UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Extracció d'entitats anomenades

Pràctica 3

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

El reconeixement d'entitats anomenades (NER, Named Entity Recognition) és una tasca crucial en el processament del llenguatge natural. Aquesta tasca consisteix a identificar i classificar els noms propis en un text en categories predefinides com ara noms de persones, organitzacions, ubicacions, expressions de temps, quantitats, valors monetaris, percentatges, etc. El NER és important per a moltes aplicacions de PLN, incloent la traducció automàtica, el resum de textos, la recuperació d'informació i la resposta a preguntes, entre d'altres.

En aquest projecte, es presentarà l'implementació d'un reconeixedor d'entitats anomenades utilitzant els Conditional Random Fields (CRF) per a l'espanyol i el neerlandès. Els CRF són un tipus de model d'aprenentatge supervisat que s'ha utilitzat amb èxit per a tasques de seqüenciació com el NER. Es farà ús de la classe *nltk.tag.CRFTagger* de la biblioteca Natural Language Toolkit (NLTK), que proporciona un implementació de CRF per a tasques de seqüenciació de text.

A més, s'exploraran diverses característiques i codificacions per a millorar el rendiment del model. La funció d'extracció de característiques per defecte en *nltk.tag.CRFTagger* inclou característiques com la paraula actual, si la paraula està en majúscules, si conté signes de puntuació, si conté números, i els sufixos de la paraula. No obstant això, aquesta funció pot ser personalitzada per a incloure característiques addicionals que podrien ajudar a millorar el rendiment del model, com la morfologia, la longitud, els prefixos, els lemmas, els *POS-tags*, els gazetteers, i les llistes de paraules.

Es consideraran també diferents esquemes de codificació per a les etiquetes del NER, incloent *BIO* (Begin-Inside-Outside), *BIOW* (Begin-Inside-Outside-With) i *IO* (Inside-Outside), entre d'altres. Cada un d'aquests esquemes té les seves pròpies avantatges i desavantatges, i l'elecció d'un esquema sobre un altre pot tenir un gran impacte en el rendiment del model.

Finalment, el model resultant serà provat en textos reals per a avaluar el seu rendiment en situacions pràctiques. Es proporcionarà una discussió detallada dels resultats obtinguts i es formularan conclusions basades en aquests resultats.

2. RECURSOS

2.1 ARXIUS

Per a la realització d'aquest projecte, es proporcionen diversos recursos que són essencials per a la comprensió completa del treball realitzat i per a la seva reproducció.

- En primer lloc, disposem de l'informe actual que està llegint. Aquest document proporciona una descripció detallada del projecte, incloent la introducció, la descripció de les dades utilitzades, les funcions que s'han implementat per al model, l'experimentació realitzada, i les conclusions obtingudes.
- En segon lloc, es proporciona un fitxer Jupyter Notebook amb el nom "practica_3_Llum_Fuster_Artur_Aubach.ipynb". Aquest fitxer conté tot el codi que s'ha utilitzat en aquesta pràctica, incloent la implementació de les funcions i els experiments realitzats.
- En tercer lloc, es proporciona una carpeta anomenada "models". Aquesta carpeta conté models preentrenats que s'han utilitzat en aquest projecte. La disponibilitat d'aquests models permet estalviar temps, ja que el lector no necessita entrenar els models des de zero. No obstant això, el codi està dissenyat per a detectar si aquests models estan disponibles i, en cas que no ho estiguin, els crearà automàticament.
- En quart lloc, es proporciona una carpeta anomenada "gazetteers". Aquesta carpeta conté llistes de paraules (gazetteers) que es faran servir com a característiques en el model de reconeixement d'entitats anomenades.
- Finalment, es proporcionen dos fitxers CSV: "model_features_esp.csv" i "model_features_ned.csv". Aquests fitxers contenen la avaluació de diferents models. Al igual que amb els models preentrenats, aquests fitxers no són obligatoris, ja que el codi pot generar aquests fitxers si no estan disponibles.

Tots aquests recursos són crucials per a la comprensió completa del treball realitzat i faciliten la seva reproducció.

2.2 DADES UTILITZADES

En aquest projecte, hem utilitzat les dades de la *Conferència sobre el Tractament Computacional del Llenguatge Natural (CoNLL-2002)* per a entrenar i validar els nostres models. Aquesta base de dades es pot descarregar directament des de la biblioteca *Natural Language Toolkit (NLTK)* i inclou dades etiquetades en diferents idiomes.

Per a l'entrenament i la validació dels nostres models, hem utilitzat dades en espanyol i neerlandès. Cada conjunt de dades inclou tres subconjunts: un per a l'entrenament (train), un per a la validació (validation) i un altre per a la prova (test).

```
#BAIXAR I GUARDAR DADES NECESSÀRIES
nltk.download('conll2002')

#ESPANYOL
train_IOB_ESP = conll2002.iob_sents('esp.train') #train
val_IOB_ESP = conll2002.iob_sents('esp.testa') #validation
test_IOB_ESP = conll2002.iob_sents('esp.testb') #test

#NEERLANDÈS
train_IOB_NED = conll2002.iob_sents('ned.train') #train
val_IOB_NED  = conll2002.iob_sents('ned.testa') #validation
test_IOB_NED = conll2002.iob_sents('ned.testb') #test
```

Cal destacar que totes aquestes dades estan etiquetades segons l'esquema *BIO*. Aquesta codificació assigna a cada paraula en el text una etiqueta que indica si la paraula és el començament d'una entitat (B), si és part d'una entitat però no és el començament (I), o si no és part d'una entitat (O). Aquesta informació serà clau per a l'entrenament dels nostres models de reconeixement d'entitats anomenades.

3. FUNCIONS PER EL MODEL

En aquest apartat, establirem una sèrie de funcions clau que seran utilitzades durant la fase d'experimentació. Aquestes funcions es dissenyen per a facilitar la implementació i la prova de diferents configuracions del model. Específicament, estarem creant funcions per a la validació del model, la codificació de les dades, l'extracció de característiques i la creació del model.

1. Les funcions de *validació* (3.1) estaran destinades a avaluar el rendiment del model basant-se en el conjunt de dades de validació. Aquesta funció serà útil per a comparar diferents models i configuracions, i per a triar la millor opció.
2. Les funcions de *codificació* (3.2) permetran convertir les etiquetes de les dades en diferents esquemes de codificació. Com es va esmentar anteriorment, l'esquema de codificació pot tenir un gran impacte en el rendiment del model, així que aquesta funció serà útil per a provar diferents esquemes.
3. Les funcions d'*extracció de característiques* (3.3) seran utilitzada per a definir quines característiques del text seran considerades pel model. Aquesta funció es podrà personalitzar per a incloure una gran varietat de característiques, com la morfologia, la longitud, els prefixos, els lemmas, els POS-tags, els gazetteers, i les llistes de paraules.
4. Finalment, Les funcions de *creació del model* (3.4) ens permetran establir el tipus de model que volem utilitzar (en aquest cas, un Conditional Random Field) i entrenar-lo amb un conjunt de dades específic.

Totes aquestes funcions seran aplicades en la fase d'experimentació (secció 3) per a provar diferents configuracions del model i optimitzar el seu rendiment.

3.1. VALIDACIÓ

La validació és una part integral del procés de modelatge, que ens permet avaluar el rendiment d'un model basant-nos en un conjunt de dades de validació. En aquest apartat, s'han desenvolupat tres funcions clau per a la validació dels models: "`filtrar_tags`", "`tutu`" i "`accuracies`".

La primera funció, "filtrar_tags", és responsable de seleccionar les etiquetes rellevants d'un conjunt de dades. Quan s'introdueix un conjunt de dades, aquesta funció filtra les etiquetes, seleccionant només aquelles que són diferents de "O". Això ens ajuda a concentrar-nos en les entitats que són rellevants per a la tasca de reconeixement d'entitats anomenades.

```
# AGAFA LES DADES IMPORTANTS/ÚTILS (les "o" les elimina)
def filtrar_tags(lista_de_listas):
    resultado = []

    for frase in lista_de_listas:
        nueva_frase = []
        i = 0
        while i < len(frase):
            tag = frase[i][0:2]
            camp = frase[i][2:]
            if tag == "B-" or tag == 'U-' or tag == 'S-':
                inici = i
                while i + 1 < len(frase) and (frase[i + 1][0:2] == "I-" or
frase[i + 1][0:2] == "E-"):
                    i += 1
                nueva_frase.append((inici,i,camp))
            i += 1
        resultado.append(nueva_frase)

    return resultado
```

La segona funció, "tutu", és encarregada d'avaluar el rendiment de la predicció del model. Aquesta funció agafa un conjunt de dades i calcula els valors correctes i incorrectes de la predicció. Concretament, calcula els següents valors:

- **Correct matches:** són aquells casos en què un text en el fitxer de validació coincideix exactament amb un text corresponent en el fitxer de referència en termes de valors d'inici i final, així com del tipus d'entitat.
- **Incorrect matches:** es reporten quan els valors d'inici i final coincideixen, però no el tipus d'entitat.
- **Partial matches:** es reporten quan dos intervals [inici, final] tenen una intersecció no buida, com ara el cas de "vías respiratorias" i "respiratorias" en un exemple anterior (i amb la corresponent etiqueta).
- **Missing matches:** són aquells que apareixen en el fitxer de referència però no en el fitxer de validació.

