UNIVERSITÀ DI PISA

Corso di Laurea in Informatica Umanistica

TESI DI LAUREA

Il segreto del successo: studio delle caratteristiche e della predizione del successo nel genere fanfiction

Relatore:
Felice Dell'Orletta

Correlatore:
Alessandro Lenci

Candidato:
Giulio Leonardi

ANNO ACCADEMICO 2022/2023

Indice

1. Introduzione

Questo studio deriva da un'esperienza svolta all'Istituto di Linguistica Computazionale "Antonio Zampolli" del Consiglio Nazionale delle Ricerche. L'obiettivo è stato quello di comprendere se esistono delle caratteristiche linguistiche e lessicali (chiamate anche feature) di un testo correlate al successo e valutare se, a partire da esse, sia possibile addestrare un algoritmo di apprendimento automatico capace di predire il successo di un racconto.

I testi utilizzati per l'analisi fanno parte di un genere preciso: la *fanfiction*, racconti basati sull'universo narrativo di un'opera madre, scaricati da uno dei più grandi archivi italiani online di storie amatoriali. Sono stati selezionati i testi basati sul mondo dei romanzi di *Harry Potter* e *Il Signore degli Anelli*.

Le feature che sono state estratte dai testi si dividono in due gruppi: linguistiche e lessicali. Le prime sono state ricavate mediante una fase di profiling linguistico, mentre quelle lessicali attraverso l'estrazione delle sequenze di forme, lemmi, parti del discorso, prefissi e suffissi dai testi. Sono state quindi studiate le possibili differenze nelle prestazioni e nei comportamenti dell'algoritmo di predizione in base al tipo di caratteristiche utilizzate per l'addestramento.

Lo scopo della tesi è quello di rispondere alle seguenti domande di ricerca:

1. Esistono delle caratteristiche linguistiche e lessicali significative per il successo di un racconto?
2. Quali tra queste caratteristiche sono maggiormente correlate al successo?
3. È possibile prevedere il successo di un racconto attraverso queste caratteristiche?
4. Le caratteristiche cambiano rispetto al dominio tematico dei testi?
5. Le caratteristiche cambiano nel tempo?
6. Per la predizione del successo le caratteristiche linguistiche e lessicali si comportano diversamente?

Nel capitolo 2 si tratterà la creazione del dataset, quindi a partire dall'estrazione dei testi attraverso un algoritmo di *crawling*, fino all'analisi del dataset completo, per scegliere quali racconti siano da considerare di "successo" e quali invece di "insuccesso", attraverso il numero di recensioni ottenute nel proprio primo capitolo.
Il secondo capitolo invece si concentrerà sullo studio delle feature: sarà descritto il tool di monitoraggio linguistico e le metodologie utilizzate per analizzare i testi e ricavare le caratteristiche linguistiche e lessicali; infine saranno effettuate delle analisi statistiche per valutare la significatività delle feature.
Gli ultimi due capitoli tratteranno la classificazione, ovvero il task di assegnare una classe (insuccesso o successo) ai testi attraverso un algoritmo di apprendimento automatico. Nel capitolo 4 si presenteranno le tecnologie utilizzate per la classificazione, e i vari scenari e modelli alla base degli esperimenti. Nel capitolo 5 invece si riporteranno i risultati degli esperimenti, che saranno discussi sia singolarmente, che a paragone tra loro, per valutare i comportamenti dei modelli linguistici rispetto a quelli lessicali.
Nelle conclusioni si riassumeranno le osservazioni emerse durante le diverse fasi dello studio, tentando di rispondere alle domande precedentemente elencate.

2. Creazione del dataset

In questo capitolo innanzi tutto sarà presentato EfpFanfic[1], il sito che contiene le storie che andranno a formare il dataset alla base di questo studio. Successivamente si tratterà il metodo di estrazione, ovvero l'algoritmo di scraping utilizzato per ricavare i racconti e i relativi metadati dalle pagine del sito. Inoltre, si mostrerà la fase di pulizia dei testi e preparazione al monitoraggio linguistico del dataset. Infine i corpora saranno descritti e analizzati, allo scopo di trovare una definizione formale di successo e insuccesso dei racconti.

2.1 Estrazione dei testi

Il genere dei testi studiati è la *fanfiction*: racconti scritti da ammiratori di un'opera che si ispirano a essa nella trama e nei personaggi. Efpfanfic è tra i più importanti archivi di racconti amatoriali italiani, fondato nel 2001, oggi è ancora attivo e conta oltre 500000 storie e 6 milioni di recensioni. Il sito permette agli utenti di pubblicare dei racconti, eventualmente divisi in capitoli, per poi ricevere interazioni da parte dei lettori.
La Homepage del sito presenta la prima divisione in due macrosezioni: storie originali e fanfiction. La sezione fanfiction è a sua volta divisa in categorie che riguardano la forma artistica dell'opera madre; cliccando una di queste apparirà la lista delle opere; cliccando su un'opera sarà mostrato l'elenco dei racconti fanfiction al riguardo.

[1] *EFP Fanfiction*. URL: `https://efpfanfic.net/`.

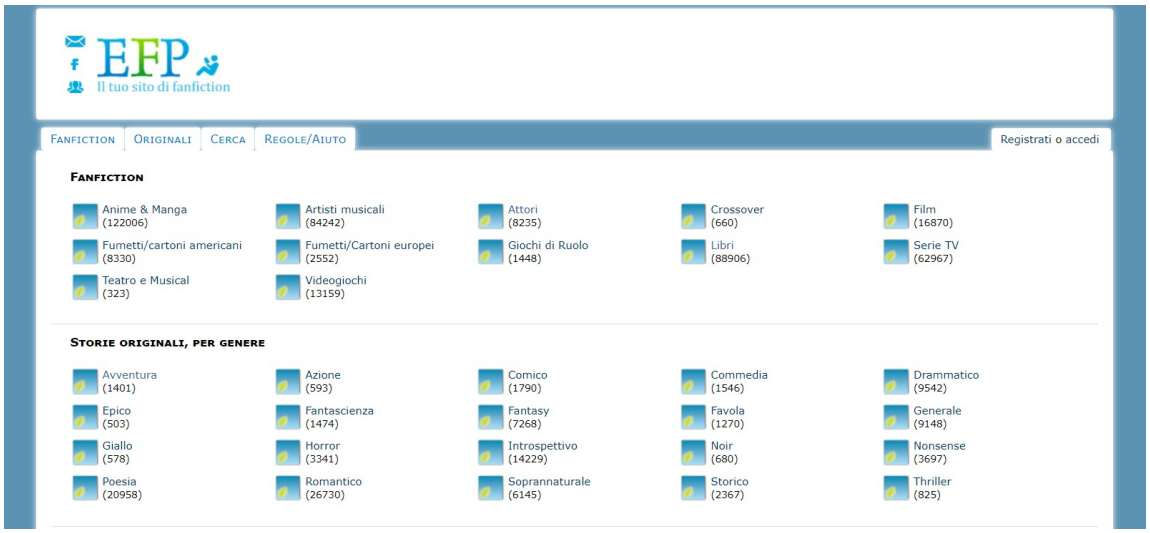

Figura 2.1: Home page di EFP Fanfiction.

Le storie del portale sono strutturate in capitoli, che possono essere pubblicati singolarmente. A ogni storia sono associati diversi metadati, come il nome dell'autore, la trama del racconto, il genere e i personaggi principali. Altri metadati sono invece associati al singolo capitolo: la data di pubblicazione e il numero di recensioni ricevute.

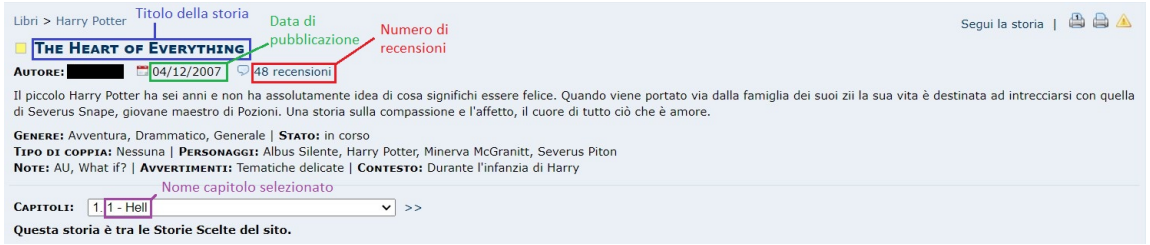

Figura 2.2: Alcuni metadati evidenziati su un capitolo di EFP Fanfiction.

