.

En primer lloc, hem examinat els vectors generats per cada model per a la paraula "parlament". Tot i que aquesta anàlisi no aporta una gran quantitat de detalls específics sobre el funcionament dels models, ens ofereix una primera impressió de com cada model interpreta aquest terme en el seu espai vectorial.

A continuació, hem realitzat proves amb la funció doesnt\_match per observar com cada model determina quina paraula, dins d'un conjunt, té la menor relació semàntica amb la resta. Hem dut a terme aquesta prova amb dos conjunts de paraules diferents. En el primer conjunt, compost per les paraules ["cadira", "sofa", "gat", "butaca"], tots els models van proporcionar el resultat esperat. No obstant això, quan vam ampliar el conjunt a un grup més gran i complex de paraules (["mà", "braç", "cua", "coll", "cap","cos", "peu", "ull", "pau", "mar"]), només el model FastText va ser capaç d'identificar correctament la paraula 'mà' com la menys relacionada.

En la següent prova, vam proporcionar a cada model la paraula "espardenyes", amb la finalitat que cadascun ens indicara quina altra paraula es semblava més a aquesta. En aquest cas, vam trobar que tant el model de 500MB com el FastText van retornar les respostes més adequades.

| ft\_model.most\_similar("espardenyes", topn=5) >>>[('espardenyes-', 0.8460028171539307), ('Espardenyes', 0.7968662977218628), ('espardenyetes', 0.7764989137649536), ('espardenyeres', 0.749383270740509), ('espardenyer', 0.7375205755233765)]  model100.wv.most\_similar("espardenyes", topn=5) >>>[('faldilletes', 0.7622565627098083), ('ungles', 0.7591675519943237), ('calcetes', 0.7253513932228088), ('dolces,', 0.7216159105300903), ('crestes', 0.7189742922782898)]  model500.wv.most\_similar("espardenyes", topn=5) >>>[('sabates', 0.8071151971817017), ('sandàlies', 0.7824795842170715), ('botes', 0.7782108783721924), ('sabates,', 0.7755588293075562), ('gorres', 0.7714869379997253)] |
| --- |

Finalment, vam aplicar una prova d'aritmètica de paraules, on proporcionàvem al model dues paraules d'una banda i una altra d'una altra banda. La tasca del model era retornar una paraula que fos similar a les dues primeres, però oposada a l'última. Novament, vam trobar que tant el model de 500MB com el FastText van proporcionar les respostes més coherents i adequades.

| ft\_model.most\_similar(positive=["reina", "home"], negative=["dona"], topn=5) >>>[('rei', 0.637349545955658), ('monarca', 0.5891644358634949), ('reinat', 0.5329504013061523), ('príncep', 0.5230845808982849), ('reinaxement', 0.5037134885787964)]  model100.wv.most\_similar(positive=["reina", "home"], negative=["dona"], topn=5) >>>[('príncep', 0.5405932664871216), ('adversari', 0.5301108956336975), ('monarca', 0.5290896892547607), ('Leon', 0.5268602967262268), ('emperador', 0.5265540480613708)]  model500.wv.most\_similar(positive=["reina", "home"], negative=["dona"], topn=5) >>>[('Borbó', 0.6063670516014099), ('heroi', 0.5928063988685608), ('rei', 0.5882807970046997), ('fill', 0.582746684551239), ('príncep', 0.5774911642074585)] |
| --- |

En resum, tots els models van demostrar la seva capacitat per capturar relacions semàntiques en el llenguatge català, però tant el model de 500MB com el FastText van destacar per la seva precisió i consistència.

# 3. TEXT SIMILARITY: SINGLE VECTOR

En aquesta secció, ens centrem en l'aplicació de diferents tècniques per avaluar la similitud de text mitjançant vectors únics. Aquests mètodes inclouen ONE-HOT-ENCODER, WORD2VEC, SPACY, ROBERTA i ROBERTA FINE-TUNED. Per tal de comparar aquests mètodes, també hem preparat una anàlisi comparativa dels resultats obtinguts.

Durant el desenvolupament d'aquesta part de la nostra recerca, hem fet ús del codi proporcionat, realitzant algunes petites modificacions per adaptar-lo a les necessitats de la nostra tasca. Entre aquestes modificacions, cal destacar la modificació de la funció map\_pairs per permetre l'ús de diferents models.

En alguns casos, vam trobar problemes degut a la forma dels vectors, que presentaven una dimensió addicional de mida 1. Per resoldre aquest problema, vam utilitzar les següents línies de codi per ajustar la forma dels vectors:

| x\_test = (tf.squeeze(x\_test[0], axis=1),tf.squeeze(x\_test[1], axis=1)) |
| --- |

| train\_dataset = train\_dataset.map(lambda x, y: ((tf.squeeze(x[0], axis=1), tf.squeeze(x[1], axis=1)), y)) val\_dataset = val\_dataset.map(lambda x, y: ((tf.squeeze(x[0], axis=1), tf.squeeze(x[1], axis=1)), y)) |
| --- |

Amb aquestes modificacions, vam poder procedir amb les nostres anàlisis sense més obstacles.

## 3.1 ONE-HOT-ENCODER

Per aquesta tècnica, hem ajustat l'encoder amb totes les paraules presents en el nostre diccionari. Per fer-ho hem afegit un nou paràmetre anomenat one\_hot\_model, que pren per valor un objecte OneHotEncoder en cas de estar executant aquest model. L'ajustament s'ha realitzat mitjançant la línia de codi següent:

| one\_hot\_model.fit(np.array(list(dictionary.values())).reshape(-1, 1)) |
| --- |

Aquesta línia transforma el diccionari de paraules en un array Numpy, que després es redimensiona per adaptar-se a les necessitats de l'encoder.

Una vegada ajustat l'encoder, hem aplicat OneHotEncoding a les frases preprocessades. La implementació d'aquesta tasca és la següent:

| vectors1 = [map\_one\_hot(sentence, one\_hot\_model) for sentence in sentence\_1\_preproc] vectors2 = [map\_one\_hot(sentence, one\_hot\_model) for sentence in sentence\_2\_preproc] vector1 = np.sum(vectors1, axis=0)  vector2 = np.sum(vectors2, axis=0) |
| --- |

En aquest codi, es calculen vectors one-hot per a cada frase preprocessada en els conjunts de frases 1 i 2. Posteriorment, es sumen tots els vectors per a cada conjunt de frases per obtenir un sol vector per a cada conjunt.