- **Spurious matches:** són aquells que apareixen en el fitxer de validació però no en el fitxer de referència.

```
# CÀLCUL DE VALORS "Correct matches", "Partial matches", "Incorrect matches",
"Missing matches" i "Spurious matches"
def tutu(real,pred):
    c, p, i, m, s = 0, 0, 0, 0, 0
    parcials = (-1,-1,'',False)
    for sent, sentence in enumerate(pred): # [1 (0, 1, 'LOC'), 2 (3,7,'pers')]
        for ini_p,fi_p,tag_p in sentence: # [0, 1, 'LOC'] pred
            for ini_r,fi_r,tag_r in real[sent]: #[0, 1, 'LOC'] real
                #es parcial de l'anterior?
                if ini_p<=parcials[1] and tag_p==parcials[2]:
                    if not parcials[3]:
                        p += 0.5
                    parcials = (ini_r,fi_r,tag_r,True)
                    real[sent].remove((ini_r,fi_r,tag_r))
                    break
                #es correcta?
                elif ini_r==ini_p and fi_r==fi_p:
                    if tag_r == tag_p:
                        c+= 1
                    else:
                        i+=1
                    real[sent].remove((ini_r,fi_r,tag_r))
                    break
                #es parcial: considerem només eparcials que siguin més petites
                elif ini_r<=ini_p and fi_r>=fi_p:
                    if tag_r == tag_p:
                        p +=1
                        parcials = (ini_r,fi_r,tag_r,False)
                    else:
                        i+=1
                    real[sent].remove((ini_r,fi_r,tag_r))
                    break
                # trobat que no ho es
                elif ini_r > fi_p:
                    s+=1
                    break
            if real[sent] == []:
                if ini_p<=parcials[1] and tag_p==parcials[2]:
                    if not parcials[3]:
                        p += 0.5
                    parcials = (ini_r,fi_r,tag_r,True)
        m += len(real[sent])

    return c, p, i, m, s
```

La tercera funció, "`accuracies`", es dedica a calcular tres mesures clau d'avaluació a partir dels valors obtinguts amb la funció "`tutu`": *recall, precision* i *F1 score*.

- **Recall (recuperació):** és la proporció d'entitats positives que s'han identificat correctament. Es calcula com el nombre de coincidències correctes més la meitat del nombre de coincidències parcials, dividit pel total de coincidències correctes, incorrectes, parcials i mancades.
- **Precision (precisió):** és la proporció d'identificacions positives que eren en realitat correctes. Es calcula com el nombre de coincidències correctes més la meitat del nombre de coincidències parcials, dividit pel total de coincidències correctes, incorrectes, parcials i espúries.
- **F1 score:** és la mitjana harmònica de recall i precision. Aquesta mesura proporciona una única mètrica que combina ambdues mesures, i és especialment útil quan les classes estan desequilibrades. L'F1 score es calcula com la mitjana harmònica de la precisió i la recuperació, proporcionant un equilibri entre aquestes dues mesures en termes de la seva importància. Aquesta és una mesura útil quan volem comparar dos models o més i necessitem una sola mesura per resumir el rendiment.

```
#CALCULA DIFERENTS ACCURACY (Rec, Prec i F1)
def accuracies(tupl):
    c, p, i, m, s = tupl
    recall = (c+1/2*p)/(c+i+p+m)
    precision = (c+1/2*p)/(c+i+p+s)
    f1 = 2 * (precision*recall)/(precision+recall)
    return recall, precision, f1
```

Aquestes tres funcions són essencials per avaluar la qualitat del nostre model i ajustar-lo si és necessari. Cada funció proporciona una vista diferent de la eficàcia del model, des del nivell de detall del "`filtrat`" de les entitats, passant per l'anàlisi dels encerts i errors amb la funció "`tutu`", fins a l'avaluació global del rendiment del model amb la funció "`accuracies`".

En resum, a través d'aquestes funcions de validació, som capaços de realitzar una avaluació àmplia i detallada del rendiment dels nostres models, permetent-nos identificar àrees de millora i ajustar el nostre enfocament de modelatge en conseqüència.

3.2. CODIFICACIÓ

Les funcions de codificació es fan servir per convertir les etiquetes de les dades en diferents esquemes de codificació. Això és crucial per a la nostra tasca perquè diferents esquemes de codificació poden tenir impactes significatius en el rendiment del model, depenent de la naturalesa específica de les dades i la tasca a realitzar

En aquesta pràctica, hem optat per experimentar amb cinc esquemes de codificació diferents: *BIO, IO, BIOU, BIOEU* i *BIOEU+*. Ja hem explicat la codificació *BIO* en apartats anteriors ja que es la codificació de la base de dades predeterminada, però ens agradaria destacar per què hem decidit provar diferents esquemes de codificació. En funció de com estiguin distribuïdes les nostres dades i de la naturalesa específica de les entitats que volem identificar, alguns esquemes de codificació poden ser més eficaços que altres. Experimentar amb diferents esquemes ens permet optimitzar el rendiment del nostre model.

Les codificacions són les següents:

- **IO:** Aquesta codificació utilitza únicament dues etiquetes: "`I`" per a les paraules que formen part d'una entitat i "`O`" per a les paraules que no formen part d'una entitat. Per tant, no distingeix entre el començament i el continuament d'una entitat.
- **BIOU:** Similar a *BIO*, però afegeix una etiqueta "`U`" per a les entitats que es componen d'una única paraula. També afegeix una etiqueta "`E`" per a la última paraula d'una entitat.
- **BIOEU:** Aquest esquema és similar a *BIOU*, però afegeix una etiqueta "`E`" per a la última paraula d'una entitat.
- **BIOEU+:** Aquest esquema amplia *BIOEU* afegint una etiqueta "`S`" per a les paraules que estan soles i no formen part d'una entitat.

Per implementar cada un d'aquests esquemes de codificació, hem creat quatre funcions: "`bio_to_io`", "`bio_to_biou`", "`bio_to_bioeu`" i "`bio_to_bioeu_plus`". Cadascuna d'aquestes funcions pren un conjunt de dades amb codificació *BIO* i el transforma en l'esquema de codificació pertinent. Aquesta flexibilitat ens permet experimentar amb diferents esquemes de codificació de manera eficient i fàcil.

```python
#PASA LES DADES DEL ENCODING BIO A IO
def bio_to_io(sent):
    io_sent = []
    for word, pos, bio in sent:
        if bio.startswith('B') or bio.startswith('I'):
            tag = 'I'
        else:
            tag = 'O'
        io_sent.append((word, pos, f'{tag}-{bio[2:]}'))
    return io_sent
#PASA LES DADES DEL ENCODING BIO A BIOU
def bio_to_biou(sent):
    biow_sent = []
    for i, (word, pos, iob) in enumerate(sent):
        if iob.startswith('B'):
            tag = 'B'
        elif iob.startswith('I'):
            if i + 1 < len(sent) and sent[i + 1][2].startswith('I'):
                tag = 'I'
            else:
                tag = 'E'
        else:
            tag = 'O'
        biow_sent.append((word, pos, f'{tag}-{iob[2:]}'))
    return biow_sent
#PASA LES DADES DEL ENCODING BIO A BIOEU
def bio_to_bioeu(sent):
    bioeu_sent = []
    for i, (word, pos, bio) in enumerate(sent):
        if bio.startswith('B'):
            if i + 1 < len(sent) and sent[i + 1][2].startswith('I'):
                tag = 'B'
            else:
                tag = 'U'
        elif bio.startswith('I'):
            if i + 1 < len(sent) and sent[i + 1][2].startswith('I'):
                tag = 'I'
            else:
                tag = 'E'
        else:
            tag = 'O'
        bioeu_sent.append((word, pos, f'{tag}-{bio[2:]}'))
    return bioeu_sent
#PASA LES DADES DEL ENCODING BIO A BIOEU+
def bio_to_bioeu_plus(sent):
    bioeu_plus_sent = []
    for i, (word, pos, bio) in enumerate(sent):
        if bio.startswith('B'):
            if i + 1 < len(sent) and sent[i + 1][2].startswith('I'):
                tag = 'B'
            else:
```

```
                tag = 'U'
        elif bio.startswith('I'):
            if i + 1 < len(sent) and sent[i + 1][2].startswith('I'):
                tag = 'I'
            else:
                tag = 'E'
        elif bio.startswith('O'):
            if (i == 0 or sent[i - 1][2] == 'O') and (i + 1 == len(sent) or
sent[i + 1][2] == 'O'):
                tag = 'S'
            else:
                tag = 'O'
        bioeu_plus_sent.append((word, pos, f'{tag}-{bio[2:]}'))
    return bioeu_plus_sent
```

3.3. FEATURES

La detecció de característiques és una part fonamental en el procés de modelatge en l'aprenentatge automàtic. En aquesta pràctica, volem tenir en compte una sèrie de característiques del text per alimentar el nostre model. Aquestes característiques incloure aspectes com la capitalització, la presència de números, puntuació, sufix, longitud, prefix, lema, etiqueta POS (part of speech), paraula anterior i posterior, així com la presència del mot en un conjunt específic de paraules (conegut com a 'gazetteer') tant de caràcter general com de localització.

