Model de llenguatge detector d'idiomes

Roger Baiges Trilla i Adrià Cantarero Carreras

4 de Març de 2024

Universitat Politècnica de Catalunya

Grau en Intel·ligència Artificial

Processament del Llenguatge Humà

Contents

1 Introducció

Aquest projecte explora el desenvolupament d'un model de predicció de llenguatge basat en trigrames de caràcters, aprofitant les tècniques d'Identificació del Llenguatge (Llei de Lidstone, LID) per calcular valors de probabilitat. L'objectiu principal és classificar amb precisió mostres de text entre sis idiomes: anglès, espanyol, italià, neerlandès, alemany i francès. Mitjançant mètodes de compteig per trobar la freqüència d'aquests trigrames dins dels diferents corpus de text, el model aplica la Llei de Lidstone per tal de generar puntuacions de probabilitat que reflecteixen la probabilitat d'una mostra de text de pertànyer a cada categoria d'idioma, facilitant així una classificació precisa.

El conjunt de dades d'aquest projecte consisteix en 30.000 frases d'entrenament (*train*), un 30% del qual s'utilitza com set de validació per tal d'escollir els hiperparàmetres òptims per la funció de LID; i 10.000 frases de prova (*test*) per a cada idioma. Aquest extens conjunt d'entrenament permet al model aprendre perfils de trigrames detallats per a cada idioma, mentre que el conjunt de prova proporciona un marc robust per avaluar l'exactitud de la classificació del model i la seva capacitat de generalització.

2 Implementació

2.1 Introducció

En aquesta secció expliquem la implementació principal del programa, utilitzant la Llei de Lidstone (LID):

$$P^T(ej) \approx P^T_{LID}(ej) = \frac{c_{T(ej)} + \lambda}{N_T + \lambda B}$$

on:

- $C_T(x)$: #ocurrències de x en el corpus d'entrenament T
- N_T: #ocurrències de trigrames en el corpus d'entrenament T
- B: #trigrames potencialment observables

Per tal d'aplicar LID en diferents frases o texts, calcularem el LID total com la suma de LIDs de cada trigrama existent a la frase. D'aquesta manera, calcularem el LID de la frase per a cada idioma, per posteriorment comparar els resultats i escollir l'idioma amb el resultat més alt com la predicció.

Aquesta idea la vam començar implementant d'una forma més manual i menys òptima (apartat 2.2), però posteriorment la vam descartar per una implementació molt més òptima, que ha resultat ser la final (apartat 2.3). A continuació, exposem les dues perspectives comentades

2.2 Idees inicials

Abans que res, hi ha algunes funcions que utilitzem en les dues implementacions. Primerament, tenim la funció que aplica el preprocessament al text, tal com es requeria:

```
def preprocess_text(text):
    text = re.sub(r'\d+', '', text)  # Delete numbers
    text = text.lower()              # Lowercase
    text = re.sub(r'\s+', ' ', text) # Delete multiple spaces
    text = re.sub(r'\n', '  ', text) # Doble space for new line
    return text
```

A més, tenim una funció per crear trigrames a partir de text, així com una altra que filtra aquells trigrames poc freqüents, és a dir, que apareixen menys de cinc vegades:

```
def create_trigrams(text):
    characters = [char for char in text]
    return [''.join(trigram) for trigram in ngrams(characters, 3) if
        len(trigram) == 3]

def filter_infrequent_trigramsDICT(list_trigrams):
    # Count the frequency of each trigram in the list
    trigram_counts = Counter(list_trigrams)
```

```
8
9       # Select the trigrams that appear at least 5 times
10      frequent_trigrams = {trigram for trigram, count in
            trigram_counts.items() if count >= 5}
11
12      # Filter the list of trigrams to include only the frequent
            trigrams
13      filtered_list_trigrams = [trigram for trigram in list_trigrams
            if trigram in frequent_trigrams]
14
15      return filtered_list_trigrams
```