Aquesta estratègia permet transformar cada paraula en un vector únic en un espai de dimensió igual al nombre total de paraules en el diccionari, destacant la presència d'una paraula concreta amb un 1 i la resta amb zeros. És un mètode de codificació binària que, tot i no capturar relacions semàntiques entre paraules, pot ser útil en certs contextos. Les comparacions de rendiment d'aquest enfocament amb altres tècniques d'incrustació de paraules es detallen en seccions posteriors del present informe.

| else: *#opcio 2 -> no fem servir tf\_idf*  if one\_hot\_model is not None:  *# Ajustamos el encoder con todas las palabras en el diccionario*  one\_hot\_model.fit(np.array(list(dictionary.values())).reshape(-1, 1))  *# Aplicamos OneHotEncoding a las frases preprocesadas*  vectors1 = [map\_one\_hot(sentence, one\_hot\_model) for sentence in sentence\_1\_preproc]  vectors2 = [map\_one\_hot(sentence, one\_hot\_model) for sentence in sentence\_2\_preproc]  vector1 = np.sum(vectors1, axis=0)   vector2 = np.sum(vectors2, axis=0) |
| --- |

## 3.2 WORD2VEC

En aquesta secció, hem fet ús de l'algorisme Word2Vec per transformar cada paraula dels nostres conjunts de frases en vectors de paraules, conservant les relacions semàntiques entre elles. Hem executat dues estratègies diferents per a aquesta tasca: la mitjana ponderada de vectors (denominada "Amb Mean") i la ponderació per Term Frequency-Inverse Document Frequency (TFIDF).

### 3.2.1 AMB MEAN

Per aquesta estratègia, la implementació s'ha enfocat en calcular la mitjana ponderada dels vectors de paraules per a cada frase. No obstant això, hi ha hagut casos en què cap paraula d'una frase no s'ha trobat en el model Word2Vec. Per aquests casos, s'ha introduït una condició per retornar un vector de zeros:

| *# Afegit pels casos on els vectors no tenen cap valor (no s'ha trobat cap paraula de la sentence)* if vectors1 == []:  vector1 = np.zeros((100, )) else:  vector1 = np.average(vectors1, weights=weights1, axis=0, ) *#mitjana ponderada* if vectors2 == []:  vector2 = np.zeros((100, )) else:  vector2 = np.average(vectors2, weights=weights2, axis=0, ) |
| --- |

Aquest codi retorna un vector de zeros quan no es troba cap paraula de la frase en el model Word2Vec, i en cas contrari, calcula la mitjana ponderada dels vectors de paraules.

### 3.2.2 AMB TFIDF

Amb l'estratègia TFIDF, hem aplicat un esquema de ponderació similar a l'estratègia "Amb Mean", amb la diferència que aquí hem usat pesos de Term Frequency-Inverse Document Frequency per cada paraula:

| *# Afegit pels casos on els vectors no tenen cap valor (no s'ha trobat cap paraula de la sentence)* if vectors1 == []:  vector1 = np.zeros((100, )) if vectors2 == []:  vector2 = np.zeros((100, )) |
| --- |

La codificació TFIDF és útil per donar més importància a les paraules que són rares en el corpus però freqüents en documents específics. Aquesta estratègia pot captar millor el significat de les frases en comparació amb l'estratègia "Amb Mean", especialment quan es tracta de textos més llargs. En aquest cas, també hem introduït una condició per a retornar un vector de zeros quan cap paraula de la frase no es troba en el model Word2Vec.

## 3.3 SPACY

Per a l'ús de l'eina Spacy en aquesta secció, hem optat per la versió ‘small’ de Spacy per processar el nostre text. Primer hem fet el mateix que amb el One-Hot, afegir el model com a paràmetre; en aquest cas seria nlp, que serà un model carregat de spacy (nlp = spacy.load("ca\_core\_news\_sm")). Hem ampliat la implementació existent per a incloure vectors de característiques per a cada paraula individualment en les frases. Posteriorment, hem calculat el vector mitjà de cada frase:

| elif nlp is not None: *#per spacy*  sent1 = nlp(sentence\_1)  sent2 = nlp(sentence\_2)  vectors1 = [sent1[i].vector for i in range(len(sent1)) if str(sent1[i]) in nlp.vocab]  vectors2 = [sent2[i].vector for i in range(len(sent2)) if str(sent2[i]) in nlp.vocab]  vector1 = np.mean(vectors1, axis=0)  vector2 = np.mean(vectors2, axis=0) |
| --- |

## 3.4 ROBERTA

Per a l'ús del model RoBERTa, hem optat per utilitzar el model ‘transformer’ de spacy que equival al RoBERTa. A més, hem elegit l'opció 'CLS' que retorna l'estat ocult del token especial 'CLS', el qual és útil per a tasques de classificació. A diferència dels models anteriors, RoBERTa codifica tota la informació de la frase en aquest token 'CLS'. Una altra opció que podríem plantejar seria agafar la matriu i fer-ne la mitjana (codi en comentari).