I lettori, per ogni capitolo di un racconto, possono scrivere una recensione; questa sarà contrassegnata da una bandierina che esprimerà il giudizio: verde per positivo, blu per neutrale e rossa per negativo. In questo studio non si terrà conto del giudizio delle recensioni, che quindi saranno considerate come valore assoluto.

I testi estratti riguardano due diverse opere: *Harry Potter* di J.K. Rowling e *Il Signore degli Anelli* di J.R.R. Tolkien. Quindi, si costituiranno due corpora separati, che verranno analizzati in modo parallelo durante questo lavoro. Inoltre, sarà selezionato solo il primo capitolo di ogni storia, considerato come rappresentativo dell'intero racconto.

Per l'estrazione è stato utilizzato un *crawler*, ovvero un programma che naviga nel web e scarica automaticamente dati e contenuti testuali.
Il crawler adoperato è il primo dei due sviluppati da Andrea Mattei nel suo elaborato di laurea triennale, programmato in linguaggio Python sfruttando la libreria Scrapy[2].
I risultati dell'estrazione sono due file in formato json, uno riferito ai capitoli a tema Harry Potter e l'altro ai capitoli a tema Signore degli Anelli. I file contengono rispettivamente 57196 e 2441 testi, scritti tra il 2003 e il 2023.
Lo schema dei dataset json ottenuti è un array di oggetti contenenti delle stringhe chiave associate a un valore; questa struttura dati una volta importata in un programma Python diventerà una lista di dizionari. Ogni oggetto json rappresenta un capitolo, ogni coppia chiave-valore all'interno rappresenta uno dei dati estratti per quel capitolo. La lista delle chiavi è uguale per ogni oggetto, quindi possiamo vedere il tutto come una tabella dove ogni capitolo rappresenta una riga.

2.2 Pulizia dei testi

Il contenuto testuale dei dataset presenta diverse difformità che rischierebbero di compromettere l'efficacia dell'analisi linguistica automatica; è stata quindi necessaria una fase di pulizia dei corpora.
Le anomalie eliminate non riguardano inesattezze grammaticali o sintattiche, ma errori nella spaziatura e nei ritorni a capo. Inoltre, sono stati rimossi alcuni caratteri molto rari che creano problemi al tool linguistico che è stato utilizzato per l'analisi.
La pulizia è stata svolta dalla funzione *cleantext* definita nel file *clean.py*. Per poter trovare e sostituire questi caratteri nel testo sono state utilizzate le espressioni regolari: sequenze di caratteri o simboli che permettono di definire un pattern preciso che poi potrà essere ritrovato nel testo.
Le espressioni regolari sono implementate in Python attraverso il modulo re[3]; in particolare è stata utilizzata la funzione *sub*, che permette di sostituire uno o più caratteri di una

[2] *Scrapy*. URL: `scrapy.org`.

[3] *Documentazione Python, voce Regular expression operations*. URL: `https://docs.python.org/3/library/re.html`.

stringa, selezionati attraverso un'espressione regolare.
Di seguito la lista delle correzioni effettuate:

- sostituiti i caratteri "U+2028" e "U+0085" con dei ritorni a capo;
- sostituito il carattere "U+2026" con tre puntini di sospensione;
- corretti i casi in cui sono stati utilizzati più di tre punti per i puntini di sospensione;
- aggiunto uno spazio se non presente dopo i segni di interpunzione;
- rimossi gli spazi presenti prima dei segni di interpunzione;
- eliminati gli spazi ripetuti.

2.3 Descrizione dei corpora e identificazione delle classi

Una volta ottenuti i corpora di testi, è stato necessario analizzarli quantitativamente per poter definire due classi, che rappresentino le storie che hanno riscosso un certo successo e le storie che invece sono considerabili di insuccesso.
Il successo di una storia sarà stabilito dal numero di recensioni ottenute dal suo primo capitolo. L'obiettivo è trovare due valori soglia: numero massimo di recensioni per assegnare la classe "insuccesso" e numero minimo di recensioni per assegnare la classe "successo".
Per stabilire i valori soglia è stato innanzi tutto necessario rappresentare, per ogni anno, come sono distribuite le storie in base al numero di recensioni sul proprio primo capitolo.
Nel file *listofdicts.py* è stata definita la classe *listofdicts*, che permette di interagire con liste di dizionari che condividono l'insieme delle chiavi.
Con il programma *datasetsToCharts.py* sono state inizializzate due istanze della classe *listofdicts*, una per corpus. I dataset sono stati internamente suddivisi per anno attraverso il metodo *splitfor*, che, preso in input un campo dei dizionari contenuti, suddivide il dataset in base al valore di quel campo; l'oggetto restituito è un dizionario nel quale ogni anno è una chiave, alla quale è associata una lista di dizionari. Ognuna di queste liste risultanti è stata inizializzata come oggetto *listofdicts*. A questo punto è stato usato il metodo *groupby*, che raggruppa i dizionari in base al valore di un campo, associandogli il conteggio dei

dizionari raggruppati da esso; quindi applicato al campo ”This_Rec” associa al numero di recensioni nel primo capitolo il conteggio delle storie. Infine è stato usato il metodo *to_csv* per riportare questi dati in un file in formato csv.

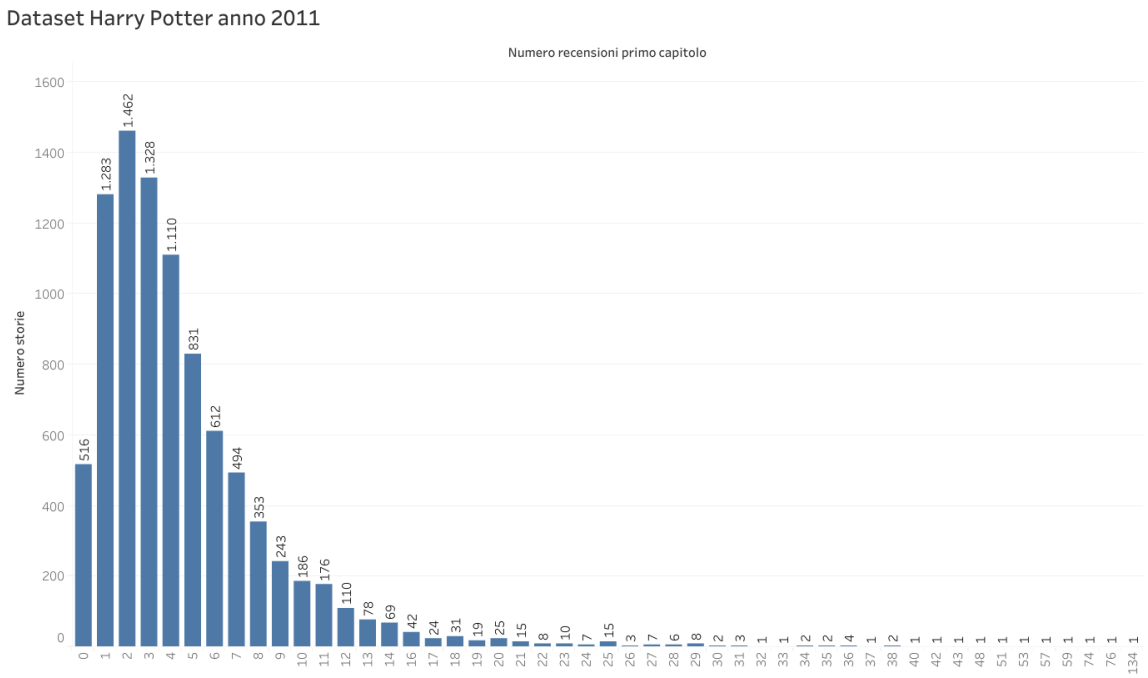

Figura 2.3: Distribuzione delle recensioni nei primi capitoli a tema Harry Potter dell’anno 2011.

