Identificació d'idioma

Pau Hidalgo Pujol i Cai Selvas Sala

4 de març de 2024

Universitat Politècnica de Catalunya
Grau en Intel·ligència Artificial
Processament del Llenguatge Humà

Resum

En aquest informe s'explica el desenvolupament que s'ha fet de la pràctica d'identificació d'idiomes 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

Identificar l'idioma en el qual està escrit un text és una tasca bàsica del Processament del Llenguatge Natural, permetent classificar textos, analitzar-los i facilitant altres problemes més complexos com la traducció.

En aquest treball s'analitzarà l'ús de trigrames de caràcters com a eina d'identificació de la llengua. Usant dades provinents de *Wortschats Leipzig Corpora* [1] en espanyol, italià, neerlandès, anglès, alemany i francès, s'intentarà obtenir un model robust per detectar cadascuna d'aquestes llengües. Addicionalment, es compararan tres tècniques de *smoothing* diferents, analitzant el seu rendiment.

Els resultats s'avaluaran no només numèricament, sinó també observant en quines frases s'equivoca per tal de determinar quins patrons afecten de forma negativa a la predicció.

2 Documents i estructura general de l'entrega

A part d'aquest document, l'entrega conté diversos altres fitxers. En distingim dos tipus: els de codi (*.py* i *.ipynb*) i els de dades (*.txt* i *.json*).

Els primers contenen el codi utilitzat pel model. En el fitxer *train.py* es realitzen totes aquelles tasques relacionades amb l'entrenament del model, com seria el recompte de trigrames únics, el preprocessament dels texts i la separació de les diverses particions. Aquests resultats es guarden en fitxers *.json*, fet que permet no haver d'executar aquest programa cada vegada. En el fitxer *functions.py* hi ha les funcions de suavitzat programades en Python, així com una que s'encarrega de calcular les probabilitats d'una frase. Finalment, el fitxer *main.ipynb* és el principal. Aquest comença llegint els fitxers generats pel *train.py*. A continuació, conté els codis necessaris per executar la validació i el test.

Els fitxers de dades contenen diversos preprocessaments dels texts inicials. D'entrada observem, a la carpeta *preprocessed_langid*, dos fitxers *.json train* i *test*. Contenen, per cada idioma (que funcionen com a clau), els texts preprocessats (en el cas de *test*, una llista de les frases preprocessades). A *weights* hi ha tots aquells fitxers relacionats amb el model en si. Modificar-los afectaria el comportament d'aquest. Els fitxers de *test* i *validation_sent* contenen el nombre de trigrames de cada frase precalculat. Aquest fet agilitza molt el càlcul de les probabilitats més endavant. Els de *trigrams* i *validation_trigrams* són els que contenen, per cada llengua, el nombre d'aparicions de cada trigrama. Finalment, els de *unique_chars* i *validation_unique_chars* contenen simplement el nombre de caràcters únics de cada idioma, per calcular la B.

Aquesta és l'estructura dels documents de l'entrega. Per tal de facilitar la col·laboració, l'historial i tenir els fitxers ben ordenats, s'ha usat la plataforma *GitHub*.

3 Funcions de suavitzat (*smoothing*)

Tal com comenta la teoria, el *Maximum Likelihood Estimator* no és útil en programes de processament del llenguatge humà a causa del fet que poden aparèixer combinacions no vistes anteriorment (i això provocaria que les probabilitats de la frase fossin 0 quan no hauria de ser així). És per això que s'usen funcions de suavitzat, que reparteixen certa probabilitat entre els valors no observats. A continuació, s'explicaran per sobre els que s'han provat i les implementacions.

3.1 *Lidstone*

La funció de suavitzat de Lidstone (també anomenada *additive smoothing*, i en cas que lambda sigui 1, de Laplace) té la següent fórmula:

$$P_{LID}^{[trigram]} = \frac{C_T(trigram) + \lambda}{N_T + B\lambda}$$

On C_t és la quantitat de vegades que apareix el trigrama en les dades d'entrenament, N_t la quantitat de trigrames vistos, λ un paràmetre de regulació i B el nombre total de trigrames possibles. Els valors de les dues primeres variables es poden calcular, però els altres s'hauran d'escollir. Més endavant, durant la validació, es comentaran quins s'han acabat triant i per què.

La implementació en Python de la funció de *Lidstone de suavitzat* és bastant senzilla.

```
1 def lidstone_smooth(lambda_value: float, total_trigrams: int,
     trigram_counts: dict, b_value: int, trigram: tuple):
2     number = trigram_counts.get(trigram, 0)
3     probs = (number + lambda_value) / (total_trigrams + lambda_value
          * b_value)
4     return probs
```

Té més paràmetres que els dos comentats abans, ja que d'aquesta forma es pot usar amb models diferents, fet útil durant el procés de validació. A més, aquesta funció es crida moltes vegades i per estalviar càlculs s'han utilitzat diccionaris amb alguns dels valors ja calculats (total_trigrams). Aquesta decisió reduïa el temps d'execució en aproximadament el 25%.

3.2 *Absolute Discounting*

Una alternativa ala funció de Lidstone és l'*Absolute Discounting*. Aquesta, matemàticament, es pot expressar així:

$$P_{ABS}^{[trigram]} = \begin{cases} \frac{C_T(trigram)-\delta}{N_T}, & \text{if } C_T(trigram) > 0 \\ \frac{(B-N_0)\delta/N_0}{N_T}, & \text{otherwise} \end{cases}$$

On N_0 són els trigrames possibles observats 0 vegades, i la resta de símbols els mateixos esmentats anteriorment.

Utilitza un paràmetre delta(δ, que resta al recompte d'aparicions del trigrama (en el cas que s'hagi observat en l'entrenament). En cas contrari, l'operació que fa és l'equivalent a ajuntar totes les deltes dividides pel nombre total de trigrames (les probabilitats apartades) i repartir-les entre el nombre de trigrames no vistos. Aquesta resta és interessant perquè tindrà un efecte gran en els trigrames que apareixen pocs cops (poc comuns) i un efecte més petit en els que apareixen molts.