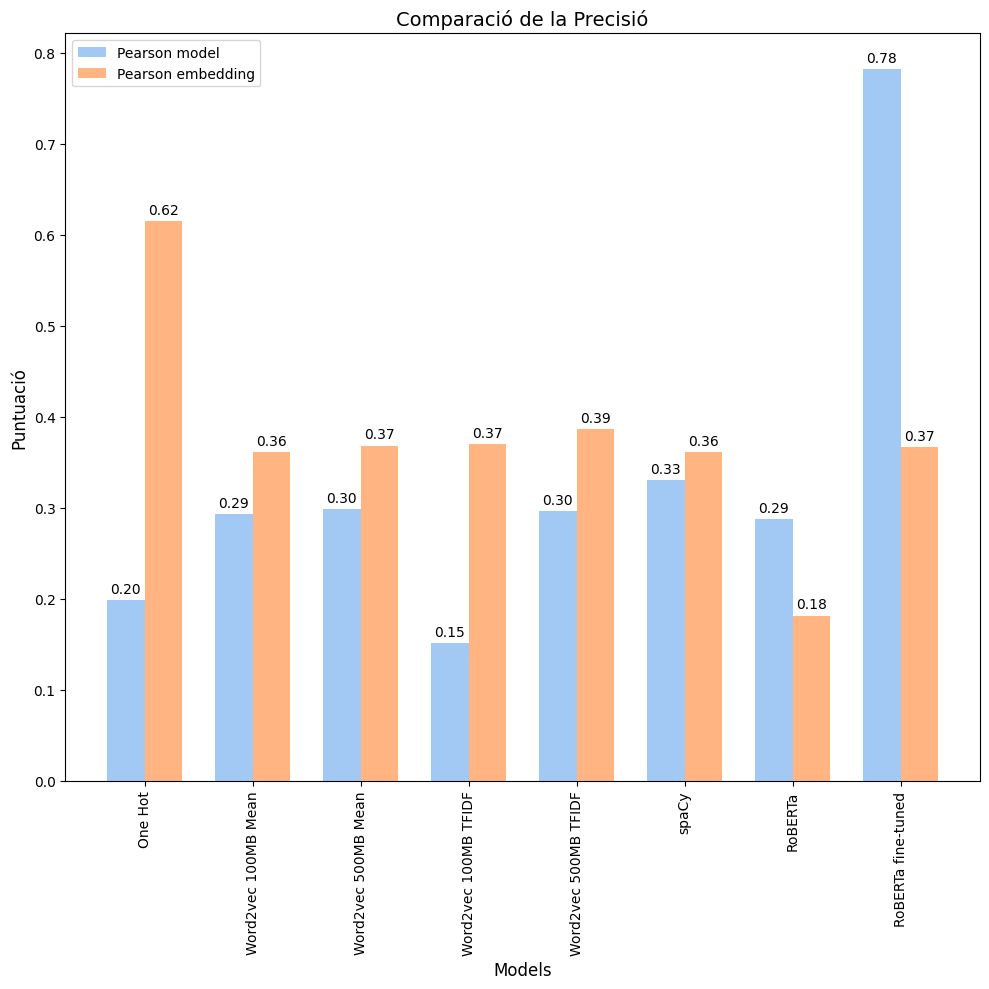
| elif nlp is not None: *#per spacy*  if 'transformer' in nlp.pipe\_names: *# RoBERTa: versio trf de spacy*  sent1 = nlp(sentence\_1)  sent2 = nlp(sentence\_2)  *#amb CLS*  vector1 = sent1.\_.trf\_data.tensors[-1]  vector2 = sent2.\_.trf\_data.tensors[-1]  *#amb mitjana*  '''matr1 = sent1.\_.trf\_data.tensors[0]  matr2 = sent2.\_.trf\_data.tensors[0]  vector1 = np.mean(matr1, axis=0)  vector2 = np.mean(matr2, axis=0)''' |
| --- |

## 3.5 ROBERTA FINE-TUNED

Per a la implementació de RoBERTa fine-tuned, hem fet ús del model de RoBERTa amb fine-tuning disponible a Hugging Face a la següent URL: https://huggingface.co/projecte-aina/roberta-base-ca-v2-cased-sts. Aquest model ha estat ajustat específicament per a la tasca de Semantic Textual Similarity (STS) i, per tant, estem expectants per veure si supera els altres models que hem utilitzat fins ara.

## 3.6 COMPARACIÓ DE RESULTATS

Per a la comparació de resultats, ens hem basat en dues mesures clau: el coeficient de correlació de Pearson per a la modelització completa (Pearson model) i el coeficient de correlació de Pearson per a l'embedding (Pearson embedding). Aquestes mesures ens proporcionen una indicació de com es correlaciona la predicció del model amb la puntuació humana, essent 1 una correlació perfecta i 0 una correlació nul·la.



Basant-nos en les puntuacions de Pearson obtingudes per a cada model, podem observar les següents tendències:

1. El model amb la puntuació més alta en "Pearson model" és el RoBERTa fine-tuned, amb una correlació significativa de 0.781989. Això indica que aquest model té la capacitat d'encapsular més eficaçment la semàntica del text i se situa molt a prop del raonament humà en la tasca de similaritat textual semàntica.
2. El model One Hot Encoder destaca en la mesura "Pearson embedding", amb una puntuació de 0.615199. Això suggerix que l'encodificació one-hot pot generar representacions vectorials del text amb una certa correlació positiva amb la puntuació humana, però la seva eficàcia en la modelització completa és significativament inferior a la de models més sofisticats com RoBERTa fine-tuned.
3. Observem que hi ha una certa divergència entre les puntuacions de Pearson per a la modelització completa i l'embedding en alguns casos. Per exemple, RoBERTa obté una puntuació relativament alta en "Pearson model", però la seva puntuació en "Pearson embedding" és significativament més baixa. Aquesta divergència pot ser indicativa de la complexitat dels models de transformer com RoBERTa, que no només confien en les representacions vectorials de les paraules, sinó també en altres factors com ara la posició de les paraules en el text i les relacions entre elles.

En conclusió, mentre que l'encodificació One Hot Encoder pot proporcionar una representació vectorial decent, és el model RoBERTa fine-tuned el que proporciona les millors prediccions en termes de similaritat de text. No obstant això, també cal tenir en compte que l'aplicació d'aquests models en entorns de producció pot tenir altres consideracions, com ara el temps de càlcul i el consum de recursos, que poden fer que models més simples com Word2Vec o spaCy siguin preferibles en alguns casos.

# 

# 4. TEXT SIMILARITY: SEQUENCE VECTOR

A diferència de la similaritat de text basada en vectors simples, l'objectiu d'aquesta secció és entrenar una xarxa neuronal que també estigui entrenant els vectors embedding de les paraules. Els vectors embeddings són representacions denses de paraules que captura la seva semàntica, i l'entrenament d'aquests vectors permet que la xarxa aprengui a millor representar el text. En aquest cas, compararem els vectors entrenats amb vectors inicialitzats de manera aleatòria i amb els tres models de la secció 1.

## 4.1 MODELS DE MLP

Per tal d'explorar la similaritat de seqüències de text utilitzant diferents models de Perceptró Multicapa (MLP), hem realitzat diverses modificacions en la funció "build\_and\_compile\_model".