Dopo aver analizzato i dati sono state considerate tutte le combinazioni tra: due possibili soglie per l’insuccesso, 0 e 1 recensione; e tre per il successo, 5, 6 e 7 recensioni.
Inoltre, si è deciso di trattare anche un terzo corpus, contenente i testi a tema Harry Potter pubblicati esclusivamente nell’anno 2011. Quindi lo studio, da questa fase in poi, si è concentrato su tre corpora: il primo contenente tutti i testi a tema Harry Potter, il secondo contenente i testi a tema Harry Potter dell’anno 2011, e il terzo contenente tutti i testi a tema Il Signore degli Anelli; saranno d’ora in poi identificati rispettivamente con le sigle HPTOT, HP2011 e LOTR. Per ognuno di essi ci sono sei possibili combinazioni dei valori soglia:

- $insuccesso = 0$ recensioni e $successo \geq 5$ recensioni;
- $insuccesso = 0$ recensioni e $successo \geq 6$ recensioni;

- *insuccesso* $= 0$ recensioni e *successo* ≥ 7 recensioni;
- *insuccesso* ≤ 1 recensioni e *successo* ≥ 5 recensioni;
- *insuccesso* ≤ 1 recensioni e *successo* ≥ 6 recensioni;
- *insuccesso* ≤ 1 recensioni e *successo* ≥ 7 recensioni.

Per ogni combinazione è stato calcolato il numero di testi che raccoglie.

-	$i \leq 0; s \geq 5$	$i \leq 0; s \geq 6$	$i \leq 0; s \geq 7$	$i \leq 1; s \geq 5$	$i \leq 1; s \geq 6$	$i \leq 1; s \geq 7$
HPTOT	25476	20997	17596	33912	29433	26032
HP2011	3914	3083	2472	5197	4366	3755
LOTR	951	753	573	1310	1112	932

Tabella 2.1: Numero di testi dei corpora.

Successivamente, per ogni corpus è stato calcolato il bilanciamento, ovvero il rapporto tra il numero di testi della classe meno frequente e il numero di testi della classe più frequente. Quindi un bilanciamento del 100% indica che le classi hanno lo stesso numero di esempi, mentre un bilanciamento del 50% indicherebbe che una classe è rappresentata dal doppio dei testi dell'altra.

-	$i \leq 0; s \geq 5$	$i \leq 0; s \geq 6$	$i \leq 0; s \geq 7$	$i \leq 1; s \geq 5$	$i \leq 1; s \geq 6$	$i \leq 1; s \geq 7$
HPtot	22%	28%	36%	63%	80%	99%
HP2011	15%	20%	26%	53%	70%	92%
LOTR	21%	28%	41%	67%	90%	77%

Tabella 2.2: Bilanciamento dei corpora.

In conclusione, è stata scelta la combinazione di valori soglia che in media permette di avere dei corpora più bilanciati: i testi con zero o una recensione sono stati considerati di insuccesso, mentre quelli con sette o più recensioni sono stati considerati di successo.

3. Monitoraggio linguistico

Nella prima parte di questo capitolo si tratterà la fase di estrazione delle feature dai testi, sarà quindi presentato il tool di profiling linguistico, e il metodo utilizzato per ricavare le caratteristiche lessicali. Nella seconda parte invece sarà calcolato il Mann-Whitney U test sulle caratteristiche linguistiche per valutare la significatività statistica di ogni feature nel distinguere le due classi.

3.1 Estrazione delle feature

Questo studio ha trattato due tipi di feature, quelle linguistiche e quelle lessicali, entrambe sono presentate in questa sezione.

Per l'estrazione delle feature dai corpora è stato utilizzato Profiling-UD[1]: tool con interfaccia web, sviluppato dall'Italian Natural Language Processing Lab[2], che realizza il profiling linguistico di corpora testuali per un vasto numero di lingue, implementando la strategia di analisi descritta da Montemagni (2013).

La prima fase del processo è l'annotazione linguistica, svolta automaticamente utilizzando UDPipe, programma realizzato dall'Institute of Formal and Applied Linguistics della Charles University, e incluso nell'iniziativa Universal Dependecies (UD)[3]. Come spiegato da Straka et al. (2006), l'annotazione viene effettuata dividendo il testo in frasi e poi in token; successivamente ad ogni token viene assegnata la sua categoria grammaticale (POS tagging), ed il suo lemma; infine si procede con l'annotazione dei rapporti di dipendenza. Il risultato è un file in formato CoNLL-U[4], dove ogni riga contiene un token e le righe vuote indicano l'inizio di una nuova frase; ad ogni token sono associati i seguenti campi:

[1] *Italian Natural Language Processing Lab, voce Profiling-UD*. URL: `http://www.italianlp.it/demo/profiling-ud/`.

[2] *Italian Natural Language Processing Lab*. URL: `http://www.italianlp.it/`.

[3] *Universal Dependencies*. URL: `https://universaldependencies.org/`.

[4] *Universal Dependencies, voce CoNLL-U Format*. URL: `https://universaldependencies.org/format.html`.

- ID: Indice del token nella frase.
- FORM: forma del token.
- LEMMA: lemma associato al token.
- UPOS: parte del discorso universale.
- XPOS: parte del discorso specifica della lingua.
- FEATS: lista delle caratteristiche morfologiche.
- HEAD: testa (a livello di dipendenza) della parola.
- DEPREL: relazione della parola con la testa.
- DEPS: grafo delle dipendenze.
- MISC: note aggiuntive.

```
# sent_id = 3
# text = Lo stesso ragazzo che si diceva avesse distrutto il Grande Signore Oscuro quando era ancora piccolo.
1     Lo    il    DET   RD    Definite=Def|Gender=Masc|Number=Sing|PronType=Art     3     det   _     _
2     stesso      stesso      ADJ   A     Gender=Masc|Number=Sing 3     amod  _     _
3     ragazzo     ragazzo     NOUN  S     Gender=Masc|Number=Sing 8     nsubj _     _
4     che   che   PRON  PR    PronType=Rel      6     obj   _     _
5     si    si    PRON  PC    Clitic=Yes|Person=3|PronType=Prs    6     expl:impers _     _
6     diceva      dire  VERB  V     Mood=Ind|Number=Sing|Person=3|Tense=Imp|VerbForm=Fin  3     acl:relcl   _     _
7     avesse      avere AUX   VA    Mood=Sub|Number=Sing|Person=3|Tense=Imp|VerbForm=Fin  8     aux   _     _
8     distrutto   distruggere VERB  V     Gender=Masc|Number=Sing|Tense=Past|VerbForm=Part      0     root  _     _
9     il    il    DET   RD    Definite=Def|Gender=Masc|Number=Sing|PronType=Art     10    det   _     _
10    Grande      grande      PROPN SP    _     8     obj   _     _
11    Signore     signore     PROPN SP    _     10    flat:name   _     _
12    Oscuro      oscuro      PROPN SP    _     10    flat:name   _     _
13    quando      quando      SCONJ CS    _     16    mark  _     _
14    era   essere      AUX   V     Mood=Ind|Number=Sing|Person=3|Tense=Imp|VerbForm=Fin  16    cop   _     _
15    ancora      ancora      ADV   B     _     16    advmod      _     _
16    piccolo     piccolo     ADJ   A     Gender=Masc|Number=Sing 8     advcl _     SpaceAfter=No
17    .     .     PUNCT FS    _     8     punct _     SpacesAfter=\n\n\n\n
```

Figura 3.1: Esempio di frase annotata con il formato CoNLL-U.

La seconda fase invece riguarda il profiling: a partire dal risultato dell'annotazione il tool estrae per ogni testo più di 140 feature, illustrate nella sottosezione 3.1.1. L'output sarà un file formato csv dove ogni riga è il vettore di feature linguistiche di un testo.

3.1.1 Caratteristiche linguistiche

In questa sottosezione, seguendo quanto descritto da Brunato et al. (2020), saranno presentati i diversi gruppi di feature estratti da Profiling-UD.

Proprietà testuali

Proprietà quantitative di base del testo come: lunghezza totale del documento, lunghezza media in token delle frasi, lunghezza media in caratteri delle parole.

Varietà lessicale

La misura utilizzata per valutare la varietà lessicale di un documento è il *Type/Token Ratio* (TTR), ovvero il rapporto tra la cardinalità del vocabolario del testo e il numero totale di token. Il TTR viene calcolato per le forme e per i lemmi, basandosi una volta sui primi 100 token e un'altra sui primi 200 token, dato che considerando l'intero testo i documenti più grandi sarebbero stati sfavoriti.