Per últim, per tal de llegir els diferents fitxers i aplicar les funcions anteriors, tenim la següent funció, que crea un diccionari amb els trigrames d'entrenament per a cada idioma. No obstant, aquesta funció ha anat evolucionant lleugerament junt amb els canvis d'implementació del projecte. (A més, en aquesta implementació no vam tractar encara les dades de *test*, així que no comentarem la seva lectura ni tractament)

```
1   def load_data_dicc(filenames):
2       data = {}
3       for file in filenames:
4           with open(file, 'r', encoding='utf-8') as f:
5               text = preprocess_text(f.read())
6               trigrams = create_trigrams(text)
7               filename = os.path.basename(file).split('_')[0]
8               data[filename] = trigrams
9
10      return data
```

Esmentades aquestes funcions comunes, podem començar a definir la implementació inicial. En un primer moment, vam pensar en implementar el programa d'una manera bastant humana, és a dir, seguint els passos de càlcul que faríem nosaltres. D'aquesta manera, el procediment era calcular el LID de cada trigrama existent a la frase a predir per cada idioma, i sumar-los per tindre el LID de la frase completa per cada idioma.

Per tant, primerament partíem d'una funció LID, tal com he vist a la introducció d'aquesta mateixa secció (apartat 2.1), que hem implementat de la següent forma:

```
1   def LID(count_trigram, NT, B, lam=0.5):
2       result_LID = np.log((count_trigram + lam) / (NT + lam * B))
3       return result_LID
```

On, en aquesta implementació i la següent, vam centrar els hiperparàmetres *lambda* i B a uns valors iguals per a tots els idiomes. No obstant, en el programa definitiu implementem una validació del model per escollir els valors òptims per aquests dos hiperparàmetres per a cada idioma (veure apartat 2.4).

Un cop definida la funció de la Llei de Lidstone, anàvem fent el procés humà d'anar fent aquest càlcul per a cada idioma, per després aplicar-la per a cada idioma, per finalment aplicar aquesta última per a la suma de trigrames de cada frase a predir. D'aquesta manera, obteníem tres funcions, que feien una feina molt poc òptima, repetint el que es podia fer d'una manera molt més resumida. Arribat a aquest punt, ens adonem de la pobre implementació que hem aplicat, de manera que pensem una forma més òptima de fer el mateix. Això ho expliquem al següent apartat.

2.3 Implementació definitiva

En aquest apartat, expliquem la nostra implementació definitiva del programa. No obstant, no entrem en el tractament i elecció dels hiperparàmetres de la funció LID, *lambda* i B (aquesta part del model es troba a l'apartat 2.4).

Dit això, ja hem vist com obteníem els texts de *train*, com els preprocessavem, transformàvem a trigrames, i els filtràvem. No obstant, en aquesta implementació ja tractem les dades de *test*, amb les quals podrem provar el model i analitzar els seus resultats. Degut a això, per obtenir les dades de *test*, utilitzem la següent funció:

```
def load_data_test(filenames):
    texts = []
    labels = []
    for file in filenames:
        label = os.path.basename(file).split('_')[0]  # The tag is
           the first part of the filename
        with open(file, 'r', encoding='utf-8') as f:
            for line in f:
                texts.append(line.strip())
                labels.append(label)
    return texts, labels
```

Més endavant tornarem a les dades de *test*, però per ara seguim amb l'explicació de la implementació de la funció LID.

Com havíem definit anteriorment, tenim una funció bàsica de LID ja implementada. No obstant, aquí no esperem a la frase a predir o les dades de *test* per calcular el valor del LID del text. En comptes d'això, creem un diccionari el qual té com a *keys* tots els trigrames no repetits de les dades de *train*, i cada trigrama té un altre diccionari amb el compteig d'aquell trigrama per a cada idioma. D'aquesta manera, a continuació apliquem sobre el mateix diccionari de diccionaris la funció LID, obtenint així un diccionari de tots els trigrames amb els que hem entrenat el model, els quals ja tenen calculades les seves probabilitats per idioma utilitzant la funció LID anteriorment implementada:

```
def create_probability_dict(data, lam=0.5):

    trigram_counts = {}

    for lang, lang_trigrams in data.items():
```

```
        for trigram in lang_trigrams:
            if trigram not in trigram_counts:
                trigram_counts[trigram] = {'deu': 0, 'eng': 0, 'fra'
                    : 0, 'ita': 0, 'nld': 0, 'spa': 0}
            trigram_counts[trigram][lang] += 1

    trigram_LID = {}

    for trigram, lang_counts in trigram_counts.items():
        trigram_lang_LID = {}
        for lang, count in lang_counts.items():
            trigram_lang_LID[lang] = LID(count, info_lang[lang]['NT'
                ], info_lang[lang]['B'], lam)
        trigram_LID[trigram] = trigram_lang_LID

    return trigram_LID
```

