UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona

Pràctica 2.2 TVD - Reconeixement d'entitats anomenades

Tractament de la Veu i el Diàleg

Grau en Intel·ligència Artificial
Daniel Álvarez 23857151X
Albert Roca 48106974J

13/11/2025

Contents

1 Introducció

En aquesta segona part de la pràctica es tracta la tasca de *reconeixement d'entitats anomenades*. L'objectiu principal és aprendre a identificar i classificar les entitats presents en una oració, és a dir, assignar una etiqueta a cada paraula segons la categoria semàntica a la qual pertany. En el context específic d'un sistema de reserva de vols, aquestes entitats poden correspondre a la ciutat d'origen o destinació, la data i el dia del vol, o altres atributs rellevants del domini.

Tant la classificació d'intencions, estudiada en la primera part de la pràctica, com el reconeixement d'entitats anomenades són components essencials dels sistemes d'intel·ligència artificial. La primera permet determinar la intenció principal de l'usuari, mentre que la segona facilita l'extracció d'informació clau per generar respostes més precises i contextualitzades.

En primer lloc, s'ha fet una anàlisi detallada de les dades per entendre la seva naturalesa i la distribució de les classes, sovint molt desbalancejada. En segon lloc, s'han preparat i preprocesat les dades per fer-les adequades per a l'entrenament de models neuronals. Finalment, s'han dissenyat i entrenat diverses arquitectures, incloent-hi xarxes recurrents (*LSTM*, *GRU*), convolucionals (*CNN*) i (*Transformer*), explorant també tècniques de regularització i ponderació de classes per reduir el sobreajustament i millorar la generalització.

2 Exercici 1

En aquest primer exercici, ens han demanat que carreguem els dos fitxers `train.csv` i `test.csv` de la carpeta `data`. Els fitxers no contenen cap capçalera, de manera que cal especificar-ho en el moment de la lectura. A més, se'ns indica que hem d'agafar les últimes 900 línies del fitxer `train.csv` per tal de crear un conjunt de validació, mantenint la resta de línies com a conjunt d'entrenament.

Per tant, primer hem llegit el fitxer `train.csv` complet i n'hem separat les últimes 900 línies per formar el conjunt de validació. La resta de registres s'han mantingut com a conjunt d'entrenament. De la mateixa manera, hem carregat el fitxer `test.csv` per utilitzar-lo posteriorment en l'avaluació del model.

Després de realitzar aquesta partició, hem comprovat la mida de cada conjunt per assegurar-nos que la separació s'ha fet correctament. Els resultats obtinguts han estat els següents:

- Mida del conjunt d'entrenament: 4078 instàncies
- Mida del conjunt de validació: 900 instàncies
- Mida del conjunt de test: 893 instàncies

3 Exercici 2

- Ens han demanat centrar-nos en la primera i la segona columna dels fitxers i desar, per a cada partició (train, validació i test), les llistes d'oracions i les llistes d'etiquetes corresponents.
- Hem separat la columna 0 com a llista d'oracions i la columna 1 com a llista d'etiquetes per a cadascuna de les tres particions: *train_sentences*, *train_labels*, *val_sentences*, *val_labels*, *test_sentences* i *test_labels*.
- Hem verificat el contingut imprimint un exemple de la partició de train per assegurar la correspondència entre oració i seqüència d'etiquetes.

Exemple (train):

- Oració: `i want to fly from boston at 838 am and arrive in denver at 1110 in the morning`
- Etiquetes: `"O O O O O B-fromloc.city_name O B-depart_time.time I-depart_time.time O O O B-toloc.city_name O B-arrive_time.time O O B-arrive_time.period_of_day"`

4 Exercici 3

En aquest exercici ens han demanat preparar les dades per tal que puguin ser processades pel model de xarxa neuronal. El procés l'hem dividit així:

En primer lloc, hem construït el vocabulari utilitzant exclusivament les oracions del conjunt d'entrenament. Per fer-ho, hem utilitzat la classe `Tokenizer`, configurada perquè no elimini la puntuació i mantingui les majúscules.

En segon lloc, hem convertit totes les oracions de train, test i val en seqüències de nombres enters, on cada nombre correspon a una paraula del vocabulari creat.

En tercer lloc, hem desat la longitud original de cada oració abans d'aplicar el *padding*, ja que aquesta informació ens serà útil durant l'avaluació per ignorar els zeros afegits.

Finalment, per aconseguir que totes les seqüències tinguessin la mateixa mida, hem aplicat *padding* amb zeros al final de cada oració fins a assolir la longitud màxima observada en el conjunt d'entrenament.