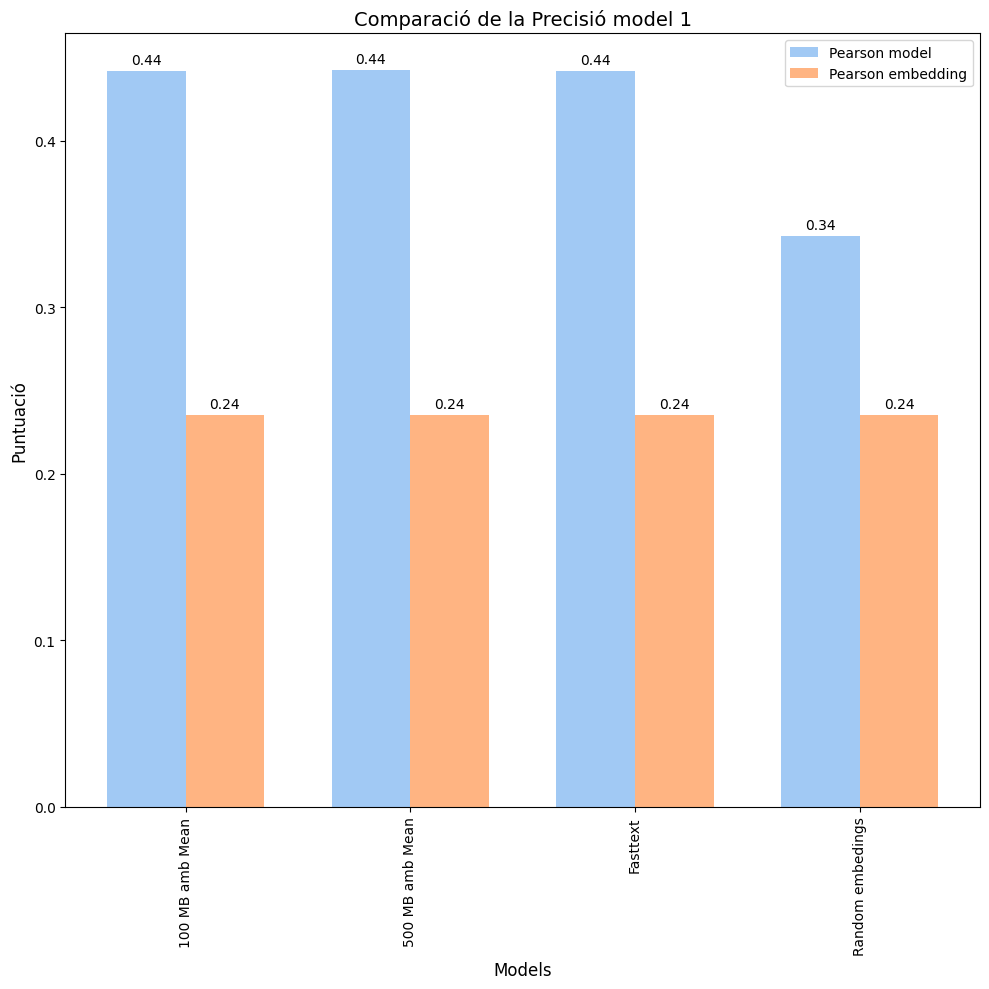
* En primer lloc, hem mantingut l'estructura del primer model que ja estava donat. Aquest model, utilitzant una configuració de paràmetres específica, ens ha servit de referència per a les modificacions posteriors.
* En segon lloc, també hem conservat l'estructura del segon model que es va proporcionar, per comparar els resultats amb els altres models.
* Tercerament, hem introduït una modificació anomenada "Model Adamax". En aquest cas, hem canviat l'optimitzador a Adamax amb l'objectiu d'observar si aquest canvi millorava els resultats del model. Això es va aconseguir mitjançant la modificació de la línia de codi: model.compile(loss='mean\_squared\_error',optimizer=tf.keras.optimizers.Adamax(...))
* Finalment, hem creat un "Model de Regularització". En aquest model, hem incorporat la regularització per a prevenir l'overfitting, que és quan el model s'adapta excessivament als dades d'entrenament i no generalitza bé a dades noves. Aquesta modificació es va implementar amb l'afegit d'un nou paràmetre: "embeddings\_regularizer = None". A més, vam observar que la pèrdua es mantenia en 14, així que vam afegir un callback per a reduir progressivament la taxa d'aprenentatge quan la pèrdua de validació deixava de disminuir. En cas de tenir pesos pre-entrenats, es va optar per utilitzar la regularització, així que vam modificar la capa d'embedding per a afegir:

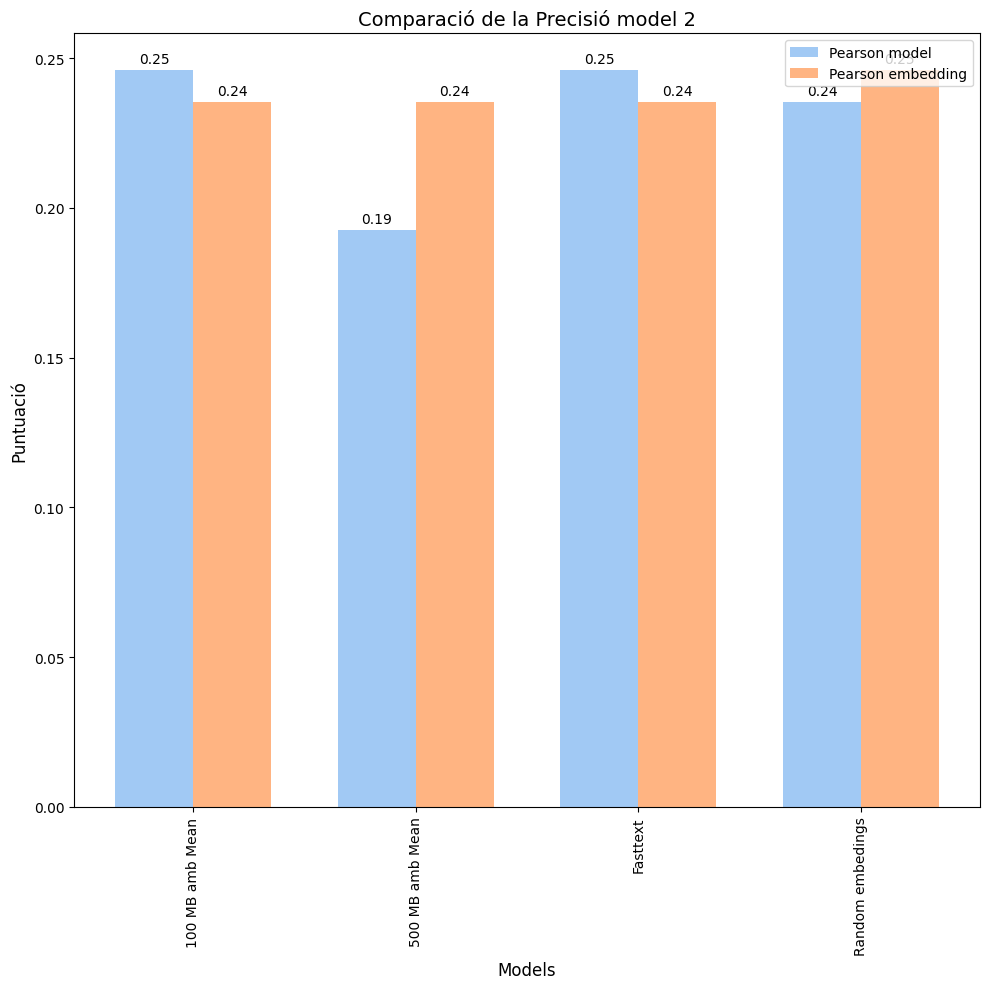
embedding = tf.keras.layers.Embedding(dictionary\_size, embedding\_size, input\_length=input\_length, mask\_zero=True, embeddings\_initializer=initializer, trainable=trainable, **embeddings\_regularizer=embeddings\_regularizer, )**

Aquestes adaptacions tenen com a objectiu millorar la capacitat dels nostres models MLP per a la tasca de similaritat de text, proporcionant així una base més sòlida per a la comparació del rendiment dels vectors embeddings amb diferents configuracions.