Informazioni morfosintattiche

Come la distribuzione per ognuna delle 17 categorie del discorso, e la densità lessicale, ovvero il rapporto tra il numero di parole piene (sostantivi, verbi, aggettivi e avverbi) e il numero totale di token del testo.

Struttura dei predicati verbali

Feature che rappresentano informazioni riguardanti i predicati verbali del documento. Ad esempio viene calcolata la distribuzione delle teste verbali, ovvero la media di teste verbali per frase; o la distribuzione delle radici verbali. Un'altra feature riguarda la Verb Arity: numero medio di collegamenti di dipendenza per testa verbale.

Struttura degli alberi globali e locali

Questo gruppo riguarda le caratteristiche degli alberi sintattici delle frasi del testo. Un esempio di feature è la profondità media degli alberi sintattici, ovvero la media delle pro-

fondità dell'albero sintattico di ogni frase del documento. Altra caratteristica calcolata è la lunghezza media dei legami di dipendenza del testo, dove la lunghezza di un legame di dipendenza è il numero di parole interposte tra la testa e il dipendente.

Relazioni sintattiche

La distribuzione delle relazioni di dipendenza, ovvero la distribuzione tra i 37 tipi di relazione sintattica dello schema di annotazione della Universal Dependecies.

Fenomeni di subordinazione

Quest'ultimo gruppo riguarda le caratteristiche delle proposizioni. Un esempio di feature è la distribuzione percentuale delle subordinate e delle principali nel testo. Altro esempio di caratteristica calcolata è l'ordine delle subordinate rispetto alla testa verbale, ovvero le percentuali delle subordinate in posizione pre-verbale e post-verbale.

3.1.2 Caratteristiche lessicali

Oltre alle caratteristiche linguistiche, i testi sono stati analizzati per ricavare delle feature di natura lessicale, attraverso l'estrazione degli n-grammi. Un n-gramma è una sequenza di n elementi contigui del testo; quindi se per esempio volessimo estrarre i bi-grammi di token dalla frase "Il cane insegue il gatto.", il risultato sarebbe: (Il, cane), (cane, insegue), (insegue, il), (il, gatto), (gatto, .).
Ogni n-gramma può essere composto da elementi testuali di diversa granularità. Per ogni testo sono stati estratti sul livello token gli unigrammi, bigrammi e trigrammi per forme e lemmi, e fino ai 6-grammi per le parti del discorso. Inoltre si è anche scesi al livello del singolo carattere, estraendo fino ai 4-grammi di carattere, esclusivamente se di inizio o di fine parola, quindi i prefissi e i suffissi lunghi fino a 4 caratteri.
Per l'estrazione è stato utilizzato lo script Python *ngram_classes.py*, che definisce le classi che hanno gestito il processo, è stato scelto un approccio di programmazione a oggetti, di seguito sono riportate le classi definite:

- Token, che rappresenta un token estratto dal testo; contiene nei suoi attributi la forma, il lemma e la pos (parte del discorso). Per ottimizzare la memoria la classe viene costruita attraverso la funzione *namedtuple* del modulo built-in collections[5] che permette di definire una sottoclasse della tupla.

- Sentence, che rappresenta una frase; ogni istanza Sentence contiene nel suo attributo ”tokenslist” la lista dei token di una frase; ha un metodo *get_wordlist* che restituisce la lista delle parole.

- Document, che rappresenta il singolo testo; nell'attributo ”sentenceslist” contiene la lista di frasi del documento, strutturata come lista di oggetti Sentence; sono anche stati definiti i metodi *get_word_ngrams*, *get_char_ngrams* e *get_char_pref_suff_ngrams*, che estraggono rispettivamente gli n-grammi di parole, caratteri e affissi (gruppi di caratteri a inizio o fine parola), restituendo un dizionario di feature.

- Corpus, che rappresenta l'intero corpus; nell'attributo ”docslist” contiene una lista di oggetti Document, mentre nell'attributo ”resultslist” contiene la lista delle etichette di classificazione (0 per insuccesso, 1 per successo), associate a ogni testo; ha un metodo *get_feature_dicts* che restituisce la lista di dizionari di feature dei documenti del corpus.

Per estrarre gli n-grammi dai testi innanzi tutto sono stati analizzati i file CoNLL-U ottenuti dalla fase di annotazione linguistica. Ogni riga del file che inizia con un numero contiene un token e la sua annotazione, quindi dividendola possiamo ricavare sia forma, lemma e parte del discorso del token. Le righe vuote invece indicano il passaggio da una frase all'altra.
Il parsing dei file CoNLL-U è svolto all'interno del costruttore della classe Document, che suddivide in frasi e in token, inizializzandoli come oggetti delle rispettive classi.

[5] *Documentazione Python, voce collections*. URL: `https://docs.python.org/3/library/collections.html#module-collections`.

```
    def __init__(self, lineslist: list) -> None:
        sentenceslist = []  # conterrà la lista delle frasi
        tmp_frase = []  # lista d'appoggio
        # iteriamo sulle righe del file CoNLL-u
        for line in lineslist:
            if line[0].isdigit():  # se la riga inizia con un numero
    significa che contiene una parola
                splitted_line = line.strip().split('\t')
                if '-' not in splitted_line[0]:  # se l'id della
    parola non contiene un trattino la memorizziamo
                    tmp_frase.append(Token(forma=splitted_line[1],
                                           lemma=splitted_line[2],
                                           pos=splitted_line[3]))
            if line == '\n':  # se la riga è vuota significa che la
    frase è finita
                sentenceslist.append(Sentence(tmp_frase))
                tmp_frase = []
        self.sentenceslist = sentenceslist
```

Code 3.1: Costruttore della classe Document.

Una volta suddivisi i testi in frasi e token, l'estrazione degli n-grammi viene effettuata dai metodi della classe Document *get_word_ngrams* per il lessico (forma, lemma, pos), e *get_char_pref_suff_ngrams* per i caratteri a inizio o a fine parola.

```
    def get_word_ngrams(self, t_type: str, n_ngram: int) -> dict:
        wordcount = 0
        featuredict = {}
        if t_type == 'forma':
            t_label = 'f'
        if t_type == 'lemma':
            t_label = 'l'
        if t_type == 'pos':
            t_label = 'p'
        for sentence in self.sentenceslist:
            word_list = sentence.get_wordlist(t_type)
```

```
12              wordcount += len(word_list)
13
14              for i in range(0, len(word_list) - n_ngram + 1):
15                  ngramma = word_list[i: i + n_ngram] # usiamo il list
    slicing per selezionare l'ngramma
16                  nomefeature = f'{t_label.upper()}_{n_ngram}_' + '_'.
    join(ngramma)
17                  featuredict[nomefeature] = featuredict.get(
    nomefeature, 0) + 1 # se la feature è presente nel dizionario
    allora il conteggio aumenta di 1, se non è presente allora viene
     aggiunta
18
19          return {featurename: float(value)/float(wordcount) for
    featurename, value in featuredict.items()}  # normalizzo
    dividendo per il numero di token totale del documento e
    restituisco il dizionario di feature
```

Code 3.2: Metodo get_word_ngrams della classe Document

Attraverso i metodi di queste classi è stato possibile estrarre gli n-grammi desiderati per ogni corpus; questi dati saranno poi utilizzati come vettori di feature durante la fase di classificazione.

3.2 Analisi statistiche

Il processo di profiling linguistico ha permesso di estrarre in totale un insieme di 144 feature per ogni testo; l'obiettivo di questa fase è studiare la rilevanza statistica di ognuna di esse, ovvero la tendenza di quella feature ad assumere valori più alti in base al campione di origine dell'osservazione, quindi in base all'appartenenza alla classe insuccesso o successo.

Per lo scopo è stato utilizzato il test *Mann-Whitney U*[6]: test non parametrico d'ipotesi, che verifica se due campioni statistici provengono dalla stessa popolazione.

[6] *Wikipedia, voce Mann-Whitney U test*. URL: https://en.wikipedia.org/wiki/Mann-Whitney_U_test.