Per a això, hem creat una nova classe anomenada "`FeatureDetector`". Aquesta classe extén la funcionalitat de la classe *nltk.tag.CRFTagger*, permetent-nos definir quines característiques volem que el model consideri. Cada una d'aquestes característiques es pot activar o desactivar a l'hora de crear una instància de la classe.

```
# CREA FEATURES NOUS
class FeatureDetector:
    def __init__(self, cap=0, num=0, punt=0, suf=0, lon=0, pref=0, lemma=0,
postag=0, ant=0, post=0, gazatteers=None, list_words=None):
        self._pattern = re.compile(r'\d')
        self.cap = cap
        self.num = num
        self.punt = punt
        self.suf = suf
        self.lon = lon
        self.pref = pref
        self.lemma = lemma
```

```
        self.postag = postag
        self.ant = ant
        self.post = post
        self.gazatteers = gazatteers if gazatteers else []
        self.list_words = list_words if list_words else []

    def __call__(self, tokens, index):
        token = tokens[index][0]
        feature_list = []


        # Capitalization
        if self.cap==1:
            if token[0].isupper():
                feature_list.append("CAPITALIZATION")

        # Number
        if self.num==1:
            if re.search(self._pattern, token) is not None:
                feature_list.append("HAS_NUM")

        # Punctuation
        if self.punt==1:
            punc_cat = {"Pc", "Pd", "Ps", "Pe", "Pi", "Pf", "Po"}
            if all(unicodedata.category(x) in punc_cat for x in token):
                feature_list.append("PUNCTUATION")

        # Suffix up to length 3
        if self.suf==1:
            if len(token) > 1:
                feature_list.append("SUF_" + token[-1:])
            if len(token) > 2:
                feature_list.append("SUF_" + token[-2:])
            if len(token) > 3:
                feature_list.append("SUF_" + token[-3:])

        # Length of word
        if self.lon==1:
            feature_list.append(f"LENGTH_{len(token)}")

        # Prefix up to length 3
        if self.pref:
            if len(token) > 1:
                feature_list.append("PREF_" + token[:1])
            if len(token) > 2:
                feature_list.append("PREF_" + token[:2])
            if len(token) > 3:
                feature_list.append("PREF_" + token[:3])

        # Lemma (assuming English for simplicity)
        if self.lemma==1:
```

```
            from nltk.stem import WordNetLemmatizer
            lemmatizer = WordNetLemmatizer()
            feature_list.append(f"LEMMA_{lemmatizer.lemmatize(token)}")

        # POS Tag
        if self.postag==1:
            pos_tag = tokens[index][1]
            feature_list.append(f"POSTAG_{pos_tag}")

        # Paraula anterior
        if self.ant == 1:
            if index > 0:
                anterior = tokens[index-1][0]
            else:
                anterior = '.'
            feature_list.append(f"ANT_{anterior}")

        #Paraula posterior
        if self.post == 1:
            if index < len(tokens):
                posterior = tokens[index][0]
            else:
                posterior = '.'
            feature_list.append(f"POST_{posterior}")

        # Gazatteers
        if self.gazatteers:
            if token in self.gazatteers:
                feature_list.append("IN_GAZATTEERS")

        # List of words
        if self.list_words:
            if token in self.list_words:
                feature_list.append("IN_LIST_WORDS")

        feature_list.append("WORD_" + token)
        return dict(zip(feature_list, [True]*len(feature_list)))

    def features(self):
        if self.gazatteers:
            a = 1
        else:
            a = 0
        if self.list_words:
            b = 1
        else:
            b = 0
        return((self.cap, self.num, self.punt, self.suf, self.lon, self.pref,
self.lemma, self.postag,self.ant,self.post,a,b))
```

Pel que fa als *'gazetteers'*, són llistes de paraules o frases que es consideren rellevants per a una tasca específica. En aquest cas, hem utilitzat *'gazetteers'* per a localitzacions i de caracter general. Hem descarregat les dades de diferents fonts per a crear aquests 'gazetteers':

Per a les localitzacions, hem utilitzat les dades de Geonames, obtenint informació específica per a Espanya i els Països Baixos.

Pel que fa de caracter generals hem utilitzat dades de Naamkunde per als noms neerlandesos i dades de l'Institut Nacional d'Estadística d'Espanya per als noms espanyols.

També hem utilitzat altres fonts generals en anglès per a localitzacions i altres noms extretes de github.

Per a transformar aquestes dades descarregades en conjunts que puguem utilitzar amb la nostra classe "`FeatureDetector`", hem creat la funció "`obrit_doc`". Aquesta funció llegeix els fitxers descarregats i els converteix en conjunts de paraules que es poden utilitzar per a la detecció de característiques.

```
def obrit_doc(directorio, idioma):
    sett = set()

    def cargar_set(directorio):
        nonlocal sett  # Declaramos sett como nonlocal
        # Recorre todos los archivos en el directorio
        for nombre_archivo in os.listdir(directorio):
            # Solo considera archivos .txt
            if nombre_archivo.endswith('.txt'):
                # Abre el archivo y lee las palabras
                with io.open(os.path.join(directorio, nombre_archivo), 'r',
encoding='utf8') as f:
                    palabras = f.read().split()
                    sett.update(palabras)

            # Solo considera archivos .xlsx
            elif nombre_archivo.endswith('.xlsx'):
                # Lee el archivo de Excel
                df = pd.read_excel(os.path.join(directorio, nombre_archivo))
                # Asume que las palabras están en la primera columna
                palabras = []
                for celda in df.iloc[:, 0].tolist():
                    palabras.extend(str(celda).split())  # Convierte el
contenido de la celda a string y lo divide en palabras
                sett.update(palabras)

    directorio1 = os.path.join("gazetteers", directorio)
```

```
    cargar_set(directorio1)  # Ahora se llama después de definir la función

    directorio2 = os.path.join(directorio1, idioma)
    cargar_set(directorio2)  # Ahora se llama después de definir la función

    return sett
```

Per a gestionar totes aquestes dades, hem creat una carpeta específica anomenada "GETTEERS", on es troben tots aquests conjunts de paraules i les dades originals.

En resum, la nostra classe "`FeatureDetector`" permet personalitzar les característiques que es consideraran en el model, així com afegir informació específica de conjunts de paraules rellevants per a la tasca. La combinació de totes aquestes característiques proporciona al model una rica representació del text que pot ajudar a millorar el rendiment en la tasca de reconeixement d'entitats anomenades.

3.4. MODEL

L'apartat "3.4. MODEL" està dedicat a l'explicació de la funció i la classe utilitzades per a la creació i el maneig del model de *Classificació Aleatòria Condicional* (CRF, per les seves sigles en anglès).

En primer lloc, la funció "`tokenize_and_tag`" s'utilitza per a preparar el text que es provarà amb el model. Aquesta funció pren un text com a entrada, el tokenitza (és a dir, el divideix en paraules individuals o "tokens") i després etiqueta cada token amb la seva part de discurs (POS tag, per les seves sigles en anglès). El resultat és una llista de parells token-etiqueta que es podrà utilitzar posteriorment per provar el model.

```
#FUNCIÓ PER TOKENITZAR ELS TEXTOS EXTERNS
def tokenize_and_tag(text):
    # Tokenizar el texto
    tokens = nltk.word_tokenize(text)

    # Etiquetar las palabras
    tagged_tokens = nltk.pos_tag(tokens)

    return tagged_tokens
```

En segon lloc, la classe "`model`" es dedica a l'emmagatzematge de la informació necessària per a la creació i el funcionament del model CRF. Aquesta informació inclou la llengua en què es basarà el model, l'encoding que es farà servir, la funció de característiques (o "feature function"), i altres dades com les dades d'entrenament, validació i prova.

El model es pot entrenar amb un conjunt de dades específic a través del mètode "train_model". Aquest mètode primer prepara les dades d'entrenament (amb el mètode "data_train") i després crea i entrena un model CRF amb aquestes dades, emmagatzemant el model resultant en un fitxer. Si el fitxer ja existeix, s'assumeix que el model ja ha estat entrenat i no es realitza l'entrenament.

Una vegada entrenat el model, es pot validar amb un conjunt de dades de validació (mitjançant el mètode "`validate_model`"), i es pot testejar amb un conjunt de dades de prova (mitjançant el mètode "`test_model`"). Aquests mètodes també preparen les dades corresponents, carreguen el model des del fitxer on s'ha emmagatzemat, i calculen les prediccions del model per a les dades. També calculen mètriques com el recall, la precisió i el F1-score per a avaluar la qualitat de les prediccions.