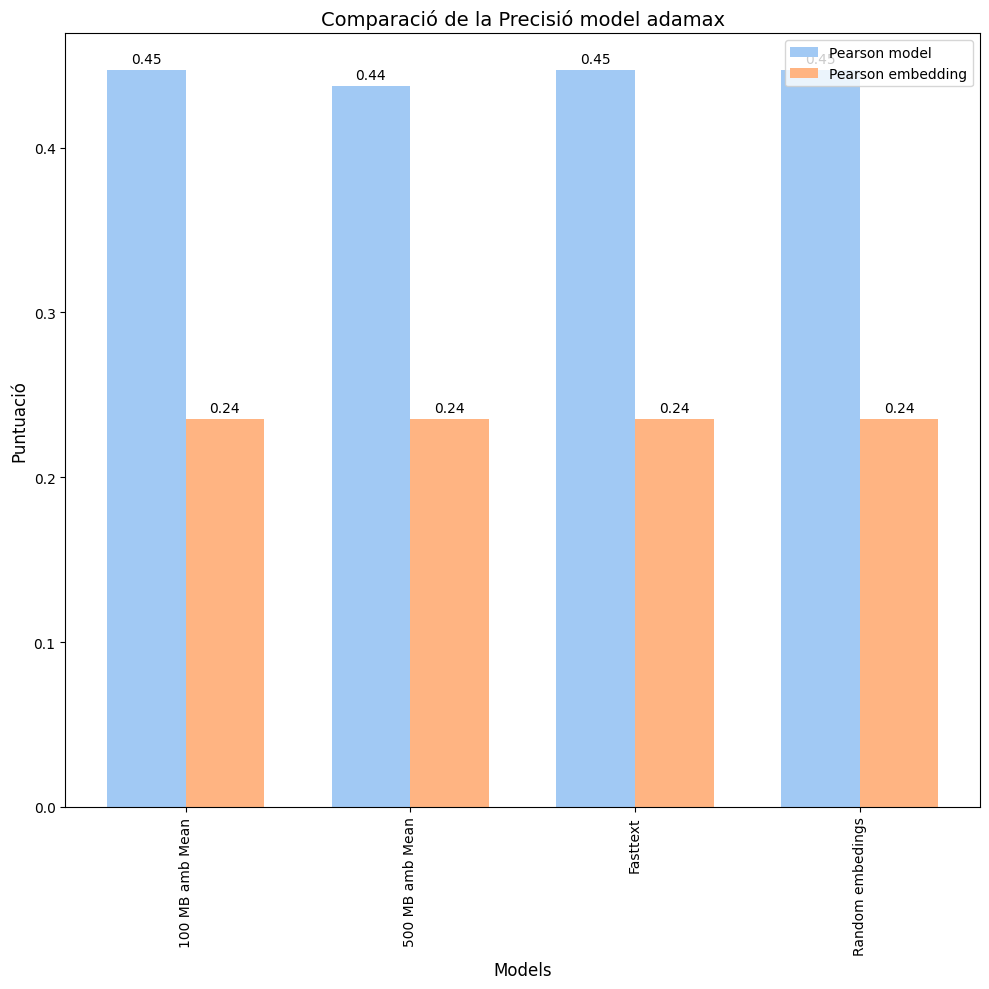
## 4.2 COMPARACIÓ D’EMBEDDINGS

De manera anàloga a l'apartat 3.6, per a la comparació d'embeddings en seqüència de text, hem utilitzat les mètriques de "Pearson model" i "Pearson embedding". Hem generat una sèrie de quatre gràfiques, una per a cada tipus de model (model 1, model 2, Adamax i Regularitzat), i hem extret conclusions en funció de les observacions de cada gràfica.  
Les embeddings pre-entrenades (100MB amb Mean, 500MB amb Mean, Fasttext) mostren una correlació més alta (0,442) amb el Model Pearson que els embeddings aleatoris (0,342). No obstant això, totes les embeddings mostren la mateixa correlació amb el Pearson embedding, que és d'aproximadament 0,235. Això pot indicar que, encara que els embeddings pre-entrenades proporcionen una millora en la modelització del text, l'impacte en la correlació amb l'embedding sembla ser limitat en aquest cas.