Il test, dati due campioni indipendenti X e Y, verifica la validità dell'ipotesi nulla H_0: la probabilità che un valore estratto dal campione X sia più grande di un valore estratto dal campione Y è uguale alla probabilità che un valore estratto dal campione Y sia più grande di un valore estratto dal campione X.

$$H_0 : P(X_i > Y_i) = P(Y_i > X_i) \tag{3.1}$$

Se l'ipotesi nulla non dovesse essere verificata, allora sarà accettata l'ipotesi alternativa H_1: le distribuzioni dei valori estratti dai due campioni non sono uguali.

$$H_1 : P(X_i > Y_i) \neq P(Y_i > X_i) \tag{3.2}$$

Il primo passo per calcolare il test è raggruppare tutte le osservazioni dei due gruppi e ordinarle in modo crescente, assegnando a ognuna un rango, ovvero la propria posizione nell'ordinamento.
Successivamente viene calcolata la statistica U per entrambi i campioni attraverso la formula:

$$U = R - \frac{n(n+1)}{2} \tag{3.3}$$

Dove R è la somma dei ranghi del campione e n è la dimensione del campione.
Per inferire la significatività di una feature si utilizza il *p-value*[7] del test, ovvero la probabilità di ottenere un valore uguale o più estremo per U, rispetto a quello osservato in H_0.
Il p-value viene poi confrontato con un valore α determinato a priori, che rappresenta la soglia di significatività statistica: se il p-value è superiore ad α allora non c'è una differenza statisticamente significativa tra le due distribuzioni, se è inferiore invece esiste una differenza significativa.
Il Mann-Whitney U test è stato calcolato per ogni feature estratta dai corpora attraverso lo script *analisi_statistiche.py*, richiamando la funzione *mannwhitneyu* del modulo SciPy[8] che, prese in input le liste delle osservazioni dei due campioni (X: documenti della classe

[7] *Wikipedia, voce p-value*. URL: `https://en.wikipedia.org/wiki/P-value`.
[8] *Documentazione Scipy*. URL: `https://docs.scipy.org/doc/scipy/index.html#`.

insuccesso, Y: documenti della classe successo), restituisce il p-value e la U del campione X per la feature analizzata.
Come valore soglia α è stato scelto $0,05$, quindi tutte le feature che hanno registrato un p-value inferiore ad esso sono state considerate statisticamente significative.
Nella seguente tabella per ogni corpus sono mostrate in verde le feature risultate come statisticamente significative, e in rosso quelle non significative. Sono state omesse le feature non significative in tutti e tre i corpora.

Proprietà testuali e varietà lessicale

Nome feature	**p-value HPTOT**	**p-value HP2011**	**p-value LOTR**
n_sentences			
n_tokens			
tokens_per_sent			
char_per_tok			
ttr_lemma_chunks_100			
ttr_lemma_chunks_200			
ttr_form_chunks_100			
ttr_form_chunks_200			

Tabella 3.1: Significatività statistica proprietà testuali e varietà lessicale.

Tra le feature riguardanti le proprietà generali del testo e la varietà lessicale, 3 si sono rivelate significative per tutti i corpora, mentre 5 solo per i corpora di testi a tema Harry Potter.

Informazioni morfosintattiche: distribuzione delle parti del discorso

Nome feature	**p-value HPTOT**	**p-value HP2011**	**p-value LOTR**
upos_dist_ADJ			
upos_dist_ADP			

upos_dist_ADV			
upos_dist_AUX			
upos_dist_CCONJ			
upos_dist_DET			
upos_dist_INTJ			
upos_dist_NOUN			
upos_dist_NUM			
upos_dist_PART			
upos_dist_PRON			
upos_dist_PROPN			
upos_dist_PUNCT			
upos_dist_SCONJ			
upos_dist_SYM			
upos_dist_VERB			
upos_dist_X			
lexical_density			

Tabella 3.2: Significatività statistica distribuzioni delle parti del discorso.

Tra le caratteristiche relative alla distribuzione delle parti del discorso invece si trovano 7 feature significative in tutti i corpora: distribuzione delle adposizioni, delle congiunzioni coordinanti, dei determinanti, delle interiezioni, dei nomi propri, della punteggiatura e dei simboli.

Informazioni morfosintattiche: morfologia inflessionale

Nome feature	**p-value HPTOT**	**p-value HP2011**	**p-value LOTR**

verbs_tense_dist_Fut			
verbs_tense_dist_Imp			
verbs_tense_dist_Past			
verbs_tense_dist_Pres			
verbs_mood_dist_Cnd			
verbs_mood_dist_Imp			
verbs_mood_dist_Ind			
verbs_mood_dist_Sub			
verbs_form_dist_Ger			
verbs_form_dist_Part			
verbs_num_pers_dist_Plur+			
verbs_num_pers_dist_Plur+1			
verbs_num_pers_dist_Plur+2			
verbs_num_pers_dist_Plur+3			
verbs_num_pers_dist_Sing+1			
verbs_num_pers_dist_Sing+2			
verbs_num_pers_dist_Sing+3			
aux_tense_dist_Fut			
aux_tense_dist_Imp			
aux_tense_dist_Past			
aux_tense_dist_Pres			
aux_mood_dist_Cnd			
aux_mood_dist_Imp			
aux_mood_dist_Ind			
aux_mood_dist_Sub			
aux_form_dist_Fin			
aux_form_dist_Ger			
aux_num_pers_dist_Plur+1			

aux_num_pers_dist_Plur+2			
aux_num_pers_dist_Plur+3			
aux_num_pers_dist_Sing+1			
aux_num_pers_dist_Sing+2			
aux_num_pers_dist_Sing+3			

Tabella 3.3: Significatività statistica morfologia inflessionale.

Per quanto riguarda la morfologia inflessionale invece sono 12 le caratteristiche statisticamente significative per tutti i corpora: distribuzione dei verbi al tempo imperfetto, dei verbi al tempo presente, dei verbi al modo indicativo, dei verbi al modo imperativo, dei verbi coniugati alla seconda persona plurale, dei verbi coniugati alla terza persona plurale, dei verbi coniugati alla seconda persona singolare, degli ausiliari al tempo imperfetto, degli ausiliari al tempo presente, degli ausiliari coniugati alla seconda persona plurale, degli ausiliari coniugati alla terza persona plurale, e degli ausiliari coniugati alla seconda persona singolare.

Struttura dei predicati verbali

Nome feature	**p-value HPTOT**	**p-value HP2011**	**p-value LOTR**
verbal_head_per_sent			
verbal_root_perc			
avg_verb_edges			
verb_edges_dist_0			
verb_edges_dist_1			
verb_edges_dist_2			
verb_edges_dist_3			
verb_edges_dist_4			
verb_edges_dist_5			
verb_edges_dist_6			

Tabella 3.4: Significatività statistica struttura dei predicati verbali

Tra le feature riguardanti la struttura dei predicati verbali, 3 caratteristiche si sono dimostrate statisticamente significative in tutti i corpora: la distribuzione media delle teste verbali, distribuzione media delle radici con a capo un verbo sul totale delle radici e la distribuzione dei verbi per la classe di verb arity 0.

Struttura degli alberi sintattici globali e locali

Nome feature	**p-value HPTOT**	**p-value HP2011**	**p-value LOTR**
avg_max_depth			
avg_token_per_clause			
avg_max_links_len			
avg_links_len			
max_links_len			
avg_prepositional_chain_len			
n_prepositional_chains			
prep_dist_3			
prep_dist_4			
prep_dist_5			
obj_pre			
obj_post			
subj_pre			
subj_post			

Tabella 3.5: Significatività statistica struttura degli alberi sintattici globali e locali.

Per quanto riguarda la struttura degli alberi sintattici, 5 feature si sono rivelate significative in tutti i corpora: la profondità media degli alberi sintattici, il numero medio di token per proposizione, la media della più lunga catena di dipendenza di ogni frase, il numero medio

di parole interposte tra la testa sintattica e i suoi dipendenti e il numero totale di catene preposizionali.