Finalment, el model també es pot fer servir per a provar textos arbitraris (mitjançant el mètode "`try_text`"). Aquest mètode tokenitza i etiqueta el text (mitjançant la funció "`tokenize_and_tag`"), carrega el model i calcula les prediccions per als tokens del text.

```
# EXPERIMENTACIÓ AMB FEATURES
class model:
    def __init__(self,language, encoding, feature_function = FeatureDetector()):
        self.language = language
        self.encoding = encoding
        self.feature_function = feature_function
        self.model_output_path = ''
        self.train_data = None
        self.val_data = None
        self.val_out = None
        self.test_data = None
        self.test_out = None
        self.path()

    def path(self):
        tupl = self.feature_function.features()
        seq = ''.join(map(str, tupl))
        self.model_output_path =
f'models/{self.encoding}_{self.language}_{seq}.mdl'
```

```
    def data_train(self):
        train = eval(f'train_IOB_{self.language}')
        if not self.encoding == 'IOB':
            fn = eval(f'bio_to_{self.encoding.lower()}')
            tra = [fn(sent) for sent in train]
            training = [[((token,postag), tag) for token, postag, tag in sent]
for sent in tra]
        else:
            training = [[((token,postag), tag) for token, postag, tag in sent]
for sent in train]
        self.train_data = training

    def data_val(self):
        val = eval(f'val_IOB_{self.language}')
        data, out = [[(token,postag) for token, postag, _ in sent] for sent in
val], [[tag for _, _, tag in sent] for sent in val]
        self.val_out = out
        if not self.encoding == 'IOB':
            fn = eval(f'bio_to_{self.encoding.lower()}')
            vali = [fn(sent) for sent in val]
            validation = [[(token,postag) for token, postag, _ in sent] for sent
in vali]
        else:
            validation = data
        self.val_data = validation
        print(self.val_data[0])
        print(self.val_out[0])

    def data_test(self):
        test = eval(f'test_IOB_{self.language}')
        data, out = [[(token,postag) for token, postag, _ in sent] for sent in
test], [[tag for _, _, tag in sent] for sent in test]
        self.test_out = out
        if not self.encoding == 'IOB':
            fn = eval(f'bio_to_{self.encoding.lower()}')
            testt = [fn(sent) for sent in test]
            testing = [[(token,postag) for token, postag, _ in sent] for sent in
testt]
        else:
            testing = data
        self.test_data = testing

    def train_model(self):
        self.data_train()
        if not os.path.exists(self.model_output_path):
            crf_model = nltk.tag.CRFTagger(feature_func=self.feature_function)
            crf_model.train(self.train_data, self.model_output_path)
        else:
            print(f'  El model {self.model_output_path[7:]} ja estava entrenat')

    def validate_model(self):
```

```
        self.data_val()
        start_time = timeit.default_timer()
        crf_model = nltk.tag.CRFTagger(feature_func = self.feature_function) #no
oblidar de passar la feature function
        crf_model.set_model_file(self.model_output_path)
        predict = crf_model.tag_sents(self.val_data)
        pred = [[tag for _, tag in sent] for sent in predict]
        print('predicció',pred[0])
        tupla = tutu(filtrar_tags(self.val_out),filtrar_tags(pred))
        recall, precission, f1_score = accuracies(tupla)
        elapsed_time = timeit.default_timer() - start_time
        return recall, precission, f1_score, elapsed_time

    def test_model(self):
        self.data_test()
        crf_model = nltk.tag.CRFTagger(feature_func = self.feature_function) #no
oblidar de passar la feature function
        crf_model.set_model_file(self.model_output_path)
        predict = crf_model.tag_sents(self.val_data)
        pred = [[tag for _, tag in sent] for sent in predict]
        tupla = tutu(filtrar_tags(self.val_out),filtrar_tags(pred))
        recall, precission, f1_score = accuracies(tupla)
        return recall, precission, f1_score

    def try_text(self,text):
        dades = [tokenize_and_tag(t) for t in sent_tokenize(text)]
        crf_model = nltk.tag.CRFTagger(feature_func = self.feature_function) #no
oblidar de passar la feature function
        crf_model.set_model_file(self.model_output_path)
        predict = crf_model.tag_sents(dades)
        pred = [[tag for _, tag in sent] for sent in predict]
        return dades, filtrar_tags(pred)
```

4. EXPERIMENTACIÓ

En l'apartat 4. "EXPERIMENTACIÓ", es realitzen una sèrie de proves i anàlisis amb l'objectiu d'optimitzar el model de reconeixement d'entitats anomenades (NER) basat en CRF (Conditional Random Fields) per a la detecció d'entitats en textos en espanyol i neerlandès.

La primera etapa d'aquesta experimentació implica la selecció de les característiques més rellevants per al model. Això es realitza mitjançant un procés anomenat "***backward elimination***". En aquest mètode, s'eliminen iterativament les característiques menys significatives del model fins que només queden les que maximitzen el rendiment del model.

A continuació, es realitza un anàlisi comparatiu de diferents codificacions utilitzades en el procés de tagatge d'entitats. Això es fa utilitzant diferents esquemes de codificació com *IOB, IO, BIOU, BIOEU, i BIOEU+*. El propòsit d'aquest anàlisi és determinar quina codificació proporciona els millors resultats en termes de precisió, record i puntuació F1.

Un cop es tenen les millors característiques i codificació, es realitza una anàlisi més detallada dels models entrenats i validats. Aquesta anàlisi inclou la generació de gràfics per a la comparació de les característiques i de les codificacions. També es menciona que el temps d'execució no es considera en aquesta anàlisi, ja que l'entrenament només s'ha de realitzar una vegada i el temps d'execució de la validació és relativament baix.

Finalment, es realitzen proves amb el millor model per a cada idioma utilitzant dades de test. Aquestes proves proporcionen una visió del rendiment final dels models.

En resum, aquest apartat d'Experimentació és crucial per a la millora i optimització del model NER, i aporta una comprensió més profunda de l'eficàcia de diferents característiques i codificacions en el procés de tagatge d'entitats.

4.1. FEATURES

Per a provar diferents feature functions està clar que no podem executar totes les combinacions possibles de variables ja que són un nombre molt elevat de models diferents. Per tant hem decidit aplicar un backward elimination. Aquest és un mètode utilitzat en el context de l'anàlisi de regressió i l'aprenentatge automàtic per seleccionar les característiques més rellevants en un model. L'objectiu és eliminar de manera iterativa les característiques

menys significatives del model fins a obtenir un conjunt òptim de característiques que maximitzin el rendiment del model.

La funció "`backward_elimination`" implementa el ***backward elimination*** per seleccionar les variables utilitzades en el feature function en el CRF Tagger. La funció rep un paràmetre lan que indica l'idioma ('ESP' o qualsevol altre valor) i selecciona els conjunts de dades corresponents (*loc_ESP, otros_ESP, loc_NED, otros_NED*) en funció de l'idioma proporcionat.

La funció principal de backward elimination es defineix dins de la funció "`backward_elimination`" com a "`Rbackward_elimination`". Aquesta funció realitza l'eliminació iterativa de característiques. Rep una tupla tupl que representa el conjunt de característiques a avaluar. A continuació, itera sobre cada element de la tupla i, en cada iteració, estableix el valor corresponent a zero i avalua el rendiment del model sense aquesta característica utilitzant l'objecte mod.

Després d'avaluar el model sense una característica específica, es calculen mesures de rendiment com el recall, precision, *f1_score* i *elapsed_time*. A continuació, es crea una nova tupla tpl amb els valors de característiques binàries (0 o 1) corresponents a loc1 i otros1.

La informació del model, la tupla tpl i les mesures de rendiment s'afegeixen a la llista results.

A continuació, es compara el millor *f1_score* obtingut (max(g)) amb el valor màxim anterior (maxim). Si el *f1_score* actual és més gran que el valor màxim anterior, s'actualitza maxim i es guarda la tupla tupla corresponent a aquest *f1_score* màxim en *max_mod*. A continuació, es realitza una crida recursiva a la funció "`backward_elimination`" amb la tupla *max_mod* per continuar avaluant les característiques restants.

Quan no es troben més característiques que millorin el *f1_score*, es retorna el màxim *f1_score* aconseguit (maxim) i la tupla corresponent a aquest *f1_score* màxim (tupla_millor).