Igual que l'anterior, també necessita un paràmetre regulador (en aquest cas delta), la B del llenguatge i, addicionalment, la quantitat de trigrames únics observats durant l'entrenament (per poder obtenir els que no s'han observat). En Python, és tal que:

```
def absolute_discounting(alpha: float, total_trigrams: int, trigram_
    counts: dict, b_value: int, trigram: tuple):
    count_trigram = trigram_counts.get(trigram, 0)
    unique = len(trigram_counts)
    if count_trigram == 0:
        prob = (unique * alpha / (b_value - unique)) / total_
            trigrams
    else:
        prob = ((count_trigram - alpha) / total_trigrams)
    return prob
```

Observar que és lleugerament diferent, ja que enlloc d'usar B - N_0, usa directament la quantitat de trigrames observats.

3.3 *Linear Discounting*

Aquesta funció és conceptualment similar a l'*Absolute Discounting* en el sentit que resta certa probabilitat dels trigrames observats (discounting). Tot i això, ho fa de manera diferent: en lloc de restar del recompte, resta una probabilitat directament. Matemàticament és:

$$P_{ABS}^{[trigram]} = \begin{cases} (1-\alpha)\frac{C_T(trigram)}{N_T}, & \text{if } C_T(trigram) > 0 \\ \frac{\alpha}{N_0}, & \text{otherwise} \end{cases}$$

En aquest cas, el paràmetre és *alpha* (α i és fàcil veure com funciona. En el cas que aparegui el trigrama en el train, la probabilitat final simplement es multiplica per 1-alpha. En cas que no, la probabilitat és *alpha* entre el nombre de trigrames no observats. Altre cop, la implementació en Python usa els mateixos arguments per a assegurar-se compatibilitat.

```
    def linear_discounting(alpha: float, total_trigrams: int,
        trigram_counts: dict, b_value: int, trigram: tuple):
    count_trigram = trigram_counts.get(trigram, 0)
    if count_trigram == 0:
        prob = alpha / (b_value - len(trigram_counts))
    else:
```

```
        prob = (1-alpha)*(count_trigram / total_trigrams)
    return prob
```

3.4 Altres funcions de suavitzat

Existeixen més funcions de suavitzat diferents: *Knesser-Ney, Katz*... Aquestes utilitzen conceptes més complexos (interpolació, *backoff*) i probablement permetrien obtenir un millor rendiment, però queden fora l'abast d'aquesta pràctica.

Un altre fet a comentar és originalment es va implementar alguna funció, pensant que era una de les anteriors, de forma errònia. Tot i estar malament matemàticament, els resultats assolits usant-les eren millors que si es feien servir les funcions correctes. Per al resultat final no s'han tingut en compte, ja que com a funcions de probabilitat són incorrectes (no estan normalitzades, per exemple), però són uns resultats interessants.

3.5 Funcions extra

Amb les funcions de suavitzat es poden calcular les probabilitats d'un trigrama, però cal una altra per calcular les probabilitats d'una frase entera.

Al ser probabilitats molt petites, sorgeix un problema a l'hora d'ajuntar-les: la multiplicació dona valors molt propers a zero (fins al punt que un ordinador no pot guardar-los). Per evitar-ho, en lloc de calcular la multiplicació de les probabilitats s'ha realitzat el sumatori dels logaritmes de les probabilitats. D'aquesta manera, s'evita tenir problemes numèrics.

La funció en Python encarregada de realitzar aquest càlcul és la següent:

```
def probs_total(b_value: int, text: str, model: dict, total_trigrams
    : int, smooth: Callable = lidstone_smooth, param: float = 0.5,
    probabilities: Optional[dict] = None):
    trigram_finder = TrigramCollocationFinder.from_words(text)
    prob_sec = 0
    if probabilities == None:
        for trigram, num_instances in trigram_finder.ngram_fd.items
            ():
            prob_sec += num_instances * math.log(smooth(param,
                trigram=trigram, b_value=b_value, total_trigrams=
                total_trigrams, trigram_counts=model))
    else:
        for trigram, num_instances in trigram_finder.ngram_fd.items
            ():
            prob_sec += num_instances * probabilities.get(trigram,
                math.log(smooth(param, trigram=trigram, b_value=b_
                value, total_trigrams=total_trigrams, trigram_counts
                =model)))
```

```
    return prob_sec
```

Obté tots els trigrames de la frase i seguidament suma els logaritmes de les seves probabilitats.

Per millorar els temps d'execució, també s'ha creat una funció que calcula totes les probabilitats dels trigrames i les guarda en un diccionari:

```
def create_prob_dict(b_value_dict: dict = b_values, smooth: Callable
    = lidstone_smooth, param: float = 0.5, model: dict = trigrams,
   total: dict = totals, unique: set = unique_trigrams):
    d = {}
    for lang in languages.keys():
        probab = {tri: math.log(smooth(param, trigram=tri, b_value=b
           _value_dict[lang], trigram_counts=model[lang], total_
           trigrams=total[lang])) for tri in unique}
        probab["None"] = math.log(smooth(param, trigram=(1,1,1), b_
           value=b_value_dict[lang], trigram_counts=model[lang],
           total_trigrams=total[lang]))
        d[lang] = probab
    return d
```

4 Preparació de dades

Les dades utilitzades (obtingudes en [1]) tenen aproximadament 40000 frases per cada un dels 6 idiomes que s'han considerat (anglès, castellà, alemany, italià, francès i neerlandès). Aquestes frases s'han dividit en dos conjunts: un amb aproximadament 30000 per l'entrenament (i validació) i l'altre amb les 10000 frases restants pel test.