Relazioni sintattiche

Nome feature	p-value HPTOT	p-value HP2011	p-value LOTR
dep_dist_acl:relcl			
dep_dist_advcl			
dep_dist_advmod			
dep_dist_amod			
dep_dist_appos			
dep_dist_aux			
dep_dist_aux:pass			
dep_dist_case			
dep_dist_cc			
dep_dist_ccomp			
dep_dist_compound			
dep_dist_conj			
dep_dist_cop			
dep_dist_csubj			
dep_dist_det			
dep_dist_det:poss			
dep_dist_discourse			
dep_dist_dislocated			
dep_dist_expl			
dep_dist_expl:impers			
dep_dist_expl:pass			
dep_dist_fixed			
dep_dist_flat			

dep_dist_flat:foreign			
dep_dist_flat:name			
dep_dist_iobj			
dep_dist_nmod			
dep_dist_nsubj			
dep_dist_nsubj:pass			
dep_dist_nummod			
dep_dist_obj			
dep_dist_obl			
dep_dist_obl:agent			
dep_dist_orphan			
dep_dist_parataxis			
dep_dist_punct			
dep_dist_root			
dep_dist_vocative			
dep_dist_xcomp			

Tabella 3.6: Significatività statistica relazioni sintattiche.

Tra le caratteristiche riguardanti le relazioni sintattiche, 13 feature si sono dimostrate statisticamente significative in tutti e tre i corpora.

Fenomeni di subordinazione

Nome feature	**p-value HPTOT**	**p-value HP2011**	**p-value LOTR**
principal_proposition_dist			
subordinate_proposition_dist			
avg_subordinate_chain_len			
subordinate_dist_1			
subordinate_dist_4			

subordinate_dist_5			
verbs_num_pers_dist_+3			

Tabella 3.7: Significatività statistica fenomeni di subordinazione.

Per i fenomeni di subordinazione, solo 2 caratteristiche si sono dimostrate significative per tutti e tre i casi: la distribuzione delle proposizioni principali e la distribuzione delle proposizioni subordinate.

In totale, per il corpus HPTOT 124 feature sono significative, mentre per HP2011 sono 85, e solo 64 per LOTR. Inoltre ci sono 45 feature che si dimostrano significative in tutti e tre i casi.

A partire dalla statistica U, per ogni feature considerata statisticamente significativa, è stato calcolato il *rank-biserial*: misura di correlazione dei ranghi che permette di riportare quantitativamente la dimensione e la direzione della tendenza statistica che hanno i valori di quella feature.

Il rank-biserial è stato calcolato dallo script *analisi_statistiche.py* attraverso la formula:

$$R_b = 1 - \frac{2U}{L_X L_Y} \tag{3.4}$$

Dove L_X e L_Y sono rispettivamente la dimensione del campione X (insuccesso) e la dimensione del campione Y (successo).

Il risultato della formula è un numero compreso tra -1 e 1; i valori interi possono essere interpretati come:

- $R_b = 1$: completa dominanza del campione Y, quindi tutte le osservazioni della feature del campione Y (successo) sono più grandi di tutte le osservazioni del campione X (insuccesso).
- $R_b = -1$: completa dominanza del campione X, quindi tutte le osservazioni della feature del campione X (insuccesso) sono più grandi di tutte le osservazioni del campione Y (successo).
- $R_b = 0$: nessuna differenza tra le osservazioni dei due campioni.

Raggruppando e ordinando i valori del rank-biserial di ogni feature è stata costruita per ogni corpus una tabella per mostrare i risultati ottenuti.

HPTOT			
#	Nome feature	Dire..	Rb
1	n_sentences	succ	0.3262
2	n_tokens	succ	0.2575
3	upos_dist_PUNCT	succ	0.2369
4	n_prepositional_chains	succ	0.2301
5	avg_max_depth	ins	0.2228
6	dep_dist_punct	succ	0.2227
7	avg_max_links_len	ins	0.2132
8	tokens_per_sent	ins	0.2127
9	dep_dist_root	succ	0.2127
10	verbal_head_per_sent	ins	0.2123
11	verbal_root_perc	ins	0.2035
12	dep_dist_conj	ins	0.1791
13	upos_dist_INTJ	succ	0.1766
14	dep_dist_discourse	succ	0.1638
15	aux_num_pers_dist_Sing..	succ	0.1490
16	ttr_lemma_chunks_200	succ	0.1417
17	ttr_form_chunks_200	succ	0.1396
18	aux_tense_dist_Pres	succ	0.1374
19	dep_dist_obl	ins	0.1361
20	verbs_mood_dist_Imp	succ	0.1328
21	verbs_num_pers_dist_Pl..	succ	0.1315
22	upos_dist_DET	ins	0.1300
23	subordinate_proposition..	ins	0.1266
24	principal_proposition_dist	succ	0.1266
25	verbs_tense_dist_Pres	succ	0.1265
26	verbs_tense_dist_Imp	ins	0.1245
27	aux_tense_dist_Imp	ins	0.1243
28	verbs_mood_dist_Cnd	succ	0.1231
29	verbs_num_pers_dist_Si..	succ	0.1206
30	dep_dist_cc	ins	0.1179

HP2011			
#	Nome feature	Dire..	Rb
1	n_sentences	succ	0.2499
2	upos_dist_PUNCT	succ	0.2151
3	dep_dist_punct	succ	0.2046
4	n_tokens	succ	0.1878
5	n_prepositional_chains	succ	0.1861
6	tokens_per_sent	ins	0.1764
7	dep_dist_root	succ	0.1764
8	verbal_head_per_sent	ins	0.1760
9	avg_max_links_len	ins	0.1755
10	avg_max_depth	ins	0.1721
11	dep_dist_conj	ins	0.1677
12	ttr_lemma_chunks_200	succ	0.1628
13	dep_dist_cc	ins	0.1496
14	dep_dist_discourse	succ	0.1491
15	upos_dist_CCONJ	ins	0.1480
16	ttr_form_chunks_200	succ	0.1434
17	upos_dist_INTJ	succ	0.1417
18	verbal_root_perc	ins	0.1256
19	principal_proposition_dist	succ	0.1217
20	subordinate_proposition..	ins	0.1216
21	ttr_lemma_chunks_100	succ	0.1209
22	avg_links_len	ins	0.1154
23	verbs_num_pers_dist_Pl..	succ	0.1120
24	verbs_form_dist_Ger	succ	0.1100
25	char_per_tok	succ	0.1082
26	verbs_mood_dist_Imp	succ	0.1080
27	verb_edges_dist_1	succ	0.1075
28	aux_num_pers_dist_Sing..	succ	0.1072
29	dep_dist_obl	ins	0.1065
30	aux_tense_dist_Pres	succ	0.1031

LOTR			
#	Nome feature	Dire..	Rb
1	verbal_root_perc	ins	0.2862
2	verbal_head_per_sent	ins	0.2510
3	n_sentences	succ	0.2222
4	avg_max_depth	ins	0.2171
5	upos_dist_PUNCT	succ	0.2146
6	dep_dist_punct	succ	0.2067
7	tokens_per_sent	ins	0.2036
8	dep_dist_root	succ	0.2036
9	upos_dist_INTJ	succ	0.1924
10	avg_max_links_len	ins	0.1915
11	verbs_mood_dist_Imp	succ	0.1898
12	dep_dist_conj	ins	0.1780
13	dep_dist_discourse	succ	0.1763
14	verbs_num_pers_dist_Sin..	succ	0.1617
15	obj_pre	succ	0.1589
16	obj_post	ins	0.1589
17	n_tokens	succ	0.1575
18	n_prepositional_chains	succ	0.1535
19	dep_dist_compound	succ	0.1507
20	dep_dist_cc	ins	0.1480
21	upos_dist_CCONJ	ins	0.1460
22	avg_token_per_clause	succ	0.1459
23	verbs_tense_dist_Pres	succ	0.1455
24	dep_dist_obl	ins	0.1424
25	dep_dist_advcl	ins	0.1413
26	upos_dist_VERB	ins	0.1399
27	subordinate_proposition_..	ins	0.1345
28	principal_proposition_dist	succ	0.1345
29	aux_tense_dist_Pres	succ	0.1317
30	dep_dist_appos	succ	0.1265

Tabella 3.8: Prime 30 feature per rank-biserial assoluto con associata la direzione.

Analizzando i risultati mostrati nella tabella 3.8 si nota che, i testi di successo, per tutti e tre i corpora, sono statisticamente caratterizzati da:

- un numero maggiore di token e di frasi totali (n_tokens, n_sentences),
- un maggiore utilizzo di segni di punteggiatura (upos_dist_PUNCT, dep_dist_PUNCT),
- frasi più brevi (tokens_per_sent) e più semplici (avg_max_depth, avg_max_links_len),
- frequenza maggiore di interiezioni (upos_dist_INTJ),
- maggiore utilizzo di verbi all'imperativo (verbs_mood_dist_Imp).