Finalment, la funció "`backward_elimination`" retorna el màxim *f1_score* aconseguit, la tupla corresponent a aquest *f1_score* màxim i la llista results que conté la informació de tots els models avaluats en el procés de ***backward elimination***.

```
#FUNCIÓ BACKWARD ELIMINATION
def backward_elimination(lan):
    if lan == 'ESP':
        loc = loc_ESP
        otros = otros_ESP
    else:
        loc = loc_NED
        otros = otros_NED
    results = []
    maxim = 0
    def Rbackward_elimination(tupl):
        nonlocal maxim
        g = []
        h = []
        for i in range(len(tupl)):
            a = list(tupl)
            a[i] = 0
            tupla = tuple(a)
            cap,num,punt,suf,lon,pref,lemma,postag,ant,post,loc1,otros1 = tupla
            mod =
model(lan,'IOB',FeatureDetector(cap,num,punt,suf,lon,pref,lemma,postag,ant,post,
loc1,otros1))
            mod.train_model()
            recall,precission,f1_score,elapsed_time = mod.validate_model()
            loc_bin = 0 if loc1==0 else 1
            otros_bin = 0 if otros1==0 else 1
            tpl =
(cap,num,punt,suf,lon,pref,lemma,postag,ant,post,loc_bin,otros_bin)

results.append([mod.model_output_path,tpl,recall,precission,f1_score,elapsed_tim
e])
            h.append(tupla)
            g.append(f1_score)
        if max(g) > maxim and tupla != tupl:
            maxim = max(g)
            max_mod = h[g.index(maxim)]
            return Rbackward_elimination(max_mod)
        else:
            cap,num,punt,suf,lon,pref,lemma,postag,ant,post,loc1,otros1 = tupl
            loc_bin = 0 if loc1==0 else 1
            otros_bin = 0 if otros1==0 else 1
            tupla_millor =
(cap,num,punt,suf,lon,pref,lemma,postag,ant,post,loc_bin,otros_bin)
            return maxim, tupla_millor

    maxim_f1score, millor_model =
Rbackward_elimination((1,1,1,1,1,1,1,1,1,1,loc,otros))
    return maxim_f1score,millor_model,results
```

Hem aplicat aquesta funció per als dos idiomes i hem obtingut el millor model de cada un d'ells. Els resultats de tots els models són emmagatzemats a un fitxer .csv per no haver de tornar a executar cada vegada el ***backward elimination***, que és un procés costós. En el cas de l'Espanyol el millor model és el que utilitza totes les variables excepte el Prefix i aconsegueix un *f1* score de `0.7488590107314665`. En el cas del Neerlandès, el millor model és amb totes les variables excepte el Signe de Puntuació i arriba a un *f1* score de `0.7379040777519544`.

4.2. CODIFICACIONS

És necessari comparar diferents codificacions i veure quina és la més efectiva. Per això agafarem les feature functions dels millors models obtinguts anteriorment i executarem les diferents codificacions: *IOB, IO, BIOU, BIOEU, BIOEU+*. Per fer-ho utilitzem la funció comparar_codificacions(lan).

Aquesa funció serveix per comparar diferents codificacions utilitzades en el procés d'extreure tagd'entitats en un text. L'objectiu principal és determinar quina codificació proporciona els millors resultats en termes de precisió, record i puntuació F1.

La funció accepta un paràmetre d'entrada, lan, que és un string que representa l'idioma en el qual s'estan etiquetant les entitats. Aquesta informació és crucial per seleccionar el model més adequat per a l'idioma específic.

Dins de la funció, primer s'obté el millor model per a l'idioma especificat, que es descompon en diverses característiques (com ara cap, num, punt, etc.) que s'han seleccionat prèviament com a les més efectives per a aquest idioma.

Després, si la variable loc (que indica si s'han de considerar les localitzacions en la detecció de característiques) és 1, s'obté la llista de localitzacions per a l'idioma especificat. Similarment, si la variable otros (que indica si es consideren altres característiques addicionals) és 1, s'obté la llista d'aquestes característiques per a l'idioma específic.

A continuació, la funció itera sobre una llista de diferents esquemes de codificació (*IOB, IO, BIOU, BIOEU, BIOEU_PLUS*). Per a cada codificació, es crea un nou model amb les característiques especificades i l'esquema de codificació. Aquest model es forma i es valida, obtenint la precisió, el record, la puntuació F1 i el temps que ha durat el procés de validació.

Finalment, els resultats de cada model (codificació, precisió, record, puntuació F1 i temps transcorregut) s'afegeixen a una llista de resultats que es retorna com a sortida de la funció. Així, la funció "comparar_codificacions" proporciona una comparació directa de la eficàcia de diferents esquemes de codificació per a un idioma en particular.

```
def comparar_codificacions(lan):
    millor_model = eval(f'millor_model_{lan.lower()}')
    cap,num,punt,suf,lon,pref,lemma,postag,ant,post,loc,otros = millor_model
    results = []
    if loc == 1:
        loc = eval(f'loc_{lan}')
    if otros == 1:
        otros = eval(f'otros_{lan}')
    for encod in ['IOB','IO','BIOU','BIOEU','BIOEU_PLUS']:
        mod =
model('ESP',encod,FeatureDetector(cap,num,punt,suf,lon,pref,lemma,postag,ant,pos
t,loc,otros))
        mod.train_model()
        recall,precission,f1_score,elapsed_time = mod.validate_model()
        results.append([encod, recall,precission,f1_score,elapsed_time])
    return results
```

Per a l'Espanyol hem obtingut aquests resultats

Encoding	Recall	Precission	F1 score	Tiempo de validación
IOB	0.725347	0.773284	0.748549	0.861244
IO	0.699425	0.757061	0.727103	0.818845
BIOU	0.733716	0.772452	0.751198	0.834681
BIOEU	0.735681	0.771860	0.753337	0.935326
BIOEU_PLUS	0.721432	0.126453	0.215187	0.970124

Per al Neerlandès obtenim els següents resultats:

Encoding	Recall	Precission	F1 score	Tiempo de validación
IOB	0.721308	0.767649	0.743757	0.795422
IO	0.701873	0.760112	0.729833	1.221701
BIOU	0.733716	0.768883	0.750888	0.833488
BIOEU	0.742120	0.775062	0.758234	0.847316

Encoding	Recall	Precission	F1 score	Tiempo de validación
BIOEU_PLUS	0.724379	0.126922	0.126922	0.880945

En ambdós casos la millor codificació és *'BIOEU'* obtenint un f1 score superior a `0.75` en els dos idiomes. Encara que augmenti la complexitat del model degut al major nombre de classes, aquesta codificació és la que millor resultat dóna i el temps de validació no és suficientment superior al *IOB* com per renunciar a ell.

4.3. ANÀLISI DE MODELS

Aquest apartat serveix per acabar d'analitzar els diferents models entrenats i validats. Per aquest objectiu mostrarem una sèrie de gràfiques per a la comparació de features i de codificacions. La mètrica de rendiment en què ens focalitzarem serà el f1 score ja que és una combinació entre la recall i la precision. Així valorem tant els falsos positius com els falsos negatius.

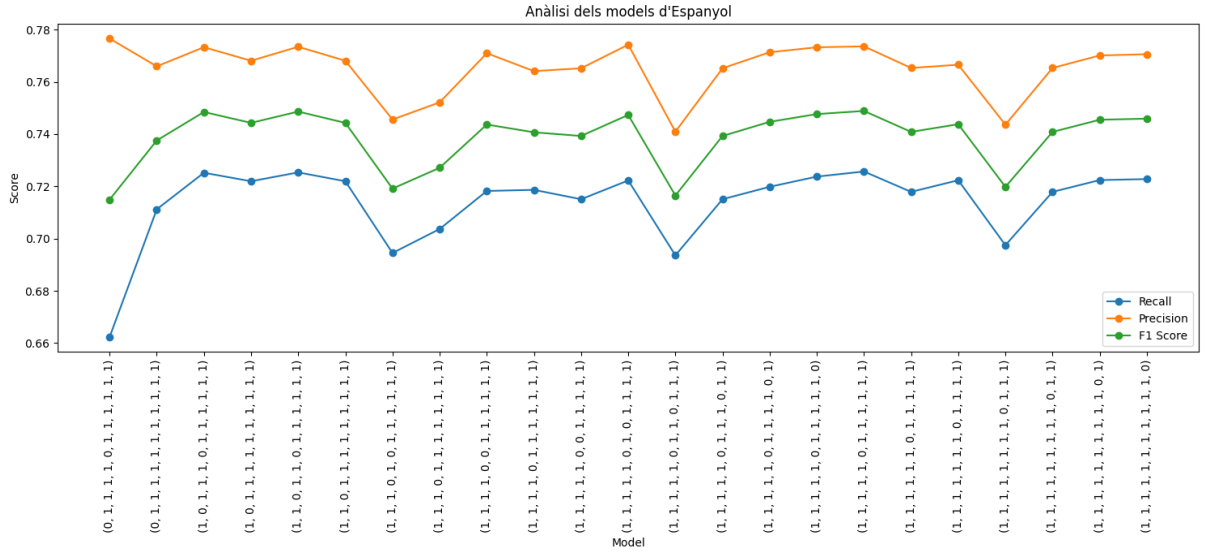
Cal mencionar que en el nostre anàlisi no tenim en compte el temps d'execució ja que l'entrenament només s'ha de realitzar una vegada ja que el model es guarda en un fitxer i el temps d'execució de la validació d'un model és suficientment baix en totes les ocasions. No supera els 5 segons i en la majoria de casos no supera un segon.

Finalment executarem les dades de test per al millor model d'Espanyol i de Neerlandès i veurem el rendiment final dels nostres models.