Per tal d'obtenir millors models en l'entrenament, s'ha realitzat un preprocessament de les frases que s'explica a continuació.

4.1 Preprocessament

El preprocesssament realitzat a les frases s'ha fet en la funció `preprocess_text()` i s'ha basat principalment en la llibreria `re` de Python, que permet treballar amb expressions regulars i poder identificar (i eliminar) caràcters, patrons o seqüències de caràcters en textos. Aquest preprocessament s'ha realitzat en l'arxiu `train.py` abans de l'entrenament dels models i consisteix principalment en el següent:

- Separar el text original en les diferents frases (separant segons el caràcter "$\backslash n$", degut a que les frases es trobaven separades per línies en l'arxiu original).
- Convertir cada frase a text totalment en minúscules.
- Eliminar tots els dígits de les frases, ja que no aporten cap indicador per distingir un idioma d'un altre.
- Eliminar tots els espais duplicats de les frases i substituir-los per espais únics.
- Eliminar els espais al principi i final de les frases.
- Finalment, concatenar totes les frases deixant un espai doble entre elles. Aquest doble espai, com que posteriorment es treballarà amb trigrames de caràcters, permet que mai hi hagi dos caràcters de frases diferents en un mateix trigrama, a més de permetre als models entendre millor on comença i acaba cada frase.

Listing 1: Codi de la funció per preprocessar el text original

```
def preprocess_text(text: str) -> str:
        new_text = ""
        sents = text.split('\n')
        for sent in sents:
                sent = sent.lower() # Lowercase
                sent = re.sub(r'\d', '', sent) # Remove digits
                sent = re.sub(r'\s+', ' ', sent) # Remove extra
                   spaces
                sent = sent.strip() # Remove leading and trailing
                   spaces
```

```
9                   if sent:
10                          new_text += sent + "  " # Add two spaces to
                                separate sentences
11          return new_text
```

4.2 Emmagatzematge de dades

Una vegada s'ha realitzat el preprocessament de les frases, s'han guardat les guardat els textos preprocessats els arxius *train.json* i *test.json* (per les particions de d'entrenament i test, respectivament) en el directori */preprocessed_langId*. Els textos d'entrenament s'han guardat sencers per cada idioma, mentre que els de test s'han guardat per cada idioma totes les frases ja separades, degut a que la predicció es farà individualment per cada una d'elles.

Aquest emmagatzematge dels textos preprocessats permet estalviar temps en les següents d'execucions que es realitzin. És a dir, només caldrà llegir els textos ja preprocessats dels arxius *.json*, evitant cridar de nou a la funció *preprocess_text()*.

5 Entrenament

5.1 Partició i preparació

La partició d'entrenament han sigut 30.000 frases de cada un dels 6 idiomes: alemany, anglès, francès, italià, neerlandès i espanyol.

Per tal de preparar aquestes frases, s'ha realitzat el preprocessament comentat en l'apartat anterior.

5.2 Mètode d'entrenament

El mètode d'entrenament d'aquest model, si es pot dir així, és contar el nombre d'aparicions de cada trigrama en el corpus de cada llenguatge. Per fer-ho, s'ha usat *TrigramCollocationFinder* de la llibreria NLTK. Aquest ha permès obtenir les freqüències de cada trigrama de manera molt senzilla.

Aquestes freqüències s'han guardat en un diccionari, on les claus eren els trigrames i els valors el nombre d'aparicions. Mencionar que és en aquest pas on s'ha realitzat l'últim requisit de preprocessat: l'eliminació dels trigrames que apareixen menys de 5 cops.

Al seu torn, aquests diccionaris de freqüències s'han guardat en un diccionari on cada clau és un identificador de país. D'aquesta manera es pot accedir als valors molt més fàcilment.

Al mateix temps, s'ha guardat el nombre de caràcters únics de cada llenguatge.

5.3 Emmagatzematge dels models entrenats

Per tal de no haver d'executar aquesta part d'entrenament cada cop, les freqüències (el diccionari esmentat) es guarden en un fitxer en format *.json*. Aquest permet guardar el diccionari fàcilment, i llegir-lo quan calgui amb poques línies.

Per poder guardar el diccionari correctament s'han creat funcions extres, ja que no permetia trigrames com a claus.

6 Validació

6.1 Partició i preparació

Per la partició del conjunt de validació s'ha escollit utilitzar un terç del conjunt d'entrenament inicial. És a dir, de les 30000 frases aproximades d'entrenament inicial per cada idioma, 20000 s'han mantingut com a entrenament i les 10000 restants com a validació.

Per seleccionar aquest terç, s'han partit totes les frases i s'han reordenat (`shuffle`) per evitar biaixos. Un cop fet això, s'hi ha aplicat el mateix preprocessament que a l'entrenament (train) i que al test, i s'ha guardat en dos fitxers nous: *validation_trigrams* i *validation_unique_chars*. El segon tan sols conté el nombre de caràcters únics vist, ja que s'ha usat per determinar la B. El primer conté un diccionari amb tantes entrades com llengües. Per cada una, hi ha una llista on el primer element és un diccionari amb el recompte d'aparicions de cada trigrama, i el segon el test.

6.2 Paràmetres

La validació dels models s'ha decidit realitzar per determinar la millor funció de suavitzat (*Lidstone Smooth*, *Linear Discounting* i *Absolute Discounting*) i el seu millor paràmetre (*lambda* en cas de *Lidstone Smooth* i *alpha* en els altres dos). Els valors de paràmetres que s'han provat són 0.001, 0.1, 0.25, 0.35, 0.5, 0.65, 0.75 i 0.9 (tots entre 0 i 1, per tal de que siguin vàlids). Per cada una de les funcions de suavitzat es provaran tots els valors mencionats per obtenir els resultats i determinar quina és la millor combinació.