Altre caratteristiche del successo sono rilevanti solo per i testi basati su Harry Potter (quindi i corpora HPTOT e HP2011): maggiore varietà lessicale (TTR_lemma_chunks_200, TTR_form_chunks_200) e un maggiore utilizzo dei verbi coniugati alla seconda persona plurale (verbs_num_pers_dist_Plur+2).

4. Esperimenti di classificazione

Una volta ottenute le caratteristiche linguistiche e lessicali dei documenti attraverso la fase di monitoraggio linguistico, i dati estratti sono stati utilizzati per svolgere degli esperimenti di classificazione automatica dei testi, allo scopo di valutare l'efficacia della predizione del successo di una storia attraverso le feature estratte.
La classificazione di testi è una delle applicazioni dell'apprendimento automatico supervisionato e ha l'obiettivo di predire l'appartenenza di un testo a una categoria, detta classe. Il processo inizia con la raccolta di un corpus di testi, ognuno rappresentato da un vettore di caratteristiche; inoltre è necessario etichettare ogni testo con la propria classe di appartenenza; quindi ogni testo del corpus avrà un proprio vettore di caratteristiche associato alla propria classe, in questo modo si costruisce il *training set*. Un algoritmo di apprendimento automatico, detto classificatore, sarà addestrato attraverso il training set e assegnerà a ogni caratteristica un peso nell'indicare l'appartenenza a una delle classi predefinite. Una volta addestrato, l'algoritmo è pronto per effettuare delle "predizioni", ovvero prendere in input un nuovo vettore di feature e restituire in output la previsione della classe di appartenenza. In questo capitolo saranno esposti innanzi tutto gli strumenti e i criteri di valutazione che sono stati utilizzati per effettuare gli esperimenti di classificazione; poi si tratteranno i diversi scenari e i modelli sui quali sono stati applicati i suddetti esperimenti.

4.1 Algoritmi di Machine Learning

L'algoritmo scelto per la classificazione si basa sulla tecnica di apprendimento automatico detta *Support Vector Machines*[1]. Questo metodologia consiste nel rappresentare i vettori del training set come dei punti in uno spazio formato da un numero di dimensioni pari al numero di caratteristiche per ogni vettore, successivamente verrà tracciato un iper-piano che separi i punti appartenenti alle classi, massimizzandone la distanza. Nel caso di una

[1] *Documentazione scikit-learn, voce Support Vector Machine*. URL: `https://scikit-learn.org/stable/modules/svm.html`.

classificazione binaria quindi l'iper-piano dividerà lo spazio in due zone. Quando si effettuerà la predizione di un nuovo vettore, sarà anch'esso proiettato in questo spazio, e la classe assegnata sarà determinata dalla zona dove giace.
Il classificatore utilizzato è implementato dalla classe LinearSVC[2] del modulo scikit-learn, basata sulla libreria LIBLINEAR[3] sviluppata dalla National Taiwan University. Una volta assegnato a una variabile, il classificatore può essere addestrato con il metodo:

$$classificatore.fit(X, y) \tag{4.1}$$

Dove X è la lista dei vettori di feature del training set e y è la lista delle etichette di classi associate.
Il classificatore dopo essere stato addestrato è pronto per effettuare le predizioni, attraverso il metodo:

$$classificatore.predict(X) \rightarrow y_{pred} \tag{4.2}$$

Dove X è la lista dei vettori di feature che sono oggetto della predizione e y_{pred} è il valore restituito dal metodo, ovvero la lista delle etichette di classe risultanti dal processo di classificazione.

4.1.1 Matrice densa e matrice sparsa

L'algoritmo di classificazione richiede in input la lista dei vettori di feature, ogni vettore deve avere lunghezza e ordinamento fisso: ogni posizione del vettore è associata a una singola feature. Data questa restrizione, la lista dei vettori è rappresentabile come una matrice di dimensione $m * n$, dove m è il numero di vettori (uno per ogni documento del corpus) e n è la cardinalità dell'insieme delle feature.
Per quanto riguarda i training set di caratteristiche linguistiche, il tool Profiling-UD, dato un insieme di testi, in automatico restituisce una lista di vettori di feature di lunghezza fissa.

[2] *Documentazione scikit-learn, voce LinearSVC*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html`.

[3] *LIBLINEAR*. URL: `https://www.csie.ntu.edu.tw/~cjlin/liblinear/`.

La situazione è diversa per i dataset di caratteristiche lessicali: ogni feature di un vettore è la frequenza di un n-gramma nel documento associato. Come detto prima i vettori devono avere ordinamento e lunghezza fissa, quindi ogni vettore in questo caso dovrà essere lungo tanto quanto la cardinalità dell'insieme di tutti gli n-grammi estratti nel corpus. Questo significa che, date due ipotetiche liste di uni-grammi:

$$\begin{aligned} ng_1 : [\text{"}il\text{"},\text{"}cane\text{"},\text{"}il\text{"}], \\ ng_2 : [\text{"}il\text{"},\text{"}gatto\text{"}] \end{aligned} \tag{4.3}$$

La matrice di vettori di feature risultante dovrebbe essere:

-	il	cane	gatto
$vettore_1$	2	1	0
$vettore_2$	1	0	1

Tabella 4.1: Esempio di matrice di feature lessicali.

Considerate le dimensioni dei corpora e la varietà lessicale che possono contenere, le matrici costruite a partire dai testi hanno dimensioni notevoli; prendendo come esempio il caso peggiore, ovvero il training set formato da tutti i vari tipi di n-grammi estratti dal corpus HPTOT, le dimensioni della matrice sarebbero $26032x6510668$. Le suddette matrici hanno un'altra peculiarità: sono matrici sparse, la maggior parte delle loro celle ha valore 0. In questi casi è conveniente, dal punto di vista del consumo in memoria, comprimere le matrici sparse in matrici dense, ovvero memorizzare esclusivamente i valori non nulli, riducendo drasticamente le dimensioni della matrice.

Gli n-grammi estratti sono stati inizialmente memorizzati attraverso dei dizionari, struttura dati inutilizzabile direttamente per l'addestramento del classificatore. Sono stati allora trasformati grazie all'oggetto *DictVectorizer*[4], implementato dalla libreria scikit-learn, che, attraverso il metodo *fit_transform*, data in input una lista di dizionari di feature restituisce

[4] *Documentazione scikit-learn, voce DictVectorizer*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.DictVectorizer.html`.

una matrice sparsa compressa, che può essere utilizzata come training set, ottimizzando il consumo della memoria.

4.1.2 Normalizzazione dei dati

Per rendere più efficaci gli esperimenti di classificazione sono stati svolti diversi processi di normalizzazione dei dati.

Per quanto riguarda i training set composti da caratteristiche lessicali, innanzi tutto è stato necessario normalizzare la frequenza assoluta degli n-grammi, dividendola per il numero totale di n-grammi dello stesso tipo del documento, ottenendo la frequenza relativa. Questo passaggio è utile in quanto i corpora utilizzati contengono documenti di lunghezza molto diversa, quindi utilizzando la frequenza assoluta i valori delle feature sarebbero stati tendenzialmente più alti per i documenti più lunghi.

Per tutti i training set, sia a base linguistica che lessicale, è stata applicata una normalizzazione dell'intervallo dei valori delle feature. Alcune caratteristiche infatti registrano valori naturalmente maggiori rispetto ad altre, ad esempio è evidente come, tra le caratteristiche linguistiche, la feature "token totali" abbia un valore mediamente maggiore rispetto alla feature "frasi totali". Quindi, come spiegato in (Hsu et al. 2003), è fondamentale mappare linearmente tutti i valori delle caratteristiche in un intervallo $[0,1]$ o $[-1,1]$, in modo da evitare che le feature con valori mediamente più alti siano considerate più importanti durante la classificazione.

I training set di caratteristiche linguistiche sono stati normalizzati attraverso l'oggetto *MinMaxScaler*[5] della libreria scikit-learn, che permette, con il metodo *fit_transform*, di normalizzare le feature dei vettori in un intervallo $[0,1]$.

$$MinMaxScaler.fit\_trasform(X_{raw}) \rightarrow X_{norm} \tag{4.4}$$

Dove X_{raw} è la lista dei vettori di feature non normalizzati, mentre X_{norm} è la lista normalizzata.