4.3.1 GRÀFICS

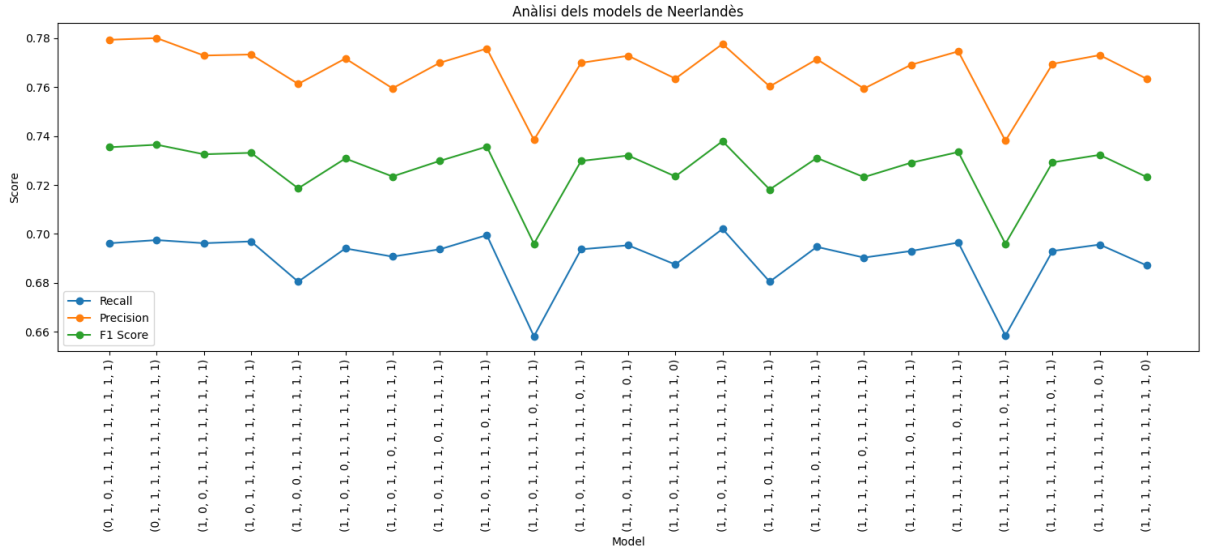
Mostrarem gràfics on es mostren les mètriques de rendiment de cada model

4.3.1.1. FEATURES

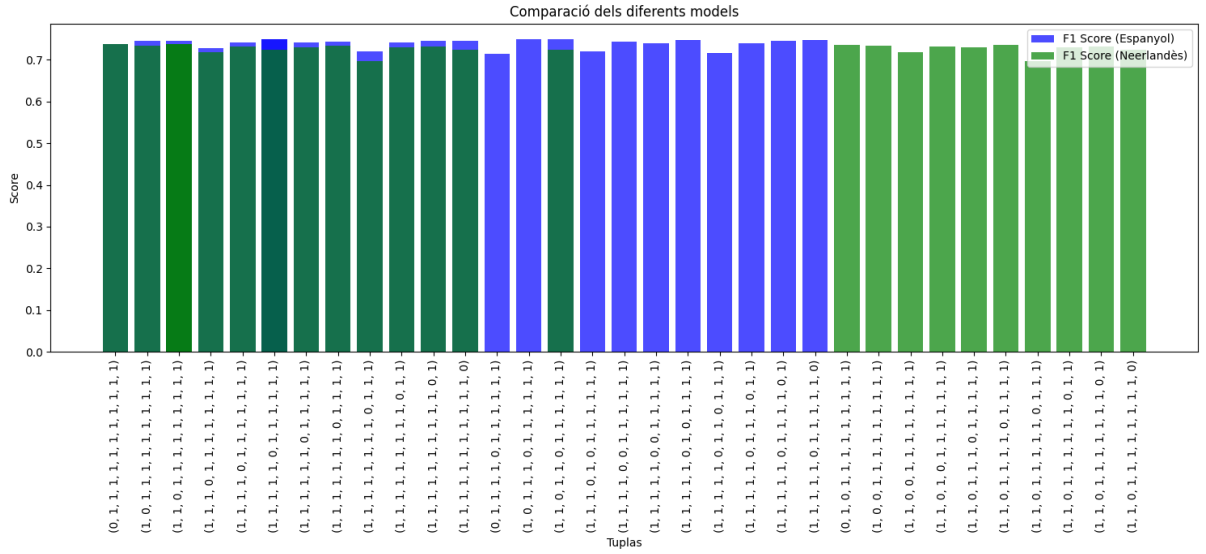


Aquí podem veure com influeixen les features en el nostre model i quines són les més importants. Pel que fa al rendiment de l'F1 Score, el model amb la configuració de característiques "(1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1)" té la puntuació més alta amb un F1 de `0.7488590107314665`. El recall i la precisió d'aquest model també són força alts, la qual cosa indica un bon equilibri entre ambdues mètriques. Aquest model no utilitza la funció de prefix a la seva configuració de característiques, per tant, sembla que la característica de prefix potser no és tan útil per a aquests models.

El model amb el rendiment més baix en termes de F1 Score és el model amb la configuració "(0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1)" amb un F1 de `0.714925954793453`. Aquest model no utilitza les funcions de capitalització ni prefix. Això demostra que la lletra majúscula és una variable important. Cal mencionar que per aquest model, la recall és molt més baixa a la dels altres models i la precision és la més alta. Això significa que hi ha moltes entitats que no es troben, però les que es troben es fa amb una alta precisió. Aquest és el model que més difereix en les diferents mètriques.

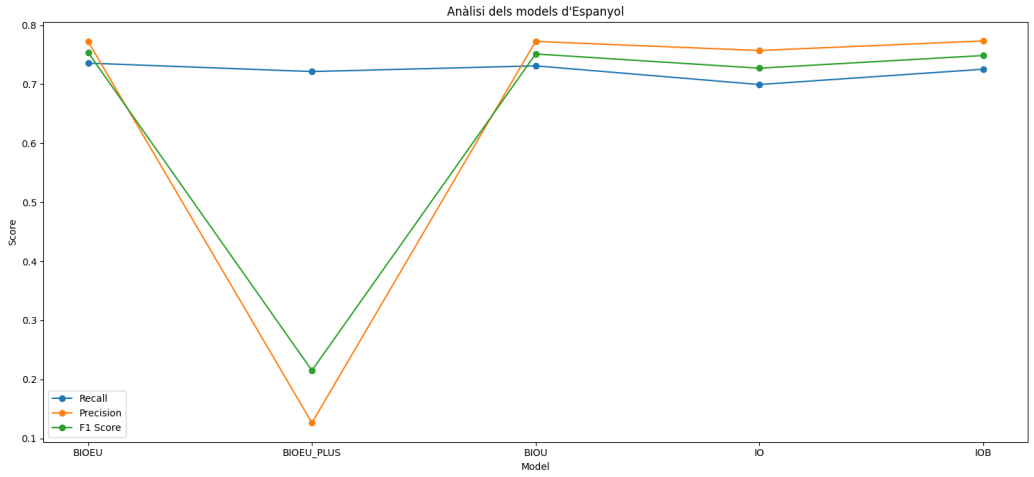


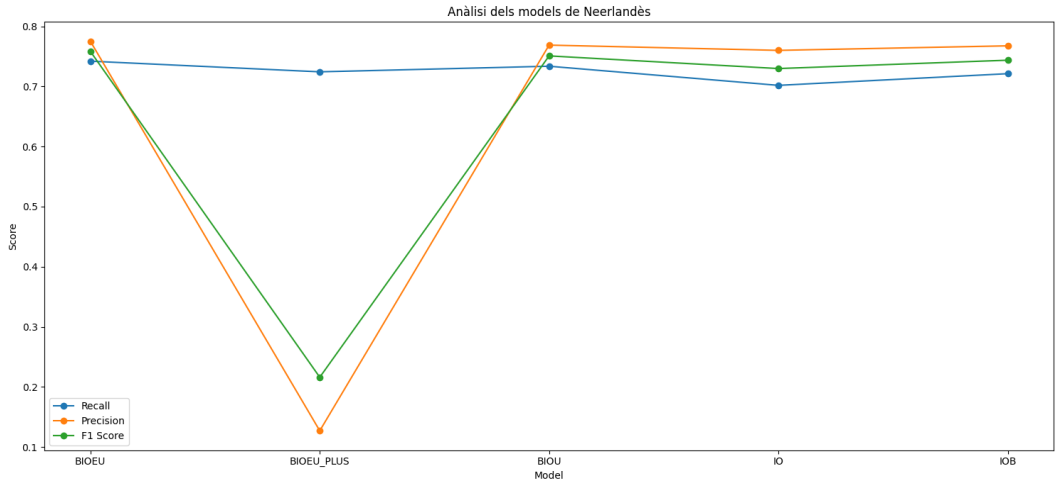
El model amb el millor F1 score és el *model 2* (IOB_NED_110111111111.mdl) amb un F1 score de `0.7379`. Aquest model també té un recall de `0.7020` i una precisió de `0.7777`, amb un temps de validació de `1.5232`. Aquest resultat suggereix que aquest model presenta un bon equilibri entre precisió i recall. Per tant, podem dir que la puntuació no és una característica important per al model en Neerlandès.



Aquí podem veure els resultats de tots els models per ambdós idiomes. Es veuen resaltats les majors f1 scores; del Neerlandès a la tercera barra i d'Espanyol a la sisena.

4.3.1.2. ENCODINGS





Ambdues llengües es comporten de forma similar en quant a les diferents codificacions. La majoria de codificacions tenen rendiments similars i una diferència mínima entre diferents mètriques, però el model BIOEU+ pateix d'una precisió molt baixa. Això significa que es troben moltes entitats falses, és a dir, detecta enitats nominals que no ho són.

En quant al millor model, és el BIOEU ja que té el f1 score més alt arribant a `0.753337` per a l'Espanyol i `0.758234` per al Neerlandès.