Per altra banda, també s'ha utilitzat la validació per determinar l'estimació del nombre de trigrames potencialment observables (paràmetre B en les funcions de suavitzat). Les opcions que s'han provat per el paràmetre B són les següents:

- ***B* màxima segons caràcters únics**: ${U_{max}}^3$, on U_{max} és el màxim nombre de caràcters únics que algun dels idiomes té en el conjunt d'entrenament (elevat al cub perquè treballem amb trigrames). És a dir, representa totes les possibles permutacions (amb repetició) de longitud 3 que es poden fer amb U_{max} caràcters diferents. Aquest valor és, amb molt alta probabilitat, una fita superior del valor ideal de B, de manera que és vàlid però pot no estar gaire a prop de l'estimació ideal (ja que hi ha una gran quantitat de permutacions que de ben segur no existeixen en cap dels idiomes). Tenint en compte el preprocessament que s'ha fet en els textos originals (explicat en l'apartat 4.1): $U_{max} = 84 \Rightarrow B = 592704$.
- ***B* mínima segons caràcters únics**: ${U_{min}}^3$, on U_{min} és el mínim nombre de caràcters únics que algun dels idiomes té en el conjunt d'entrenament (elevat al cub perquè treballem amb trigrames). És a dir, representa totes les possibles permutacions (amb repetició) de longitud 3 que es poden fer amb U_{min} caràcters diferents. Aquest valor és probablement una fita inferior del valor ideal de B, de manera que no podem assegurar que sigui un valor vàlid (pot haver permutacions que existeixen en els idiomes i no es comptabilitzen amb aquest valor de B). Tenint en compte el preprocessament que s'ha realitzat en els textos originals (explicat en l'apartat 4.1): $U_{min} = 59 \Rightarrow B = 205379$.

- ***B* mitjana segons caràcters únics**: ${U_{avg}}^3$, on U_{avg} és el nombre mitjà de caràcters únics que els idiomes tenen en el conjunt d'entrenament (elevat al cub perquè treballem amb trigrames). És a dir, representa totes les possibles permutacions (amb repetició) de longitud 3 que es poden fer amb U_{min} caràcters diferents. Aquest valor és probablement el més proper al valor ideal de B, ja que es troba entre la fita superior de B *màxima* i la probablement fita inferior B mínima. Tenint en compte el preprocessament que s'ha realitzat en els textos originals (explicat en l'apartat 4.1): $U_{avg} = 74 \Rightarrow B = 405224$.

- ***B* teòrica segons caràcters únics**: $26^3 = 17576$, on el 26 representa el nombre de caràcters de l'alfabet anglès. És a dir, aquest valor és el mateix per tots els idiomes i representa el nombre de possibles permutacions (amb repetició) de longitud 3 que es poden formar amb les diferents lletres de l'abecedari anglès. Aquesta estimació és bastant arbitrària i segurament no s'acosti gaire al valor ideal de B, ja que no considera en cap moment altres símbols a part de les lletres de l'alfabet anglès.

- ***B* variable segons caràcters únics**: ${U_i}^3$, on U_i és el nombre de caràcters únics per cada idioma $i \in \{$Anglès, Castellà, Alemany, Italià, Francès, Neerlandès$\}$. És a dir, aquest valor de B és diferent per cada un dels idiomes i representa el nombre de possibles permutacions (amb repetició) de longitud 3 que es poden fer amb els diferents de cada idioma vistos en l'entrenament. Aquest valor és probablement una fita superior per tots els idiomes; ja que, tal i com succeeix amb la B màxima segons caràcters únics, probablement hi hagi molts trigrames que es comptabilitzen en aquesta estimació de B que no existeixen realment en l'idioma. Tenint en compte el preprocessament que s'ha realitzat en els textos originals (explicat en l'apartat 4.1), els valors són els següents:

  - Alemany: 357911.
  - Anglès: 205379.
  - Francès: 592704.
  - Italià: 493039.
  - Neerlandès: 493039.
  - Castellà: 405224.

- ***B* segons total de trigrames observats**: $\sum_{i \in I} T_i$, on T_i és el nombre de trigrames únics observats durant l'entrenament en l'idioma $i \in I = \{$Anglès, Castellà, Alemany, Italià, Francès, Neerlandès$\}$. Aquest valor és el mateix per tots els idiomes. Tenint en compte el preprocessament que s'ha realitzat en els textos originals (explicat en l'apartat 4.1), el valor d'aquesta estimació és 21773.

- ***B* segons trigrames observats per idioma**: $T_i + 500$, on T_i és el nombre de trigrames únics observats durant l'entrenament en l'idioma $i \in I = \{$Anglès, Castellà, Alemany, Italià, Francès, Neerlandès$\}$. T_i és diferent per cada idioma i es pot garantir que és una cota inferior de la B ideal (ja que són tots els valors observats en l'entrenament, de manera que en el l'idioma realment ha d'haver com a mínim aquests trigrames de caràcters). Es suma 500 al valor de T_i per tenir en compte que hi ha valors que no s'han vist a l'entrenament (s'assumeix que com a molt n'hi haurà 500 més, no vistos en l'entrenament) i també per tal d'evitar errors en els càlculs de probabilitat en les funcions de *Linear Discounting* i *Absolute Discounting*. Tenint en compte el preprocessament que s'ha realitzat en els textos originals (explicat en l'apartat 4.1), els valors són els següents:

- Alemany: 11821.
- Anglès: 9890.
- Francès: 10921.
- Italià: 8825.
- Neerlandès: 10648.
- Castellà: 9794.

Cal mencionar que la majoria d'aquestes estimacions de B depenen del nombre de caràcters únics i, per tant, del preprocessament (ja que és on es poden eliminar certs caràcters).

6.3 Mètriques

Com a mètrica s'ha decidit usar el nombre d'errors totals. Es guarda una llista amb tots els errors, o per tal d'en cas de detectar un comportament estrany poder veure idioma per idioma on fallava o fer servir altres tipus de mètriques.