Amb aquesta funció ja tenim tots els valors calculats que necessitem. A més, sempre assegurem que, com a mínim, per a cada idioma el trigrama tingui un compteig de zero, per tal d'evitar errors en el càlcul del LID.

Finalment, ja només aplicarem una funció que rep un text a predir, i buscarà les probabilitats de cada trigrama del text al diccionari *trigram_LID*, només havent de sumar aquestes per a cada idioma i retornar el sumatori amb valor màxim (normalitzat per un millor enteniment [...]), el qual ens proporcionarà l'idioma predit pel text d'*input*.

2.4 Validació per cercar els hiperparàmetres

Una vegada teníem un codi funcional, encara que els resultats obtinguts ja eren excel·lents, ens feia falta poder cercar i demostrar que els hiperparàmetres escollits eren els millors per aquest projecte. Per fer-ho vam decidir crear un nou dataset a partir de les dades del train per tal de crear les de validació. En aquest cas es va considerar que era millor realitzar això que fer un *Cross-Validation* degut a que contàvem amb nombre prou elevat d'exemples.

Per crear-lo, mentre es processaven les dades d'entrenament, vam seleccionar aleatòriament el 30% de les frases de cada idioma per crear un nou conjunt de dades de validació. Aquest pas era crucial per trobar la millor combinació d'hiperparàmetres per a la nostra funció de Identificació de l'Idioma (**LID**), específicament els valors de B i λ.

Com es pot observar en l'script 4, una vegada s'havien llegit totes les frases d'entrenament, es desordenaven completament i es seleccionaven les 30% primeres. Aquest mètode ens assegura més context alhora que robustesa al comparar-ho amb l'altra opció possible que consistia en aleatòriament seleccionar el 30% dels trigrames.

```
def load_data_train_val(filenames):
    train_data = {}
    val_data = {}

```

```
    for file in filenames:
        with open(file, 'r', encoding='utf-8') as f:
            text = preprocess_text(f.read())
            sentences = text.split('  ')
            trigrams = create_trigrams(text)
            filename = os.path.basename(file).split('_')[0]
            train_data[filename] = trigrams.copy()

            # Randomly select 30% of the sentences for validation
            random.shuffle(sentences)
            split_index = int(len(sentences) * 0.3)
            val_sentences = sentences[:split_index]
            val_trigrams = create_trigrams(' '.join(val_sentences))
            val_data[filename] = val_trigrams

    return train_data, val_data
```

Pel que fa als valors a buscar, B representa el nombre de trigrames possibles a observar. És important destacar que mantenir el mateix valor de B per a cada idioma assegura una base de comparació equitativa i millora la precisió de la detecció d'idiomes. Per altra banda, λ és un factor que varia entre 0 i 1, utilitzat per assignar una probabilitat mínima als trigrames no observats, evitant així el problema de probabilitats nul·les en el càlcul de **LID**.

Els valors d'hiperparàmetres avaluats van incloure $\lambda = [0.1, 0.3, 0.5, 0.7, 1]$ i $B = [25^3, 26^3, 27^3, 30^3, 33^3]$. Els valors de B provats provenen d'aproximar una fita superior de trigrames possibles calculant totes les combinacions però sense intentar que sigui un límit massa més gran que el real. La ràpida execució d'aquestes proves va ser possible gràcies a una implementació eficient del codi comentada ja en el punt 2.3 permetent una cerca exhaustiva, concretament calia calcular les probabilitats i els resultats de les prediccions 25 vegades.