Després de completar aquest procés, hem obtingut un vocabulari format per 831 paraules úniques i una longitud màxima de seqüència de 46 paraules. Per exemple, la primera oració del conjunt d'entrenament, un cop transformada, queda representada com un vector de 46 enters, on les darreres posicions corresponen al *padding*:

```
[12, 69, 1, 38, 2, 9, 64, 415, 84, 17, 75, 16, 13,
 64, 493, 16, 4, 36, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

5 Exercici 4

En aquest ens han demanat preparar les etiquetes corresponents a cada paraula de les oracions. El procés inclou comptar totes les etiquetes úniques, codificar-les en valors numèrics, aplicar *padding* per unificar les longituds i convertir-les finalment en vectors *one-hot* per a l'entrenament.

En primer lloc, hem comptat totes les etiquetes del conjunt d'entrenament tenint en compte els prefixos del format BIO/BILOU per separat (com ara `B-`, `I-`, `L-`, `U-`) i assegurant que l'etiqueta `O` fos considerada com una classe pròpia. Això ens serveix per guardar la informació sobre la posició de cada entitat d'una seqüència, ja que no és el mateix que una paraula iniciï o continuï una entitat.

Després, hem codificat les etiquetes amb un identificador numèric mitjançant un *LabelEncoder*, utilitzant exclusivament les classes presents al conjunt d'entrenament per evitar problemes. Durant aquest procés, hem detectat algunes etiquetes a les particions de validació i test que no apareixien a l'entrenament (com ara `B/I-return_date.today_relative`, `B-compartment` o `B-booking_class`) i hem eliminat les oracions que les tenien.

Llavors hem aplicat *padding* a totes les seqüències d'etiquetes, utilitzant l'etiqueta `O`, per tal que totes les oracions tinguessin la mateixa longitud (46 tokens, segons la longitud màxima trobada a l'entrenament).

Finalment, hem convertit les etiquetes numèriques en vectors *one-hot*, de manera que cada token s'expressa com una distribució de probabilitats sobre totes les classes possibles. El resultat final mostra un total de 120 etiquetes úniques (incloent-hi `O`), i les dimensions dels conjunts són les següents: `train` (4078, 46, 119), `validació` (897, 46, 119) i `test` (885, 46, 119). La darrera dimensió correspon a les 119 classes codificades, mentre que el *padding* es representa amb l'etiqueta `O`.

6 Exercici 5

Aquí ens han demanat dissenyar i entrenar un model capaç de reconèixer entitats anomenades a partir de les oracions del conjunt de dades. La idea principal és substituir la capa de *GlobalMaxPooling1D*, utilitzada en la primera part de la pràctica, per una capa de `LSTM`.

Per tal d'entrenar el model, primer hem afegit una classe addicional a les etiquetes, de manera que el sistema pugui diferenciar les paraules reals del *padding* que s'aplica per igualar la longitud de totes les oracions. També hem generat una màscara de pesos que assigna valor zero al *padding*, evitant així que aquestes posicions afectin el càlcul de la pèrdua durant l'entrenament.

L'arquitectura del model és la següent: primer, una capa d'`Embedding` transforma els índexs de les paraules en vectors densos de dimensió 128; a continuació, una capa `LSTM` amb 128 unitats processa la seqüència mantenint una sortida per cada paraula; finalment, una capa densa amb activació `softmax` que assigna una probabilitat a cada classe possible per cada token. L'entrenament s'ha realitzat amb l'optimitzador `Adam` i la funció de pèrdua `categorical_crossentropy`.

Després de l'entrenament, el model s'ha avaluat sobre el conjunt de test, aconseguint una bona coherència entre la sortida esperada i la predicció. La forma final de les prediccions és `(885, 46, 120)`, que correspon respectivament al nombre d'oracions de test, la longitud màxima de les seqüències i el nombre total de classes considerades.

Resultats i anàlisi

Un cop entrenat el model, hem avaluat el seu rendiment sobre el conjunt de test. Com és habitual en tasques de reconeixement d'entitats anomenades (NER), les dades estan fortament desbalancejades: la gran majoria dels tokens corresponen a l'etiqueta `O` (*outside*), mentre que les entitats específiques tenen una presència molt menor. Això provoca que la mètrica d'*accuracy* no sigui del tot representativa, ja que un model podria obtenir un valor alt simplement predint `O` per a tots els tokens. Per tant, hem calculat també la mètrica *F1-score* per a cada classe i la seva mitjana (*macro average*) per obtenir una visió més rea.