La quantitat total d'errors, en aquest cas, és la mètrica més fàcil de visualitzar i entendre, ja que en la resta s'obtenen valors molt alts, fins al punt d'haver de llegir fins al quart decimal per determinar si és millor.

6.4 Resultats

Una vegada s'ha executat la validació, s'ha generat un gràfic per cada una de les estimacions de B (figures 1, 2, 3, 4, 5, 6, 7) mencionades anteriorment. En línies generals, el *Linear Discounting* acostuma a ser el millor (ha obtingut el mínim d'errors en 4 dels 7 valors de B), seguit del *Lidstone Smooth* (2 de 7) i, finalment, l'*Absolute Discounting* (1 de 7). El millor de tots els resultats (combinació de paràmetres amb menys errors en la validació) s'ha obtingut amb el *Linear Discounting* amb paràmetre $alpha = 0.35$ i amb estimació de B mitjana segons caràcters únics, amb un total de només 77 errors tal i com es pot veure en la figura 3.

Per tant, les prediccions en la partició de test es realitzaran amb els millors paràmetres obtinguts en aquesta validació: *Linear Discounting* amb $alpha = 0.35$ i amb estimació de B mitjana segons caràcters únics ($B = 74^3 = 405224$).

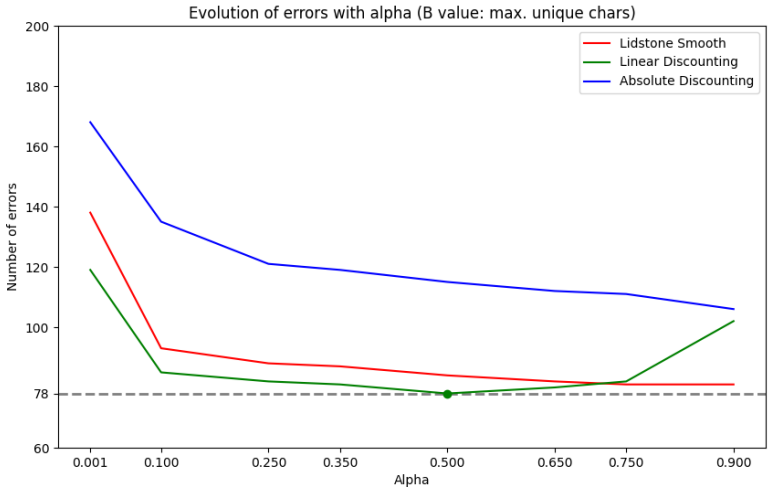

Figura 1: Gràfic del nombre d'errors en funció del paràmetre *alpha* per cada una de les funcions de suavitzat (amb B màxima segons caràcters únics)

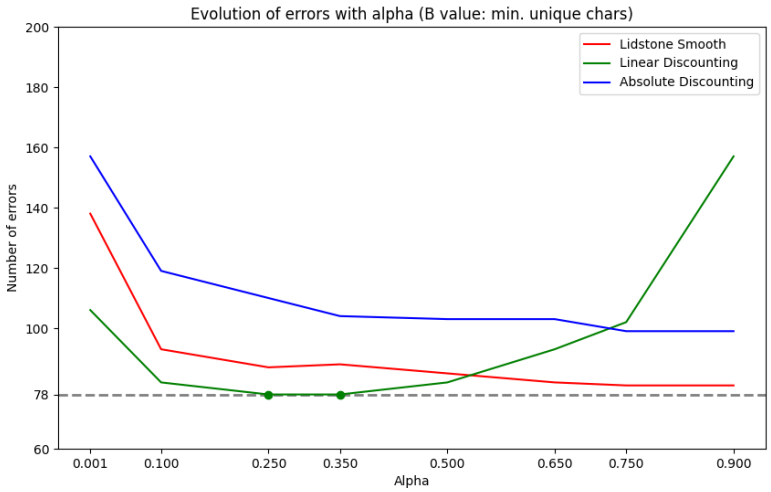

Figura 2: Gràfic del nombre d'errors en funció del paràmetre *alpha* per cada una de les funcions de suavitzat (amb B mínima segons caràcters únics)

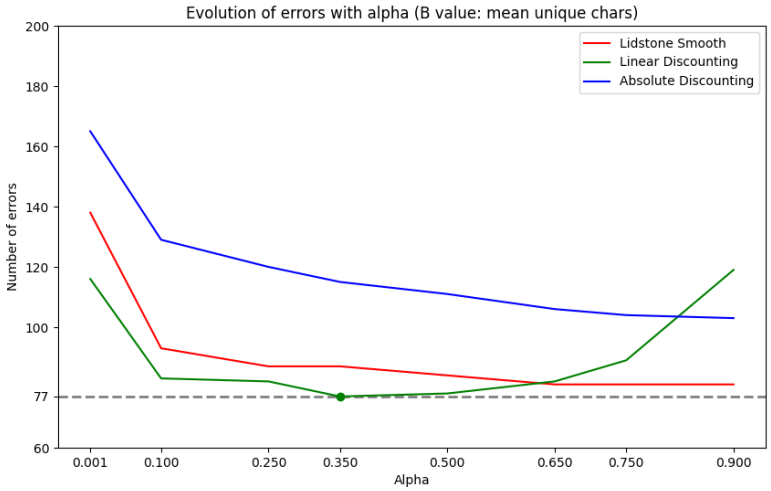

Figura 3: Gràfic del nombre d'errors en funció del paràmetre *alpha* per cada una de les funcions de suavitzat (amb B mitjana segons caràcters únics)