Després d'un profund procés de validació, els millors valors trobats van ser $\lambda = 1$ i $B = 30^3$. La selecció de $\lambda = 1$, coincident amb una suavització o correció de *Laplace*, indica que assignar una probabilitat igualitària a tots els trigrames, inclosos els no observats, va resultar ser l'estratègia més eficaç. Això pot semblar contra-intuïtiu, però en el context de la nostra aplicació, aquest enfocament va maximitzar l'eficàcia de la detecció d'idiomes. D'altra banda, el valor de $B = 33^3$ va ser el màxim possible dins del rang d'hiperparàmetres considerats, suggerint que un major nombre de trigrames potencia la capacitat del model per diferenciar entre idiomes.

Com es pot observar en la següent figura 1, l'hyperparàmetre B no afectava al desenvolupament del model de llenguatge sinó que tan sols la *lambda* controlava el rendiment. Com es pot veure a major *lambda* millors resultats cosa que podría indicar que en aquest context és millor donar probabilitats més elevades als trigrames no observats.

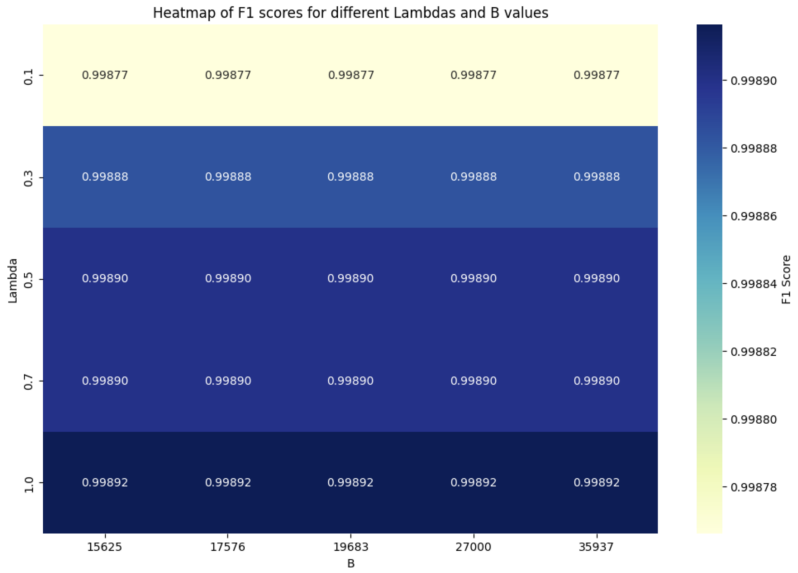

Figure 1: Matriu de confusió del model de detecció d'idiomes.

És important notar que, malgrat que aquests valors van proporcionar el millor **f1-score** en la nostra validació, altres combinacions completament diferents d'hiperparàmetres van resultar en el mateix rendiment. Això implica que aquests no necessàriament són els millors valors de manera global sinó els primers en obtenir aquest resultat.

Una vegada seleccionats aquests hiperparàmetres, vam procedir a calcular les probabilitats de cada trigrama per al conjunt de dades complet. Sorprenentment, els resultats obtinguts van ser idèntics als de la validació, amb un **f1-score** del 99.89%. Aquesta consistència suggereix que potser no eren necessàries tantes dades d'entrenament per assolir un alt nivell de precisió. A més, això indica que les poques frases incorrectament predites podrien tenir característiques comunes, un aspecte que analitzem més a fons a la secció 3.

3 Anàlisi dels resultats i conclusions

3.1 Resultats

Els resultats obtinguts pel model de detecció d'idiomes mostren una alta precisió, tal com es pot observar en la matriu de confusió presentada (figura 2). Amb un **f1-score**, **precision** i **accuracy** de 99.89%, el model demostra una capacitat destacada per classificar correctament les frases segons el seu idioma.