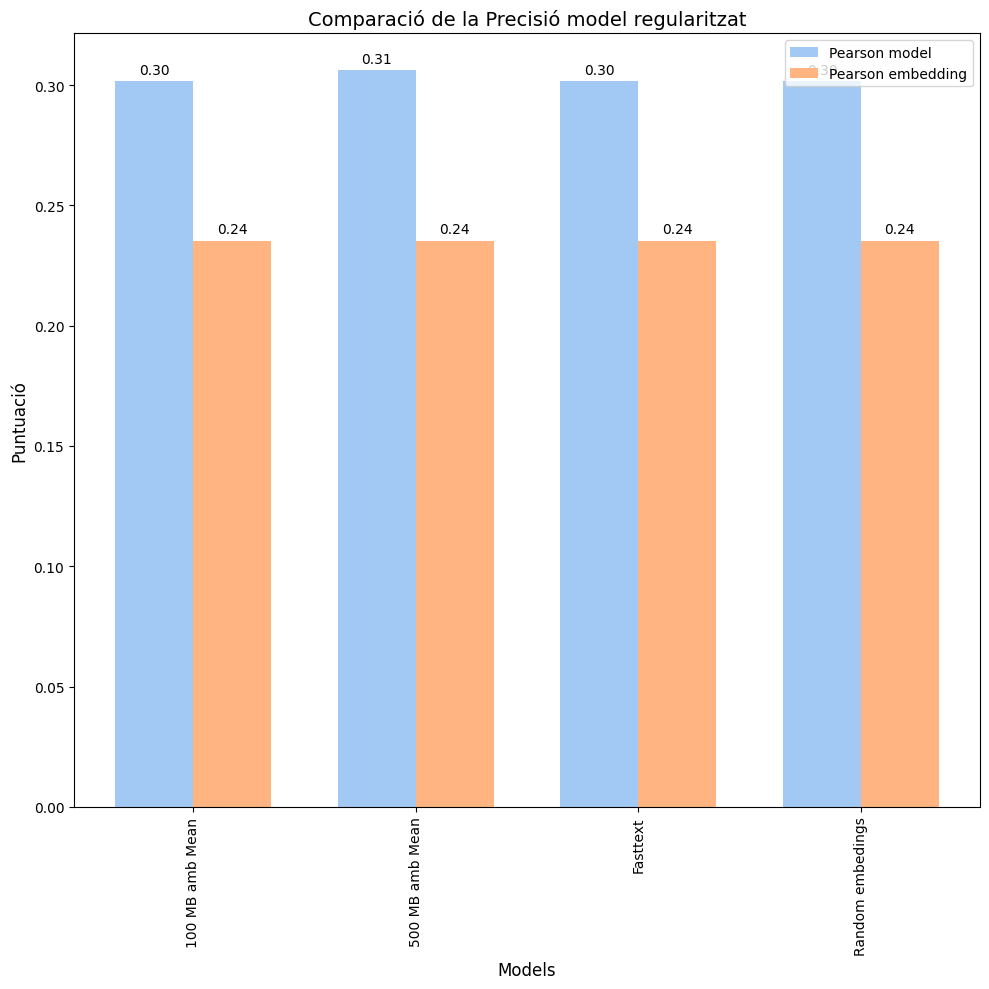
  
En aquest model, tots els embeddings pre-entrenats (100MB amb Mean, 500MB amb Mean, Fasttext) mostren una correlació de Pearson model similar (aproximadament 0,246), mentre que els embeddings aleatoris tenen una correlació lleugerament més baixa (0,235). No obstant això, els embeddings aleatoris mostren una correlació de Pearson embedding lleugerament més gran que la resta (0,246). Podem inferir que, encara que hi ha una millora amb els embeddings pre-entrenats, aquest model pot estar més influenciat per altres factors en la correlació de l'embedding.



Aquest model presenta una alta correlació de Pearson model per a tots els embeddings, tant pre-entrenats com aleatoris (aproximadament 0,447). Aquesta situació pot indicar que el model Adamax pot estar sobreajustant-se a les dades d'entrenament, produint resultats similars independentment del tipus d'embedding utilitzat. Els resultats de Pearson embedding per a tots els embeddings són consistents i més baixos (0,235), la qual cosa reforça la idea d'un possible sobreajustament.



Tots els embeddings pre-entrenats i aleatoris mostren una correlació de Pearson model similar (aproximadament 0,301). Aquest resultat pot indicar que la regularització està funcionant com es preveia, limitant l'impacte dels embeddings pre-entrenats en la correlació del model. La correlació de Pearson embedding és consistent (0,235) per a tots els embeddings, el que també pot ser un indicatiu de l'efectivitat de la regularització.



Aquesta anàlisi posa de relleu l'impacte significatiu que els embeddings pre-entrenats poden tenir en la correlació del model. També mostra l'importància de la regularització per limitar el possible sobreajustament, especialment en el cas del model Adamax.

# 5. MODEL DE CLASSIFICACIÓ

Per a entrenar el model de classificació hem decidit utilitzar els embeddings de GloVE ja que no hem entrenat el model per al Text Similarity. Així doncs, hem descarregat els embeddings de GloVE de mida de vectors 300 per tenir directament el més complex. Així doncs hem carregat els embeddings a les nostres dades i hem entrenat una xarxa MLP per a la tasca de classificació.

## 5.1 FUNCIONS

Per a entrenar el model hem necessitat diferents funcions:

La primera serveix per carregar els embedings a partir del path on es troba el model preentrenat de GloVe:

| def load\_word\_embeddings(embedding\_path):  word\_embeddings = {}  with open(embedding\_path, encoding='utf-8') as f:  for line in f:  values = line.split()  word = values[0]  vector = np.asarray(values[1:], dtype='float32')  word\_embeddings[word] = vector  return word\_embeddings |
| --- |

La segona funció l’utilitzem per fer un preprocessat dels textos previ a l’extracció d’embedings.

| def preprocess(text, stop\_words, punctuation):  tokens = word\_tokenize(text.lower())  words = [word for word in tokens if word.isalpha() and word not in stop\_words and word not in punctuation]  return words |
| --- |

La següent serveix per a extreure els emdedings de les nostres dades ja tokenitzades; només pren per paràmetre una sola paraula.

| def compute\_document\_embedding(words, word\_embeddings):  word\_vectors = []  for word in words:  if word in word\_embeddings:  word\_vectors.append(word\_embeddings[word])  if len(word\_vectors) == 0:  return None  else:  return np.mean(word\_vectors, axis=0) |
| --- |

La següent funció ens servirà per transformar la variable resposta a un vector de 0 i 1 perquè pugui ser processada per la xarxa MLP:

| def one\_hot\_encode\_labels(y):  ohe = OneHotEncoder(sparse=False)  y = ohe.fit\_transform(y)  return y, ohe.categories\_ |
| --- |

Després tenim una funció que únicament serveix per mostrar un gràfic amb les corbes de loss i mètriques de rendiment (accuracy, precision i recall):

| def plot\_training\_curve(history,name):  *#Generate the accuracy/loss plot during training*  print(history.history)  plt.plot(history.history['precision'], label='precision')  plt.plot(history.history['accuracy'], label='accuracy')  plt.plot(history.history['recall'],label='recall')  plt.plot(history.history['loss'], label='loss')  plt.title('Model Training')  plt.xlabel('Epoch')  plt.ylabel('Value')  plt.legend()  plt.savefig(f'./{name}Loss')  plt.show() |
| --- |

Finalment tenim la funció que crea, entrena i valida la xarxa MLP. Aquest afunció primer transforma la variable resposta de train i de validation, després, crea una MLP Sequential que consta de una capa oculta amb el nombre de neurones i funció d’activació determinats com a paràmetrse i després una capa de sortida amb funció softmax ja que es tracta d’una tasca de classificació.