Els resultats mostren que el model aconsegueix una *F1 macro* d'aproximadament 0.56, la qual cosa indica un rendiment limitat. El model tendeix a reconèixer correctament les etiquetes més freqüents però falla en la detecció d'entitats menys comunes. Aquest primer model amb LSTM el considerem una línia base sobre la qual millorarem en l'**Exercici 6**.

7 Exercici 6

7.1 Mida dels embeddings

En aquest experiment hem volgut analitzar com la mida dels vectors d'embedding afecta el rendiment del model. Mides petites poden ser insuficients per representar adequadament la informació, mentre que mides massa grans poden provocar sobreajustament i un entrenament més lent, fins a donar errors en memòria local.

Hem provat diferents dimensions d'embedding: **32, 64, 128, 256, 512, 1024, 2048 i 4096**, mantenint fixos la resta de paràmetres del model (una capa `LSTM(128)` i una sortida `TimeDistributed(Dense)` amb activació `softmax`).

Els resultats mostren que el rendiment millora progressivament fins a 512 dimensions, on s'obté el millor resultat amb una *macro-F1* de **0.7693**. Tot i això, valors superiors no aporten millores significatives i fins i tot poden degradar lleugerament la generalització. A més, en proves posteriors amb arquitectures Transformer, embeddings massa grans (com 512 o més) han sigut inestables i *F1* molt baixos en algunes execucions. Per aquest motiu, hem escollit una mida de **256 dimensions**.

Mida de l'embedding	Macro-F1
32	0.3394
64	0.5637
128	0.6572
256	0.7158
512	**0.7693**
1024	0.7096
2048	0.7699
4096	0.7448

Table 1: Comparació del rendiment segons la mida dels embeddings.

7.2 Xarxes convolucionals

En aquest apartat hem mirat l'ús de capes convolucionals per capturar patrons dins les seqüències. Aquestes capes són especialment útils per detectar combinacions curtes de tokens que siguin veïns. Per tal de mantenir la correspondència entre les entrades i les sortides per token, hem dissenyat models que conserven la mateixa longitud de seqüència, afegint o evitant operacions de *pooling* segons el cas.

Hem provat diverses configuracions de convolucions amb un `embedding` fix de **256 dimensions**: nuclis de mida **3**, **5** i **7** sense *pooling*, dues variants amb *MaxPooling* i *AveragePooling* seguides d'*Upsampling*, i una arquitectura **TextCNN** amb tres nuclis paral·lels (3, 4 i 5).

Els resultats mostren que les variants amb *pooling* presenten un rendiment clarament inferior, ja que perden informació posicional essencial per a la predicció per token. Les arquitectures sense *pooling* obtenen un rendiment molt més elevat, destacant especialment la configuració amb nucli de mida **7** (`conv_k7_nopool`), que aconsegueix la millor puntuació amb una *macro-F1* de **0.6471**. Això suggereix que una finestra lleugerament més ampla (set paraules) permet capturar millor el context local sense sacrificar precisió. Per aquest motiu,

aquest model s'ha considerat com la millor configuració convolucional per als experiments següents.

Model	Macro-F1
conv_k3_nopool	0.5984
conv_k5_nopool	0.6115
conv_k7_nopool	**0.6471**
conv_k5_maxpool_up2	0.3056
conv_k5_avgpool_up2	0.2892
textcnn_k345	0.6332

Table 2: Comparació del rendiment de diferents arquitectures convolucionals (embedding=512).

7.3 Xarxes recurrents

En aquesta secció hem experimentat amb diferents arquitectures de xarxes recurrents per capturar les dependències seqüencials dins les oracions. Aquest tipus són útils per a aquesta tasca, ja que permeten que el model tingui en compte el context de les paraules anteriors i posteriors dins la seqüència. Hem provat provat tant amb capes `LSTM` com `GRU`, utilitzant-les de forma unidireccionals i bidireccionals, amb **64** i **128 unitats recurrents**. Totes les configuracions comparteixen un `embedding` de **256 dimensions**.

Els resultats mostren que les versions bidireccionals obtenen millors resultats que les unidireccionals, ja que són capaces d'incorporar informació contextual tant del passat com del futur dins la frase. Els **BiGRU-128** i **BiLSTM-128** són els que presenten les millors resultats, amb una *macro-F1* de **0.7712** i **0.7545**, respectivament.