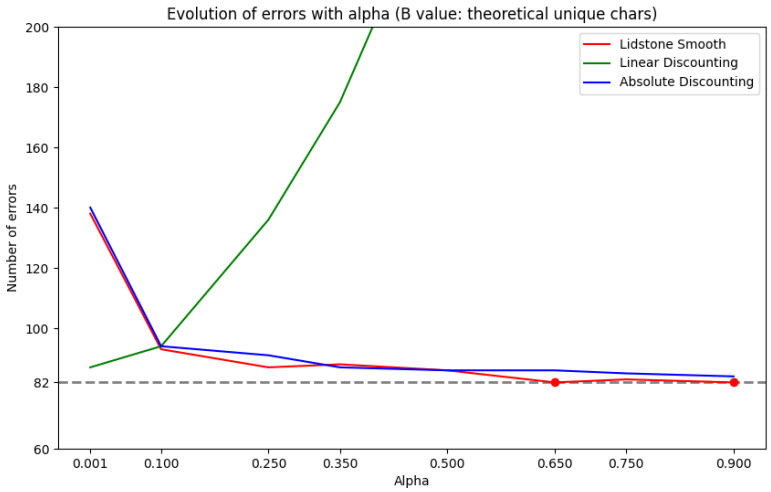

Figura 4: Gràfic del nombre d'errors en funció del paràmetre *alpha* per cada una de les funcions de suavitzat (amb B teòrica segons caràcters únics)

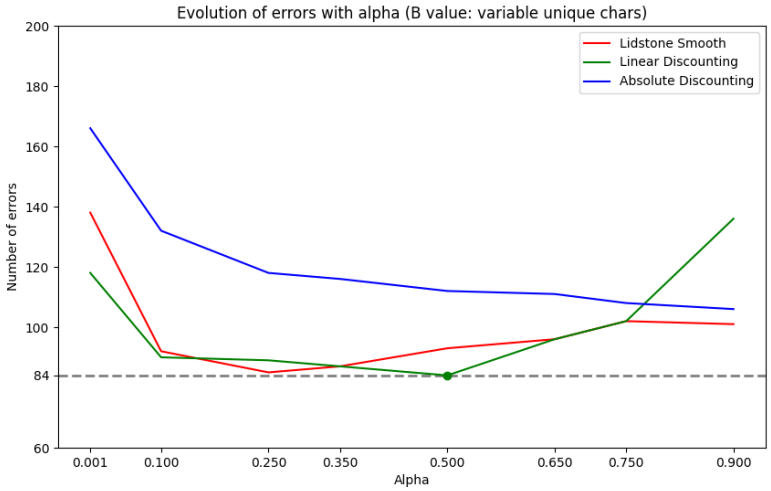

Figura 5: Gràfic del nombre d'errors en funció del paràmetre *alpha* per cada una de les funcions de suavitzat (amb B variable segons caràcters únics)

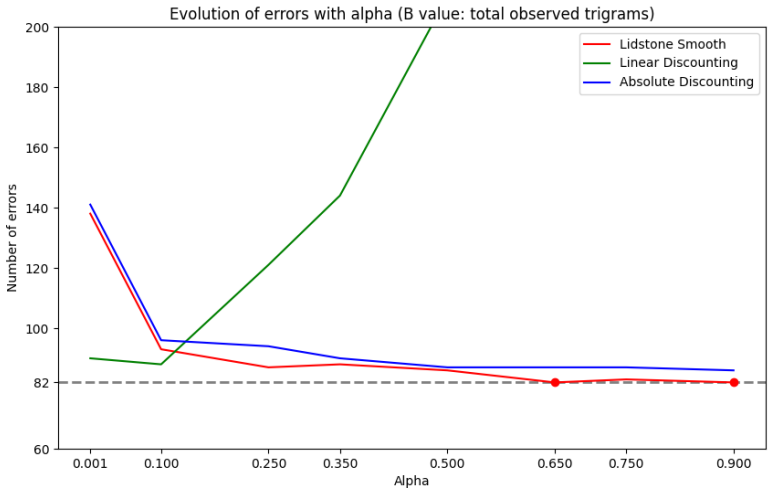

Figura 6: Gràfic del nombre d'errors en funció del paràmetre *alpha* per cada una de les funcions de suavitzat (amb B segons total de trigrames únics observats)

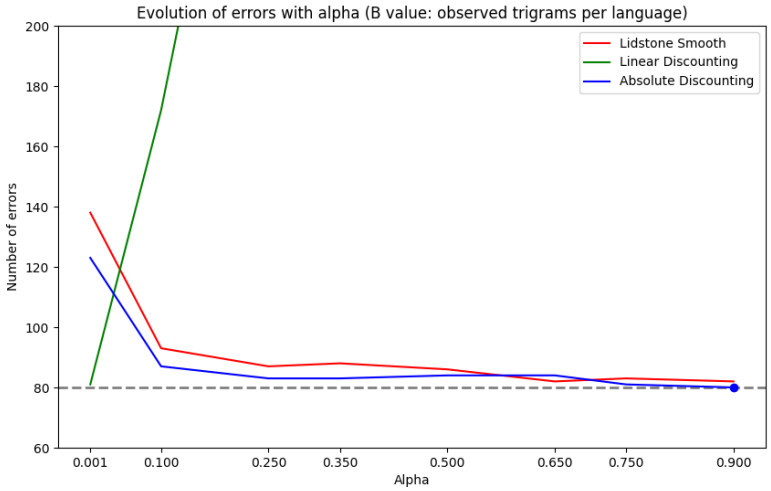

Figura 7: Gràfic del nombre d'errors en funció del paràmetre *alpha* per cada una de les funcions de suavitzat (amb B segons total de trigrames únics observats per idioma)

7 Test i resultats

7.1 Partició i preparació

La partició de test que s'ha utilitzat és la mencionada a l'enunciat de la pràctica, les 10.000 frases per cada un dels idiomes. Per tal de poder usar aquestes dades correctament, s'hi ha aplicat el mateix preprocessament que a les dades de train. L'única diferència és que, a diferència de les frases d'entrenament que anaven totes seguides, ha calgut separar-les per tal de poder predir l'idioma d'elles per separat.

Aquest test preprocessat s'ha guardat en un fitxer *.json* igual que els texts d'entrenament.