4.3.2 MODEL FINAL

Amb aquests anàlisis hem pogut obtenir els paràmetres dels millors models per a l'Espanyol i per al Neerlandès. Finalment testejarem els nostres models finals que tenen les següents característiques:

Espanyol: codificació *BIOEU* i totes les features excepte el prefix

Neerlandès: codificació *BIOEU* i totes les features excepte el signe de puntuació

Per obtenir el rendiment final del model, cal utilitzar el mètode "`model.test_model`" però ho farem amb una funció adicional "`testejar_model`" per obtenir les característiques a partir de les variables emmagatzemades en l'experimentació dels paràmetres. En aquesta funció es dona l'etiqueta del llenguatge i es realitza tot lo necessari per obtenir el rendiment obtingut amb les dades de test. Primerament s'obtenen les llistes de gazetteers en cas de que el paràmetre estigui activat, després es crea el model i s'entrena. Finalment s'utilitza el mètode test_model per obtenir les mètriques finals del model.

```
def testejar_model(lan):
    cap,num,punt,suf,lon,pref,lemma,postag,ant,post,loc_bin,otros_bin =
eval(f'millor_model_{lan.lower()}')
    if loc_bin == 1:
        loc = eval(f'loc_{lan}')
    else:
        loc = 0
    if otros_bin == 1:
        otros = eval(f'otros_{lan}')
    else:
        otros = 0
    codific = eval(f'millor_cod_{lan.lower()}')
    mod =
model(lan,codific,FeatureDetector(cap,num,punt,suf,lon,pref,lemma,postag,ant,pos
t,loc,otros))
    mod.train_model()
    recall, precission, f1_score = mod.test_model()
    return recall, precission, f1_score
```

Els resultats per a l'Espanyol són:

```
Recall: 0.7819422033152413
Precision: 0.8080187964226163
F1 score: 0.7947666616967347
```

Els resultats per al Neerlandès són:

```
Recall: 0.7499660740941784
Precision: 0.808558888076079
F1 score: 0.7781610813855252
```

Així doncs els models finals tenen rendiment (en relació a totes les mètriques) superior al 78% a l'Espanyol i al 74% al Neerlandès.

5. PROVA EL TEU PROPI TEXT

L'apartat "5. PROVA EL TEU PROPI TEXT" es centra en la importància de la prova del model amb dades del món real, a més de les dades de validació. La validació és, per descomptat, un pas essencial en el procés de desenvolupament del model, però no pot captar necessàriament totes les peculiaritats i la complexitat del llenguatge humà tal com es fa servir en situacions reals. A més, les dades de validació utilitzades durant el entrenament poden no ser representatives de tots els tipus de textos amb els que el model pot haver de tractar en aplicacions reals.

Per tant, en aquesta secció, hem proporcionat una interfície amb "*widgets*" que permet a l'usuari interactuar directament amb el model. L'usuari pot introduir el seu propi text i seleccionar el model que prefereixi, amb la possibilitat de triar entre diferents idiomes, codificacions i característiques. Aquesta flexibilitat permet a l'usuari experimentar amb diverses configuracions i veure com afecten els resultats.

Per exemple, l'usuari pot escollir provar el model entrenat amb l'idioma espanyol o neerlandès, pot seleccionar diferents tipus de codificacions i triar entre una varietat de característiques. Això no només permet a l'usuari veure com el model reacciona a diferents tipus de textos, sinó també com es comporta sota diferents paràmetres i configuracions.

A més, aquesta secció també discuteix l'ús de textos de diaris com a part del procés de prova. Aquests textos ens proporcionen una mostra rica i diversa del llenguatge humà tal com es fa servir en el món real. Ens permeten veure com el model es comporta amb diferents tipus de notícies, articles d'opinió, reportatges, etc. Això ens ajuda a comprendre millor com de útils són els models quan són provats amb dades del món exterior, i no només amb les dades de validació utilitzades durant el procés de entrenament.

5.1. INTERFÍCIE

L'objectiu d'aquesta interfície és proporcionar una eina interactiva i fàcil d'utilitzar que permeti a l'usuari interactuar directament amb el model i les seves funcions.

L'interfície està construïda amb *widgets*, que són elements d'interfície d'usuari que permeten a l'usuari interactuar amb el codi. Aquests widgets inclouen una sèrie de botons de commutació, quadres de selecció i àrees de text.

Hi ha botons de commutació per seleccionar l'idioma i el tipus d'encoding que s'utilitzarà per al text d'entrada. L'usuari pot triar entre l'espanyol i el neerlandès per a l'idioma, i entre diferents tipus d'encoding.

A continuació, l'usuari pot seleccionar les característiques que vol que el model consideri. Això es fa amb una sèrie de quadres de selecció que representen diferents característiques, com la capitalització, els números, els signes de puntuació, etc.

L'usuari també té la possibilitat d'introduir el seu propi text en un àrea de text que es proporciona. Aquest text es pot introduir directament o es pot pegar des d'una altra font.

Finalment, hi ha un botó que l'usuari pot prémer per fer que el model processi el text d'entrada segons les opcions seleccionades. Una vegada pressionat el botó, el model generarà una sortida que es mostrarà a l'usuari.

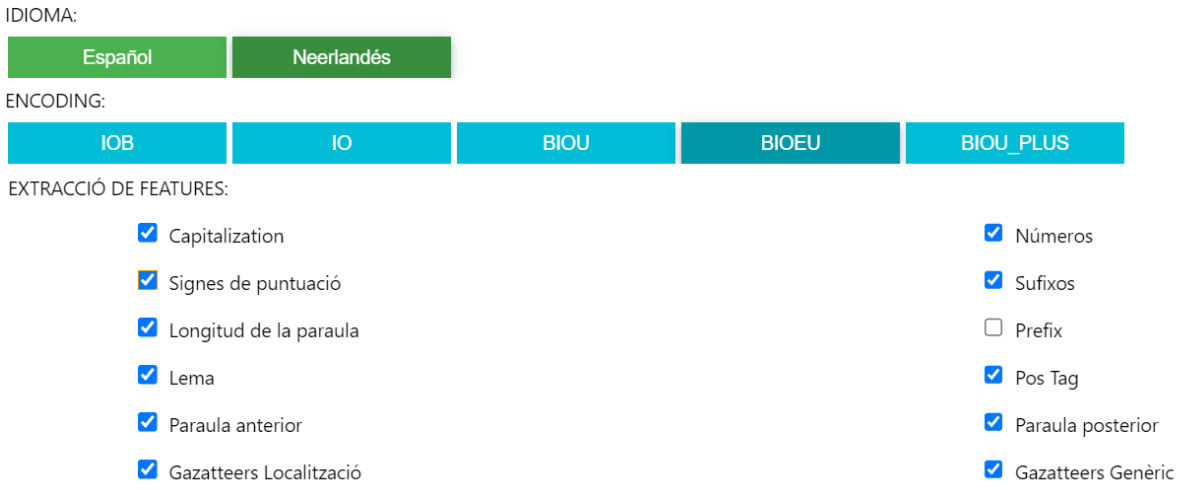
Tot i que no entrarem en detalls sobre el codi específic que fa funcionar aquests widgets, és important destacar que l'objectiu principal d'aquesta interfície és fer que la interacció amb el model sigui tan senzilla i intuïtiva com sigui possible.

5.2. PROVANT TEXTOS

En aquest apartat provarem diferents textos amb el millor model obtingut per cada idioma.

5.2.1. ESPANYOL

Per seleccionar el millor model per a l'Espanyol cal seleccionar els següents valors dels paràmetres:



Veiem els resultats per a diferents textos:

TEXT 1: extret amb el Chat GPT 3 amb la query: ‘Dame un texto en español adecuado para testear mi modelo de Name entity Recognition’:

El presidente Pedro Sánchez se reunió hoy con la canciller alemana Angela Merkel en Berlín . Ambos líderes discutieron temas relacionados con el cambio climático y la cooperación económica entre España y Alemania . Además , Sánchez anunció que España acogerá la próxima cumbre internacional sobre energías renovables . Durante la reunión , también se abordaron asuntos de seguridad y lucha contra el terrorismo . Sánchez destacó la importancia de la colaboración internacional para hacer frente a estos desafíos . Merkel elogió los esfuerzos de España en la gestión de la crisis migratoria y expresó su apoyo a las políticas de integración . En otro orden de cosas , el famoso actor Antonio Banderas presentó su nueva película en el Festival de Cannes . El largometraje , titulado 'El último viaje ' , cuenta la historia de un explorador que se aventura en un viaje épico por diferentes continentes . La película ha recibido excelentes críticas y se espera que sea un gran éxito en taquilla . En el ámbito deportivo , el tenista Rafael Nadal se prepara para competir en el torneo de Roland Garros . Nadal , considerado uno de los mejores jugadores de tenis de todos los tiempos , buscará obtener su decimocuarto título en el prestigioso campeonato . Finalmente , en noticias de tecnología , la empresa española TeleSoft anunció el lanzamiento de su nuevo dispositivo inteligente . El producto , llamado SmartHome 2.0 , ofrece soluciones avanzadas para la automatización del hogar , permitiendo a los usuarios controlar luces , electrodomésticos y sistemas de seguridad desde sus teléfonos móviles .