Posteriorment, hem combinat la millor capa convolucional obtinguda en la secció anterior (`conv_k7_nopool`) amb les capes recurrents per analitzar si millora el resultast. En aquests experiments combinats, la CNN actua com a extractor inicial de característiques, seguida per les capes recurrents encarregades de modelar. Els resultats indiquen que la incorporació de la capa convolucional manté rendiments similars, sense aportar una millora clara respecte a les RNN pures.

Model	Macro-F1 (RNN pura)	Macro-F1 (+CNN conv_k7)
lstm_64	0.4998	0.4480
lstm_128	0.7494	0.7165
bilstm_64	0.7111	0.7563
bilstm_128	**0.7545**	**0.7692**
gru_64	0.6658	0.6415
gru_128	0.7236	0.7185
bigru_64	0.7470	0.7561
bigru_128	**0.7712**	**0.7626**

Table 3: Comparació dels models RNN purs i combinats amb la millor CNN (`conv_k7_nopool`).

7.4 Transformers

En aquesta secció hem implementat diferents arquitectures basades en **Transformers**, amb l'objectiu d'explorar si els mecanismes d'atenció poden millorar el reconeixement d'entitats anomenades. Els Transformers són

models molt potents per capturar relacions de dependència a llarg abast dins d'una seqüència, ja que cada token pot atendre directament a qualsevol altre, sense la limitació seqüencial de les xarxes recurrents.

Per construir els models, hem utilitzat blocs personalitzats amb components d'*Multi-Head Self-Attention* i capes de *Feed Forward* internes, juntament amb una sortida per token amb activació `softmax`. Els paràmetres que hem variat en els experiments són el `num_heads`, la profunditat del model (`depth`) i la mida de la capa feed-forward (`ff_dim`). Hem provat quatre configuracions:

- `trf_base_s_h4_d1_ff512`: 1 bloc Transformer, 4 caps, FF=512 (arquitectura simple).
- `trf_base_m_h4_d2_ff1024`: 2 blocs, 4 caps, FF=1024 (configuració base).
- `trf_base_h_h8_d2_ff1024`: 2 blocs, 8 caps, FF=1024 (major capacitat atencional).
- `trf_base_d_h4_d3_ff1024`: 3 blocs, 4 caps, FF=1024 (més profunditat).

Els resultats obtinguts mostren que els models Transformer aconsegueixen un rendiment raonable però inferior al de les RNN. Concretament, el model amb **2 blocs, 4 caps i FF=1024** és el que obté el millor resultat, amb una *macro-F1* de **0.5338**. Aquest rendiment, tot i ser modest, mostra que el model és capaç d'aprendre certes relacions contextuals, però probablement requereix més dades o una optimització més acurada per superar les RNN. També s'ha observat una variabilitat més alta en els resultats, amb alguns entrenaments que derivaven en valors de *F1* molt baixos (fins a 0.01).

Model	**Macro-F1**
trf_base_s_h4_d1_ff512	0.4882
trf_base_m_h4_d2_ff1024	**0.5338**
trf_base_h_h8_d2_ff1024	0.4790
trf_base_d_h4_d3_ff1024	0.5165

Table 4: Comparació del rendiment dels diferents models Transformer.

7.5 Regularització amb Dropout

Aquí hem analitzat l'efecte de la **regularització** en el comportament del model, amb l'objectiu de reduir l'*overfitting* observat en configuracions passades. Per a l'experiment, hem seleccionat expressament el model amb **pitjor rendiment anterior** (`LSTM-64`), que havia obtingut una *macro-F1* de només **0.49**, per comprovar si la introducció de Dropout podia millorar-ne la generalització.

Hem incorporat el Dropout en tres possibles punts de la xarxa: després de la capa d'`Embedding`, després de la capa recurrent `LSTM`, i després d'una capa densa intermitja projectada per cada token. Això ens ha permès avaluar quina ubicació té un impacte més positiu sobre la capacitat del model per evitar l'overfitting. També hem provat diferents valors de Dropout: **0.05, 0.10, 0.20, 0.40 i 0.60**.

Els resultats mostren que la regularització mitjançant Dropout millora de manera notable el rendiment del model. En concret, el millor resultat s'obté aplicant Dropout només **després de la capa densa** (`postdense_only`) amb una taxa de **0.05**, aconseguint una *macro-F1* de **0.77**. Això suposa una millora

molt significativa respecte al valor original de **0.49**. També destaquen configuracions similars amb taxes de Dropout del 20% o amb Dropout en totes les capes, que ofereixen rendiments comparables.

Configuració	Macro-F1
`postdense_only_drop0.05`	**0.7728**
`postdense_only_drop0.20`	0.7612
`all_on_drop0.10`	0.7498

Table 5: Millors configuracions de Dropout per al model LSTM-64 (embedding=256).