7.2 Execució

Per tal de dur a terme el test, s'ha iterat per les frases dels tests de tots els llenguatges. Per cada frase, s'han calculat les probabilitats de cada idioma, quedant-se amb la més elevada. Si no coincideix amb la llengua esperada, s'afegia la frase, així com l'idioma esperat i el predit, en una llista. Aquesta llista ha permès analitzar els resultats i observar en quines frases el model s'ha equivocat.

Per tal de calcular les probabilitats, s'han usat dos bucles per iterar per totes les frases, i per cada una s'ha iterat per tots els idiomes. Per millorar els temps d'execució, les probabilitats han estat precalculades i guardades en un diccionari. A la funció que calcula les probabilitats se li han passat tots els paràmetres trobats en la validaicó.

Listing 2: Codi usat en l'execució del test

```
for language in language_list:
        for sentence in pre_trigrams[language]:
                probs = []
                for lingua in language_list:
                        prob_sec = 0
                        for trigram, num_instances in sentence:
                                prob_sec += num_instances * prob_
                                    dict[lingua].get(trigram, prob_
                                    dict[lingua]["None"])
                        probs.append((lingua, prob_sec))
                max_prob = max(probs, key = lambda x: x[1])
                predictions[language].append(max_prob[0])
                if max_prob[0] != language:
                        sentence = ''.join([t[0][0] for t in
                            sentence]).join(sentence[-1][0][1:])
                        prediction_errors_list.append({"text":
                            sentence, "true": language, "pred": max_
                            prob[0]})
```

7.3 Resultats

Un cop realitzada l'execució, s'han analitzat els resultats.

Language	Accuracy
German (deu)	0.9994
English (eng)	0.9997
French (fra)	0.9989
Italian (ita)	0.9986
Dutch (nld)	0.9979
Spanish (spa)	0.9991
Total Accuracy	0.9989

Taula 1: Language Identification Accuracy

El model ha fet prediccions errònies en 64 frases. L'*accuracy* total és de 99.8932 %. Desglossant per llengua, la que encerta més és l'anglès, amb un 99.969%, mentre que la pitjor el neerlandès (99.79%).

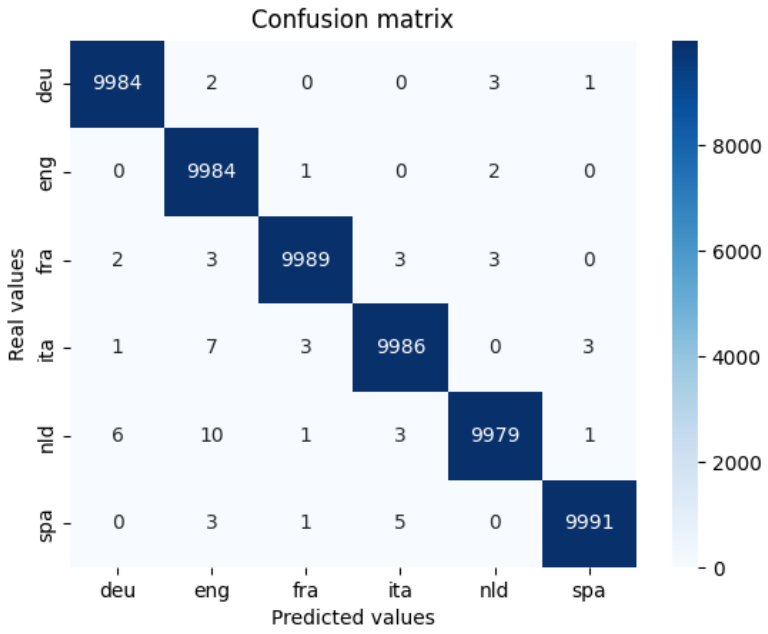

Figura 8: Matriu de confusió

Analitzar els resultats de la matriu de confusió és interessant. L'idioma que classifica malament més cops és el neerlandès, equivocant-se un total de 21 cops. D'aquests errors, 10 són predient anglès i 6 alemany. Aquests resultats tenen sentit, ja que són llengües molt similars.

En el cas del castellà, en 5 de les frases creu que és italià. Totes dues són llengües romàniques, amb arrels similars i que, per tant, comparteixen algunes paraules o morfemes. És per això que és normal que en aquests casos s'hagi equivocat. Un resultat més estrany és el de l'italià, que en 7 dels casos el prediu com a anglès. Analitzant les frases en les quals s'equivoca, podem observar per què: moltes contenen paraules en anglès, en alguns casos arribant a tenir-ne més que en italià. Aquests errors, per tant, no són culpa del model, sinó de les dades amb les quals s'ha testejat.

Algunes de les frases en les que s'equivoca són:

- all nippon airways recibe el primer boeing dreamline tokio (efe). (castellà, prediu anglès)
- actueilement, il enseigne à la city universy of new york et collabore au webzie the daily beast. (francès, prediu anglès)
- anne hathaway mostra tutto (italià, prediu anglès)
- wat er nu gebeurt, is heel zwaar voor hem als jonge gast (alemany, prediu neerlandès)

L'alemany i l'anglès tenien menys de 10.000 frases (9990 i 9987 respectivament) i per això no correspon l'*accuracy* amb el nombre d'encerts (en els altres idiomes és una simple divisió entre 100 per obtenir el percentatge).

7.4 Noms propis

Observant les frases que fallava el test, s'ha pogut observar com en un gran nombre dels casos la predicció errònia es deu a l'aparició d'un nom propi en una altra llengua (com per exemple noms d'empreses). Per tal d'evitar aquest problema, s'ha realitzat un preprocessat en el qual s'intentava eliminar aquests noms propis. Aquesta és una tasca complicada de fer, ja que costa identificar els noms propis. Tot i ser poc fiable, s'ha considerat que, com a experiment, és interessant per analitzar com varia el comportament del model.