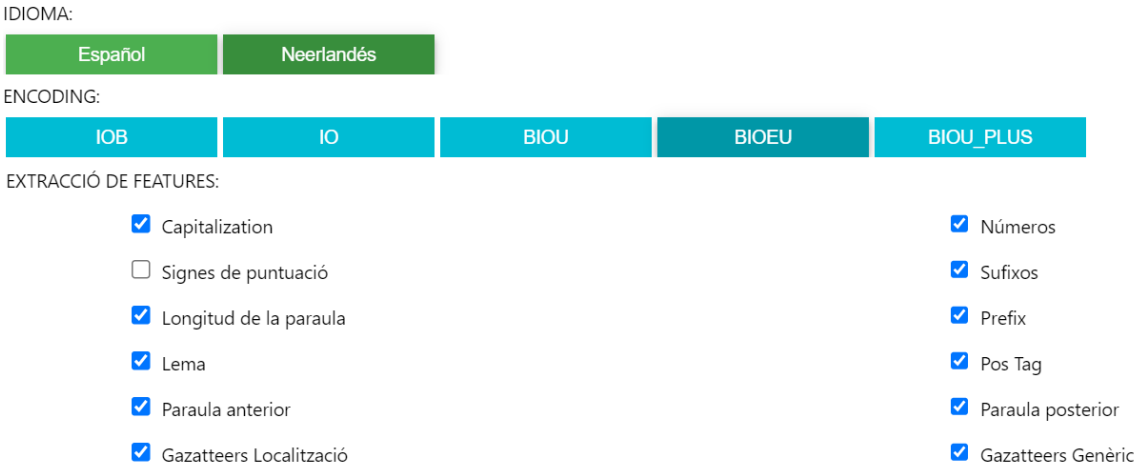
TEXT 2:
https://www.publico.es/mujer/informe-policial-infancia-libre-elaboro-declaraciones-padres-datos-falseados-inculpar-madres.html#md=modulo-portada-bloque:4col-t5;mm=mobile-big

A finales de mayo de 2019 , A.C.R . viajó de Granada a Madrid para denunciar ante la unidad de la Policía Nacional adscrita a los juzgados de Plaza de Castilla que su exmujer , E.J.G. , actuaba según el mismo modus operandi que otras que por entonces salían en medios de comunicación . La detención de María Sevilla (presidenta de la asociación Infancia Libre) a principios de abril de ese año y la de otras dos mujeres a las que se ligó a la organización casi un mes más tarde coparon de una forma sorprendente los espacios informativos y las tertulias televisivas . A.C.R. había sabido por las noticias (o así lo afirmó) que esa unidad policial estaba elaborando un informe con distintos casos y que se entrevistaba con padres que afirmaban que sus mujeres también encajaban en la supuesta trama . En su denuncia ante los agentes , explicó que su expareja lo había denunciado por malos tratos psicológicos hacia su hija y que la menor había sido atendida por la pediatra de Granada N.P.U. , a pesar de que no era su médica de

referencia.Esta pediatra había sido señalada por la Policía judicial como una de las colaboradoras necesarias de la trama junto a la psiquiatra A.M.R. de Granada y al psiquiatra A.E.N. de Madrid . Según la Policía , se dedicaban a realizar informes `` falsos '' y `` a medida '' para estas madres .

5.2.2. NEERLANDÈS

Per seleccionar el millor model per al Neerlandès cal seleccionar els següents valors dels paràmetres:



Veiem els resultats per a diferents textos:

TEXT 1: extret amb el Chat GPT 3 amb la query: 'Dame un texto en neerlandés adecuado para testear mi modelo de Name entity Recognition':

De premier Mark Rutte kondigde vandaag nieuwe maatregelen aan om de verspreiding van het coronavirus te beperken . Alle restaurants , cafés en niet-essentiële winkels zullen voor de komende drie weken gesloten zijn . Alleen supermarkten en apotheken blijven open voor essentiële boodschappen en medicijnen . Daarnaast heeft koning Willem-Alexander aangekondigd dat hij een staatsbezoek zal brengen aan Canada volgende maand . Het bezoek heeft tot doel de bilaterale betrekkingen tussen Nederland en Canada te versterken en samenwerking op verschillende gebieden te bevorderen , waaronder handel , cultuur en technologie . In het sportnieuws heeft het Nederlandse voetbalteam een belangrijke overwinning behaald in hun kwalificatiewedstrijd voor het Europees kampioenschap . Ze versloegen het team van Frankrijk met 2-1 dankzij doelpunten van Memphis Depay en Frenkie de Jong . Op het gebied van technologie lanceerde het Nederlandse bedrijf TechVision zijn nieuwe virtual reality-headset . De headset , genaamd VRX-500 , biedt gebruikers een meeslepende ervaring met geavanceerde grafische mogelijkheden en realistisch surround-geluid . Tot slot opende het Van Gogh Museum in Amsterdam een nieuwe tentoonstelling ter ere van de 100e verjaardag van

Vincent van Gogh 's schilderij 'De Sterrennacht ' . De tentoonstelling toont verschillende werken van de beroemde Nederlandse kunstenaar en biedt inzicht in zijn leven en artistieke ontwikkeling.

TEXT 2:
https://www.telegraaf.nl/nieuws/1246688829/ontvoerde-kayla-15-in-vs-na-zes-jaar-teruggevonden-dankzij-netflixserie

Slechts negen jaar oud was Kayla Unbehaun toen haar moeder Heather haar 4 juli 2017 weghaalde bij haar vader . Haar moeder had alleen bezoekrechten , de volledige voogdij lag bij Kayla ' s vader . Toen hij de volgende dag zijn dochter wilde ophalen bij zijn ex-vrouw , waren ze allebei verdwenen . Nadat er jarenlang geen enkel spoor van Kayla en Heather te bekennen was , kwam haar ontvoering aan het licht in een aflevering van de Netflixserie 'Unsolved Mysteries ' . Ook werd vorige maand een foto vrijgegeven door het Amerikaanse National Center for Missing and Exploited Children (NCMEC) met hoe onderzoekers dachten dat Kayla er nu uit zou zien . En dus niet zonder succes . Een winkelmedewerker in de plaats Asheville herkende Kayla omdat hij de serie had gezien . De politie kwam ter plaatse en kon Heather arresteren . Kayla werd herenigd met haar vader . Heather wordt verdacht van kinderontvoering en zit nog altijd vast .

6. CONCLUSIONS

En aquest treball s'ha implementat un reconeixedor d'entitats anomenades (NER) utilitzant els Conditional Random Fields (CRF) i la biblioteca Natural Language Toolkit (NLTK). A més, s'han explorat diverses característiques i codificacions per millorar el rendiment del model.

1. **Treball en espanyol:**

- El millor model utilitza la codificació BIOEU i totes les característiques excepte el prefix.
- El model final ha estat avaluat en textos reals i ha obtingut un rendiment prometedor:
    - Recall: 0.7819422033152413
    - Precisió: 0.8080187964226163
    - F1 score: 0.7947666616967347

2. **Treball en neerlandès:**

- El millor model utilitza la codificació BIOEU i totes les característiques excepte el signe de puntuació.
- El model final ha estat avaluat en textos reals i ha obtingut un rendiment prometedor:
    - Recall: 0.7499660740941784
    - Precisió: 0.808558888076079
    - F1 score: 0.7781610813855252

En ambdós treballs, s'ha demostrat la importància de les característiques i codificacions en el rendiment del model de NER. Les codificacions BIOEU han mostrat un bon rendiment en els dos idiomes, obtenint f1 scores superiors al 75%. Això indica que aquesta codificació és eficaç en la detecció d'entitats anomenades en textos en espanyol i neerlandès.

Els models finals han estat avaluats en dades de test i han mostrat un rendiment satisfactori en termes de recall, precisió i f1 score. Això indica que els models són capaços de reconèixer entitats anomenades amb una alta precisió i recordar-ne la majoria. No obstant això, cal tenir en compte que el rendiment pot variar en funció del tipus de text i del domini.

7. BIBLIOGRAFIA

1. Geonames.org. (2023). Geonames Database. https://download.geonames.org/export/dump/

2. Naamkunde.net. (2023). Databank van Achternamen in Nederland en Vlaanderen. http://www.naamkunde.net/?page_id=293

3. Instituto Nacional de Estadística (INE). (2023). INEbase. https://www.ine.es/dyngs/INEbase/es

4. GitHub. (2023). GitHub Repository. https://github.com/

5. Público. (2023, Maig 14). Informe policial 'Infancia Libre' elaboró declaraciones de padres con datos falseados para inculpar a madres. https://www.publico.es/mujer/informe-policial-infancia-libre-elaboro-declaraciones-padres-datos-falseados-inculpar-madres.html#md=modulo-portada-bloque:4col-t5;mm=mobile-big

6. De Telegraaf. (2023, Maig 14). Ontvoerde Kayla (15) in VS na zes jaar teruggevonden dankzij Netflixserie. https://www.telegraaf.nl/nieuws/1246688829/ontvoerde-kayla-15-in-vs-na-zes-jaar-teruggevonden-dankzij-netflixserie