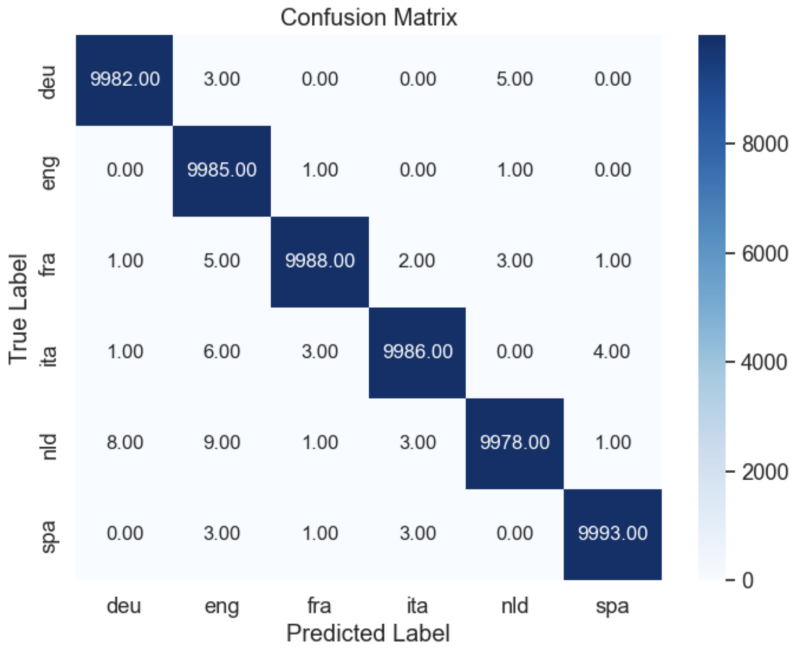

Figure 2: Matriu de confusió del model de detecció d'idiomes.

Com es pot veure a la Figura 2, les confusions més comunes entre idiomes són mínimes, indicant que el model pot distingir efectivament entre les llengües amb una alta taxa d'èxit. No obstant això, tenim algunes confusions:

- L'holandès (nld) és l'idioma més confós, principalment amb l'alemany (deu) i l'anglès (eng).
- Hi ha una confusió moderada entre els idiomes romànics, especialment entre l'italià (ita) amb el francès (fra) i l'espanyol (spa).

Com ja s'havia mencionat en el punt de validació 2.4, els resultats obtinguts al calcular les probabilitats, amb tan sols 10.000 frases respecte amb les 30.000, són exactament els mateixos. Aquest fet ens porta a la necessitat d'analitzar més detingudament les frases que el model no ha pogut detectar correctament. L'objectiu seria identificar patrons o motius comuns entre aquestes. Aquesta anàlisi ens pot ajudar a comprendre millor les limitacions del model i a explorar maneres de millorar la seva capacitat per gestionar les variacions lingüístiques o les anomalies en les dades.

3.2 Anàlisi de les Classificacions Errònies

En aquest apartat presentem un examen detallat de les frases incorrectament classificades pel model. Observem que les frases més curtes tendeixen a ser confoses, probablement a causa de la falta de context suficient. A continuació es discuteixen exemples específics:

- "Les Anderlechtois devront avant tout museler Ruud Van Nistelrooy.": Predicció en holandès, real en francès. La presència del nom "Ruud Van Nistelrooy" podria haver induït el model a associar la frase amb l'holandès.
- "Salle Wilfrid-Pelletier, vendredi 26 mars 2010.": Predicció en alemany, real en francès. La brevetat d'aquesta frase i la falta de context lingüístic clar podrien ser la causa de la classificació errònia.
- "Bert van Marwijk ne sourit pas toujours.": Predicció en holandès, real en francès. Similar a l'exemple anterior, el cognom "van Marwijk" pot tenir una aparença holandesa.
- "'Non non, il dit.": Predicció en italià, real en francès. La frase és extremadament curta i manca de context, el que pot conduir a confusió.
- "Encore une star addict de Twiiter!": Predicció en anglès, real en francès. La paraula "addict" és comuna en anglès, el que pot haver confós el model.
- "Herman Van Rompuy Bart De Wever : Ne traîne pas trop, zeg.": Predicció en holandès, real en francès. La presència de noms flamencs i la interjecció "zeg" podrien haver desviat la predicció.
- "Ingenico: retenu par Best Buy en Turquie.": Predicció en espanyol, real en francès. La inclusió de noms d'empreses internacionals pot haver portat a una classificació errònia.
- "Enzo Bricolo (page perso,) le 03/04/2010 à 20:47.": Predicció en italià, real en francès. El nom propi "Enzo Bricolo" podria semblar italià, influenciant la classificació.
- "Wolseley: Morgan Stanley relève son objectif de cours.": Predicció en anglès, real en francès. Novament, els noms d'empreses o entitats poden portar a confusions.