| def mlpClassifier(X\_train, y\_train, X\_val, y\_val, name,neuron,activ):  y\_train, classes = one\_hot\_encode\_labels(y\_train)  classes = classes[0]  y\_val, \_ = one\_hot\_encode\_labels(y\_val)   print(y\_train.shape, y\_val.shape, classes, len(classes))   model = Sequential(  [Dense(neuron,activation=activ, kernel\_initializer=he\_normal())] + *#input\_size??*  [Dense(len(classes),activation='softmax')]  )    learning\_rate = 0.001  adam = Adam(learning\_rate=learning\_rate)   model.compile(loss='categorical\_crossentropy',  optimizer=adam,  metrics=[tf.keras.metrics.Precision(name='precision'),  tf.keras.metrics.CategoricalAccuracy(name='accuracy'),  tf.keras.metrics.Recall(name='recall')])   *# Train the model*  history = model.fit(X\_train, y\_train, epochs=20, verbose=1, batch\_size=64)  plot\_training\_curve(history,name)   *# Test the model*  predictions = model.predict(X\_val)  *#plot\_decision\_boundary(model,predictions,y\_val)*    *#print(f'### MLP - {name} ###')*   '''fig, ax = plt.subplots()  ax.scatter(predictions, y\_val, edgecolors=(0, 0, 1))  ax.plot([y\_val.min(), y\_val.max()], [y\_val.min(), y\_val.max()], 'r--', lw=3)  ax.set\_xlabel('Predicted')  ax.set\_ylabel('Actual')  plt.savefig(f'./{name}')  plt.show()'''   model.summary()  return predictions, classes |
| --- |

## 5.3 DADES

Les dade utilitzades pertanyen al dataset "projecte-aina/tecla" i es carrega i es fan una sèrie de transformacions per adaptar-les al nostre model:

| dataset = load\_dataset("projecte-aina/tecla") train = dataset['train'] *#datasets.dataset\_dict.DatasetDict* train = train[0:len(train)] *# canvia format de dataset a diccionari*  validation = dataset['validation'] validation = validation[0:len(validation)] *# canvia format de dataset a diccionari*  test = dataset['test'] test = test[0:len(test)] *# canvia format de dataset a diccionari*  *#convertir a Dataframe* df\_tr = pd.DataFrame(train) df\_val = pd.DataFrame(validation) df\_ts = pd.DataFrame(test) |
| --- |

A continuació carreguem els word embeddings i els apliquem a les dades creant una nova columna al dataframe anomenada ‘emb’:

| stop\_words = set(stopwords.words('english')) punctuation = set(string.punctuation)  embedding\_path = 'glove.6B.300d.txt' word\_embeddings = load\_word\_embeddings(embedding\_path) df\_tr['emb'] = None for i,text in enumerate(df\_tr['text']):  df\_tr['emb'][i] = compute\_document\_embedding(preprocess(text, stop\_words=stop\_words, punctuation=punctuation), word\_embeddings=word\_embeddings)  df\_val['emb'] = None for i,text in enumerate(df\_val['text']):  df\_val['emb'][i] = compute\_document\_embedding(preprocess(text, stop\_words=stop\_words, punctuation=punctuation), word\_embeddings=word\_embeddings)  df\_ts['emb'] = None for i,text in enumerate(df\_val['text']):  df\_ts['emb'][i] = compute\_document\_embedding(preprocess(text, stop\_words=stop\_words, punctuation=punctuation), word\_embeddings=word\_embeddings) |
| --- |

Seguidament transformem la variable label1 per a obtenir una variable categòrica amb les labels que es poden veure a la pàgina web de projecte aina que pertanyen a cada un dels nombres:

| def asignar\_etiqueta(valor):  if valor == 0:  return 'Societat'  elif valor == 1:  return 'Política'  elif valor == 2:  return 'Economia'  elif valor == 3:  return 'Cultura'  else:  return 'Otro'  *# Aplicar la función a la columna 'columna' y crear una nueva columna 'etiqueta'* df\_tr['etiqueta'] = df\_tr['label1'].map(asignar\_etiqueta) df\_val['etiqueta'] = df\_val['label1'].map(asignar\_etiqueta) df\_ts['etiqueta'] = df\_ts['label1'].map(asignar\_etiqueta) |
| --- |

Finalment obtenim les dades dividides entre X i y i les transformem al format adient amb np.array:

| X\_train = df\_tr['emb'] y\_train = df\_tr['etiqueta'] X\_val = df\_val['emb'] y\_val = df\_val['etiqueta'] X\_test = df\_ts['emb'] y\_test = df\_ts['etiqueta'] y\_train, y\_val = np.array([y\_train]).T, np.array([y\_val]).T  X\_train = np.array(X\_train.to\_list()) X\_val = np.array(X\_val.to\_list()) |
| --- |

from sklearn.preprocessing import OneHotEncoder

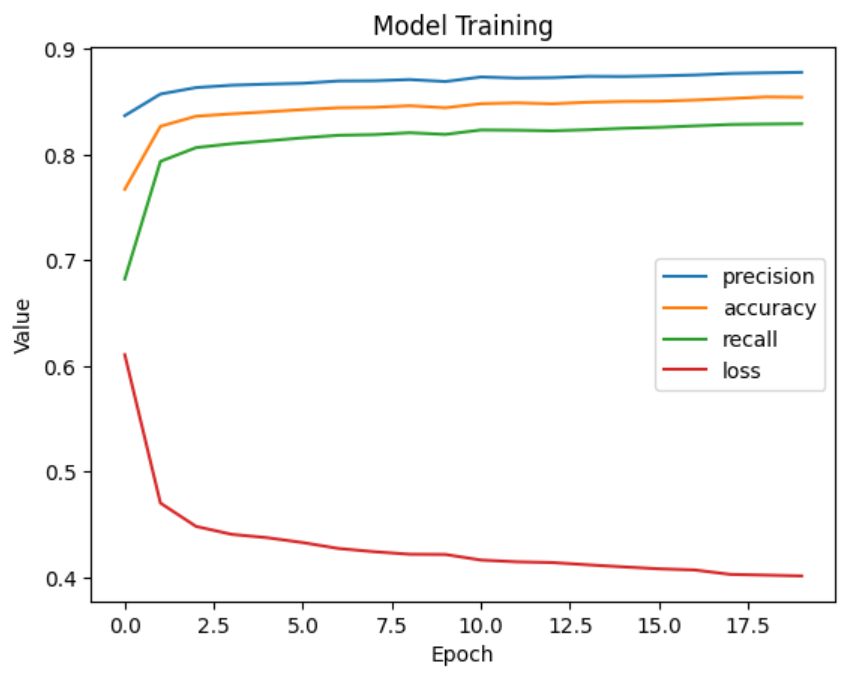
preds, classes = mlpClassifier(X\_train, y\_train, X\_val, y\_val, 'mlp1',128,'tanh')

## 5.3 RESULTATS

El model entrenat resultant ha estat el següent, amb 39044 paràmetres. Es pot veure com a la capa de sortida hi ha 4 neurones, una per cada classe de la variable.

| 178/178 [==============================] - 0s 2ms/step  Model: "sequential\_1"  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  Layer (type) Output Shape Param *#*  *=================================================================*  *dense\_2 (Dense) (None, 128) 38528*  *dense\_3 (Dense) (None, 4) 516*  *=================================================================*  *Total params: 39,044*  *Trainable params: 39,044*  *Non-trainable params: 0*  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |
| --- |

L’entrenament del model ha estat satisfactori obtenint aquesta corba de validació, que demostra que el model convergeix ràpidament i té una precisió molt elevada. La recall és una mica més baixa, però suficientment alta com per dir que el model dona bons resultats.