S'ha usat la llibreria `Spacy`, ja que conté les eines necessàries per realitzar aquesta tasca. Preprocessant les frases eliminant aquests noms els resultats són lleugerament diferents. Si eliminem també els caràcters estranys, la millor funció de suavitzat per aquestes dades és *lidstone*, a diferència de les altres en les quals era *linear discounting*. Assoleix, en el test i usant *Laplace* (*Lidstone* amb lambda 1) tan sols 63 errors, però no és del tot fiable. Si no eliminem els caràcters estranys, per exemple, fa més errors que sense eliminar els noms propis (71). És interessant la matriu de confusió en aquest cas:

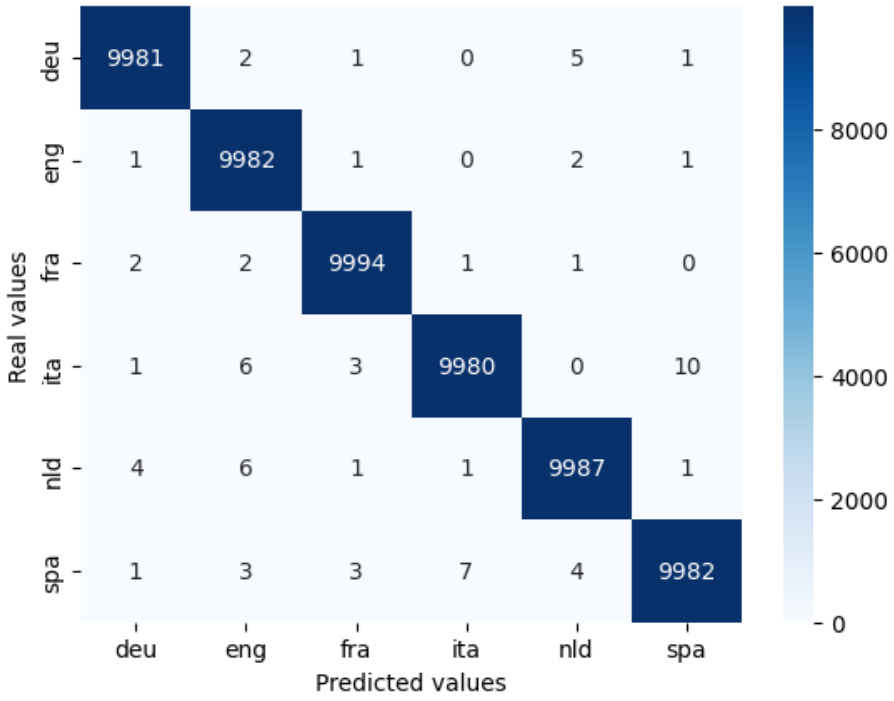

Figura 9: Matriu de confusió del model entrenat eliminant noms propis

Observem com els errors estan més distribuïts que abans, i n'hi ha menys a la columna d'anglès (el prediu menys vegades). Aquest fet ens indica que, el problema que intentàvem arreglar (que predeia anglès perquè els noms de les companyies són en aquesta llengua) en alguns casos s'ha solucionat. Tot i això, s'han creat problemes nous, ja que ara hi ha frases tal que ". ", originalment conformades només per noms propis.

Després de valorar, s'ha arribat a la conclusió que el temps d'execució addicional (a l'hora de fer el train) i la pèrdua d'informació (algunes frases deixaven de tenir sentit) provocaven que aquesta opció no fos bona en un model (almenys com està actualment). Tot i això, com a experiment ha servit per veure com es poden combinar tècniques diferents de Processament del Llenguatge Natural per un mateix objectiu.

8 Conclusions

Els resultats obtinguts en el test permeten extreure diverses conclusions. Una conclusió que s'ha comentat per sobre en l'apartat anterior és la dificultat de distingir llengües similars.

En les dades, s'observa com el neerlandès costa de distingir de l'alemany i de l'anglès. Aquest fet es deu al fet que les tres són llengües germàniques, concretament occidentals. El fet que provinguin de la mateixa branca (d'una mateixa família) de llenguatges provoca similituds entre els tres. Tenen arrels comunes, gramàtiques bastant similars i uns quants doblets (paraules amb el mateix origen etimològic). En alguns casos fins i tot comparteixen paraules, com aigua, que tant en neerlandès com en anglès s'anomena *water*. Aquest fet, evidentment, complica separar i distingir els idiomes entre ells. De manera similar passa amb l'espanyol, l'italià i el francès (llengües romàniques), tot i que en aquests casos el model ha estat millor a l'hora de determinar l'idioma.

Una altra conclusió, fruit de provar diverses frases, és que hi ha caràcters que permeten identificar quin idioma és, com per exemple els accents o lletres especials. N'hi ha d'aquests que apareixen tan sols en un idioma, fet que ajuda molt al model. En alguns casos, és la clau que permet distingir entre dos idiomes propers. Aquestes regles o patrons que sembla que trobem quan analitzem un model, les fem inconscientment constantment. Un exemple seria l'accent obert a la lletra a en català, que permet distingir pàgina (català) de página (castellà). D'aquí podem extreure que, molts dels patrons que inconscientment apliquem per tal de discriminar en quin idioma està un text, els pot aplicar o li poden servir també a una màquina.

Els resultats del model són molt bons, tenint en compte els pocs errors que fa. És un model bastant senzill comparat amb algunes altres aplicacions de processament del llenguatge humà, i tot i això els resultats són molt bons. Aquest fet ens indica que a vegades, no cal crear un model molt complex per resoldre un problema, sinó que és millor centrar-se en entendre'l bé i buscar formes senzilles de resoldre'l. Una alternativa seria utilitzar un model que usés paraules enlloc de caràcters, que probablement seria millor.

9 Referències

[1] Leipzig Corpora Collection Download Page. `https://wortschatz.uni-leipzig.de/en/download`.