Aquests casos específics destaquen la importància del context i com la presència de noms propis, termes tècnics i paraules aïllades amb similituds en múltiples idiomes poden portar a errors de classificació. Els patrons identificats aquí poden servir per refinar el model i millorar la seva capacitat per manejar aquestes anomalies.

Per abordar les anomalies observades, podríem refinar el preprocessament del text incloent tècniques que contextualitzin millor les frases curtes i que gestionin els noms propis i termes tècnics. L'ús de llistes de paraules d'aturada específiques per idioma podria evitar que paraules no significatives influenciïn la classificació. Així mateix, l'enriquiment del context en el conjunt de dades podria proveir al model més pistes lingüístiques per a una identificació més precisa.

Cal mencionar

3.3 Conclusions

El projecte de model de llenguatge per a la detecció d'idiomes ha demostrat ser extremadament útil i eficaç, amb un f1-score de 99.89%. Aquest resultat indica que el model té una capacitat destacada per identificar correctament l'idioma de les frases analitzades, la qual cosa subratlla la seva aplicabilitat en situacions reals on es requereix una identificació ràpida i precisa de l'idioma.

Els resultats obtinguts mostren que les tècniques utilitzades per construir el model, incloent el preprocessament del text, la creació de trigrames de caràcters i el suavitzat aplicat durant el càlcul de les probabilitats, han estat ben escollides i implementades. La metodologia ha estat capaç de gestionar eficientment tant les frases llargues com les curtes, encara que les segones presenten més dificultats en la classificació degut a la menor quantitat de trigrames en elles.

Tot i l'elevat rendiment, s'han identificat alguns errors de classificació, majoritàriament en frases amb noms propis o termes tècnics que poden ser confusos. Per adreçar aquests errors, es podrien explorar estratègies addicionals de preprocessament, com l'ampliació de les llistes de paraules d'aturada o l'increment de l'enriquiment contextual dels conjunts de dades. A més, podrien implementar-se mètodes de post-processament que revisin les classificacions basant-se en un anàlisi de context més profund.

En resum, el projecte ha demostrat que és possible desenvolupar eines de detecció d'idiomes altament precises i ràpides; les quals podrien ser aplicades en una àmplia gamma de camps.

4 Extra: Model Classificador de Naive Bayes

Abans de començar a crear el nostre model de llenguatge predictor d'idiomes, ja explicat en seccions anteriors, vam crear un model classificador multinomial de Naive Bayes.

La idea bàsica era la mateixa que l'anterior:

- Preprocessar el text
- Crear els trigrames
- Entrenar el model (encara que anteriorment no s'entrenava i simplement es calculaven les probabilitats de cada trigrama)
- Predir les frases del test

Amb això no en teníem suficient degut a que per poder entrenar un model d'aquest tipus fa falta que totes les entrades siguin numèriques. Per tant, es va haver de traduir cada document del train separat en trigrames a un vector mitjançant un **CountVectorizer()** com es pot veure en l'script 4. Cal mencionar que per tal de suavitzar les classificacions el model es va entrenar amb una *alpha* de 0.5 com a paràmetre de regularització o de *Laplace*.

```
# Vectorize the training and test data

vectorizer = CountVectorizer()
X_train_vect = vectorizer.fit_transform(X_train_filtered)
X_test_vect = vectorizer.transform(X_test)
```

Finalment per poder analitzar els resultats va ser tan fàcil com entrenar el model amb els idiomes vectoritzats com es pot veure en l'script 4, i per a cada frase del test, passar-la a trigrames, vectoritzar-la i finalment comparar les prediccions del model amb els valors reals.

```
# Train the Multinomial Naive Bayes model

model = MultinomialNB(alpha=0.5)
model.fit(X_train_vect, y_train)
```

En aquest cas el model va obtenir unes mètriques lleugerament inferiors que els del model de llenguatge, concretament un **f1-score** de 99.78%. Tret d'això, el model de *Naive Bayes* ofereix una velocitat de predicció instantània, una característica destacable quan el temps és un factor crític.