En resum, s'observa que el Dropout és especialment efectiu quan s'aplica després de les capes densament connectades, ja que introdueix variabilitat controlada i redueix l'overfitting sense afectar greument la representació seqüencial.

7.6 Balanceig de classes

En aquest apartat hem analitzat l'efecte del desbalanceig de classes sobre el rendiment dels diferents models. En primer lloc, hem comprovat que el conjunt d'entrenament presenta una distribució molt desequilibrada: més del **75%** dels tokens corresponen al símbol `<pad>` i aproximadament un **15%** a la classe `O`, mentre que la resta d'entitats (com ara `B-toloc.city_name` o `B-fromloc.city_name`) apareixen amb freqüències inferiors a l'1%.

Així doncs, hem calculat pesos per classe amb la funció `compute_class_weight('balanced')`, a partir dels identificadors d'etiqueta del train. Els pesos obtinguts els hem estabilitzat aplicant l'arrel quadrada i limitant-los entre 0.5 i 5.0 per evitar valors extrems. Posteriorment, hem convertit en una matriu de `sample_weight` per token, que hem incorporat al `fit()` per ponderar la pèrdua segons la freqüència de cada classe. A la classe `<pad>` li hem assignat pes zero per tal d'evitar que afecti la funció de cost.

Inicialment, hem aplicat aquest esquema als models basats en *Transformers*, però els resultats no han estat positius: el rendiment global ha empitjorat significativament. En concret, ha baixat fins a valors al voltant de **0.48** i **0.43** segons l'arquitectura.

Posteriorment, hem repetit l'experiment amb arquitectures recurrents bidireccionals, concretament **BiLSTM(128)** i **BiGRU(128)**. En aquest cas, els resultats han estat més estables: la BiLSTM ha millorat lleugerament la seva *macro-F1*, passant de **0.75** a **0.79**, mentre que la BiGRU ha empitjorat una mica de **0.76** a **0.74**.

Model	Macro-F1 amb pesos
`trf_base_m_h4_d2_ff1024`	0.4898
`trf_base_h_h8_d2_ff1024`	0.4327
`bilstm_128_emb256+weights`	**0.7985**
`bigru_128_emb256+weights`	0.7443

Table 6: Rendiment (macro-F1) amb balanceig de classes aplicat com a *sample_weight* per token. El balanceig millora lleugerament la BiLSTM, però empitjora la BiGRU i els Transformers.

8 Conclusions

Al llarg d'aquest treball hem desenvolupat i analitzat diferents arquitectures per a la tasca de reconeixement d'entitats (NER) a nivell de token. Hem partit d'una representació bàsica de les oracions i etiquetes, i hem anat incrementant progressivament la complexitat dels models per tal d'avaluar com diferents estratègies afectaven el rendiment. Hem provat amb xarxes convolucionals (CNN), xarxes recurrents (LSTM i GRU unidireccionals i bidireccionals), *dropout* i, finalment, arquitectures basades en *Transformers*.

Els resultats han mostrat una evolució. Les primeres CNN han aconseguit resultats raonables, però limitats. Les RNN, i en particular les bidireccionals, han millorat notablement el rendiment gràcies a la seva capacitat per capturar informació tant del passat com del futur dins la seqüència. Dins d'aquest grup, el model **BiLSTM(128)** amb embeddings de 256 dimensions ha estat el que ha obtingut els millors resultats globals, assolint una puntuació *macro-F1* propera a **0.80**. Aquest model equilibra bé la capacitat de generalització i la complexitat, mostrant-se robust davant l'overfitting i estable entre les diferents particions del conjunt de dades.

També hem mirat tècniques de regularització, com el *dropout* aplicat després de la capa densa, que han ajudat a reduir l'overfitting i a millorar el *macro-F1* de models simples (de valors propers a 0.49 fins a 0.77). Tot i això, l'ús de pesos de classe per baslanceijar dataset no ha resultat especialment beneficiós: en els *Transformers* ha provocat una pèrdua significativa de rendiment, i en les xarxes recurrents només ha tingut un efecte positiu en la BiLSTM, mentre que la BiGRU fins i tot ha empitjorat lleugerament.

En conjunt, podem concloure que les xarxes recurrents bidireccionals són una opció sòlida per a tasques de NER quan el conjunt de dades és moderadament petit i desbalancejat. Els *Transformers*, malgrat el seu potencial, requereixen més dades i ajustos més fins per superar-los en aquest context. Finalment, hem comprovat que tècniques senzilles com la regularització amb *dropout* o l'ús adequat d'embeddings poden tenir un impacte molt més positiu que l'increment de complexitat del model.