Després d’entrenar la xarxa, analitzem els resultats amb la matriu de confusió. Cal mencionar que no utilitzem el test ja que no hem provat diferents paràmetres, ens quedem amb el resultat de la validation, utilitzada com a test ja que no hem fet tria de models ni de paràmetres

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from scipy.spatial.distance import cosine

from sklearn.manifold import TSNE

import string

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

asign = {i:j for i,j in enumerate(classes)}

prediccions = []

for p in preds:

prediccions.append(asign[p.argmax()])

# Crear la matriu de confusió

import seaborn as sns

matriu\_conf = confusion\_matrix(y\_val.T[0], prediccions)

cm = np.array(matriu\_conf)

# Crear el mapa de calor

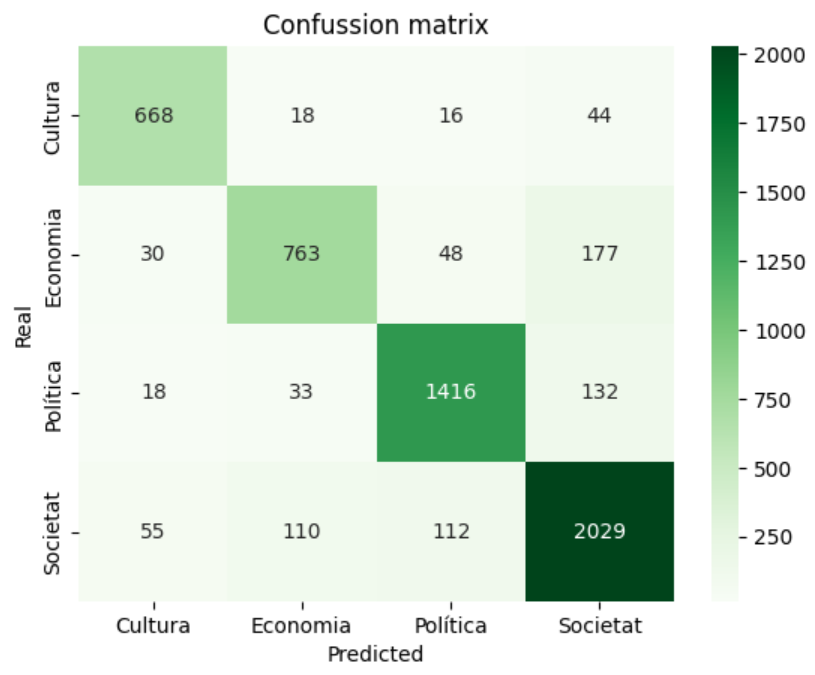
sns.heatmap(cm, annot=True, fmt="d", cmap="Greens", xticklabels=asign.values(), yticklabels=asign.values())

plt.xlabel("Predicted")

plt.ylabel("Real")

plt.title("Confussion matrix")

plt.show()



Podem veure que Societat és la categoria amb més falsos negatius i falsos positius, per tant, seria la menys precisa, però realment això és deu a que és la classe més poblada i en proporció totes les classes obtenen un rendiment semblant.

En resum, hem pogut entrenar un model de classificació de textos a través dels embeddings de GloVE. Hem pogut fer prediccions acurades de les classes així vist en les corbes d’entrenament i en la matriu de confusió.

# 6. CONCLUSIONS

Després d'una anàlisi aprofundida i comparativa de diversos models i tècniques per a la similaritat textual, s'ha confirmat la complexitat inherent a la tasca de modelar la similaritat semàntica del text. La conclusió general és que no hi ha una única "millor" solució aplicable a totes les circumstàncies. La selecció del model òptim depèn més aviat de les necessitats específiques de cada aplicació, que pot incloure factors com el temps de càlcul, el consum de recursos, la interpretabilitat del model i la naturalesa específica de les dades de text.

L'ús d'embeddings de paraules pre-entrenats, com ara Word2Vec i FastText, ha demostrat ser beneficiós per augmentar la correlació del model. Aquesta constatació és consistent tant per a models de vector únic com de seqüència. Tot i això, es va observar que la correlació amb l'embedding era més o menys constant independentment del tipus d'embedding que es feia servir.

D'entre tots els models analitzats, el model RoBERTa fine-tuned ha ressaltat pel seu elevat rendiment en termes de correlació de Pearson per a la modelització completa. Aquest model va demostrar una gran habilitat per encapsular eficaçment la semàntica del text. Per contra, l'encodificació One Hot Encoder, malgrat no ser tan eficaç en la modelització completa, va mostrar una correlació decent amb l'embedding. Aquesta observació és interessant, ja que il·lustra que fins i tot tècniques més simples poden ser útils en certs contexts. No obstant això, cal destacar que, teòricament, la distància cosinus (una mesura de la similaritat entre vectors que es fa servir sovint en aquest context) pot afavorir altres tipus d'embeddings en lloc de l'encodificació One Hot, la qual cosa pot explicar el seu rendiment més baix.

En relació als models de Perceptró Multicapa (MLP), l'anàlisi ha ressaltat la importància de la regularització per evitar el sobreajustament. En particular, es va observar que la regularització podia limitar l'impacte dels embeddings pre-entrenats en la correlació del model, el que és particularment útil en models com l'Adamax, que semblen propensos al sobreajustament.

En general, aquesta recerca posa de manifest la importància de l'elecció i l'ajust adequats dels models i tècniques d'aprenentatge automàtic per a la tasca de la similaritat de text. Les conclusions obtingudes proporcionen un sòlid punt de partida per a futures recerques i desenvolupaments, i constitueixen un pas més en la millora de la comprensió de les millors pràctiques en aquest àmbit.