[5] *Documentazione scikit-learn, voce MinMaxScaler*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html`.

Per quanto riguarda le caratteristiche lessicali, il *MinMaxScaler* è inutilizzabile. Infatti in questo caso la struttura dati alla base è una matrice sparsa compressa; come si era accennato precedentemente la compressione avviene memorizzando per ogni vettore esclusivamente i valori diversi da 0. Utilizzando un algoritmo di normalizzazione che può mappare i valori a 0 la matrice perderebbe di significato. Quindi è stato utilizzato un diverso algoritmo di normalizzazione, implementato dall'oggetto *MaxAbsScaler*[6] della stessa libreria. Il *MaxAbsScaler* ci permette di mappare i valori delle feature in un intervallo $[-1,1]$ basandosi sul massimo valore assoluto di quella feature nella matrice. L'utilizzo pratico del *MaxAbsScaler* è del tutto simile al *MinMaxScaler*.

4.2 Criteri di valutazione e baseline

Per valutare le performance di un classificatore serve un insieme di dati annotati, chiamato *test set*, con la stessa struttura del training set, quindi una lista di vettori di feature con associate le etichette di classe. I dati del test set devono essere diversi da quelli usati per l'addestramento del classificatore: effettuare una predizione su un vettore già presente nel training set porterebbe a una classificazione sicuramente corretta, deviando l'intera valutazione. Per sfruttare al massimo i dati ricavati si può usare il metodo chiamato *n-fold crossvalidation*, che permette di utilizzare un unico dataset sia per l'addestramento che per la valutazione. Il dataset, infatti, viene diviso in n parti, a questo punto saranno svolti n esperimenti di valutazione, dove una parte farà da test set e le altre $n-1$ parti faranno da training set. La prestazione dell'intero sistema sarà la media degli n esperimenti svolti. La prestazione di un classificatore è quantificata attraverso una metrica di valutazione, la più diffusa per i sistemi che assegnano sempre un singolo output è l'*accuracy*, ovvero il numero di predizioni corrette diviso per la cardinalità del test set.
I risultati del sistema vanno poi paragonati a una *baseline*: valore che viene assunto come limite inferiore di prestazioni, rispetto al quale valutiamo la significatività degli incrementi dati dal classificatore.

[6] *Documentazione scikit-learn, voce MaxAbsScaler.* URL: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MaxAbsScaler.html`.

Per valutare gli esperimenti di classificazione *in-domain* del progetto è stata utilizzata una 10-fold crossvalidation, implementata dalla funzione di scikit-learn *cross_val_score*, che prende in input il classificatore, la lista dei vettori di feature e la liste delle etichette associate; inoltre, attraverso il parametro opzionale *cv* possiamo scegliere il numero di parti della *crossvalidation*, che quindi è stato impostato a 10. Il risultato è un array contenente l'accuracy di ogni ciclo di valutazione svolto, calcolandone la media aritmetica è stata ottenuta l'accuracy del classificatore.
Sono state utilizzate due diverse baseline per ogni esperimento di classificazione, calcolate attraverso l'oggetto *DummyClassifier*[7] di scikit-learn, che implementa un classificatore che effettua predizioni utilizzando delle semplici strategie applicate alla lista delle etichette di classe, ignorando totalmente i vettori di feature. Le strategie scelte sono: *most_frequent*, dove il risultato della predizione è sempre la classe più frequente, che d'ora in poi sarà chiamata "baseline_mf"; e *uniform* che invece assegna casualmente la classe, chiamata "baseline_u". Il *DummyClassifier* può essere utilizzato in modo molto simile a un normale classificatore: dopo averne inizializzato l'istanza, specificando la strategia da utilizzare con il parametro *strategy*, può essere addestrato con il metodo *fit*; infine possiamo ottenere le prestazioni con il metodo *score*.

$$DummyClassifier.score(y) \rightarrow baseline \tag{4.5}$$

Dove y rappresenta la lista delle etichette di classe necessarie per la valutazione del sistema, e il valore restituito è l'accuracy del sistema, quindi la baseline che sarà poi utilizzata per valutare i classificatori reali.

4.3 Scenari di classificazione

Sono stati svolti degli esperimenti di classificazione su diversi scenari, con scenario si intende la scelta dei testi usati come training set e di quelli invece usati per la valutazione del sistema.

[7] *Documentazione scikit-learn, voce DummyClassifier.* URL: `https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html`.

I corpora di testi sono quelli presentati nella sezione 1.3, ovvero HPTOT, HP2011 e LOTR. Il primo scenario di test è quello *in-domain*, ovvero il classificatore è stato valutato su dei testi che fanno parte dello stesso dominio di quelli di addestramento, in questo caso è stato necessario utilizzare il metodo della *n-fold crossvalidation* presentato nella sezione 4.2.
Sono stati anche svolti degli esperimenti *out-domain*, nei quali il classificatore è stato valutato su dei racconti appartenenti a un dominio diverso rispetto al training set, utile ad analizzare la capacità di generalizzazione del classificatore su testi di natura differente rispetto a quelli di addestramento. Gli esperimenti out-domain sono stati realizzati utilizzando i corpora HPTOT e LOTR, è stato invece escluso HP2011 perché si tratta di un sottoinsieme di HPTOT, quindi fa parte dello stesso dominio.
Per il dominio dei testi ispirati a Harry Potter è stato osservato un ulteriore scenario: il classificatore è stato addestrato con il corpus HP2011, ovvero l'insieme dei testi scritti nel 2011, successivamente sono stati realizzati diversi cicli di valutazione, ognuno utilizzando come test set i testi realizzati in un preciso anno diverso dal 2011. Quindi i testi del 2011 sono stati utilizzati come base fissa di addestramento, mentre i testi degli anni 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019 e 2020 si sono alternati per la valutazione del sistema. In questo scenario, che d'ora in poi sarà chiamato *cross-time*, il dominio varia non in relazione al tema dei testi, ma all'anno di produzione, in modo da analizzare il mantenimento dell'efficacia del classificatore nello scorrere del tempo.

4.4 Modelli di classificazione

La classificazione si è basata su due diversi modelli: linguistico e lessicale. Per il modello linguistico sono state utilizzate le 144 feature estratte dal tool linguistico Profiling-UD e presentate nel paragrafo 2.1.1. Gli esperimenti di classificazione linguistica, a livello di programmazione, sono stati interamente gestiti attraverso una apposita classe, chiamata *ClassificazioneLinguistica*, definita nel file *class_ling_classes.py*.
Per quanto riguarda le feature lessicali, presentate nel paragrafo 2.1.2, le prove di classificazione sono state svolte su 5 sotto-modelli differenti:

- *forma*, che utilizza come feature uni-grammi, 2-grammi e 3-grammi di token;
- *lemma*, che utilizza come feature uni-grammi, 2-grammi e 3-grammi di lemmi;
- *pos*, che utilizza da uni-grammi fino ai 6-grammi di parti del discorso;
- *carattere*, che utilizza le sequenze di caratteri a inizio o fine parola lunghe 1, 2, 3 o 4 caratteri;
- *totale*, modello che comprende tutti gli altri precedentemente elencati.

Nella tabella di seguito (Tab. 4.2) è esposto il numero di feature per i modelli lessicali di ogni corpus.

-	**Forma**	**Lemma**	**Pos**	**Carattere**	**Totale**
HPTOT	3142528	2573479	733395	61266	6510668
HP2011	476607	426613	250869	26772	1180861
LOTR	166324	157745	124721	17412	466202

Tabella 4.2: Quantità di feature lessicali per i diversi corpora.

Infine è stato realizzato un ulteriore esperimento di classificazione, basato esclusivamente sugli uni-grammi di token. In questo caso l'obiettivo è stato estrarre i pesi assegnati dal classificatore alle singole parole, per poter costruire una classifica del lessico correlato al successo dei racconti.

5. Risultati e discussione

In questo capitolo si mostreranno e discuteranno i risultati degli esperimenti di classificazione presentati nel capitolo precedente. Inoltre, per ogni training set usato per la classificazione, sono stati estratti i pesi che il classificatore ha assegnato durante la fase di addestramento a ogni feature nel determinare una delle due classi; questa lista è memorizzata nell'attributo "coef_" dell'oggetto classificatore. Il peso attribuito a una feature, in un classificatore lineare e binario, può essere interpretato come l'importanza della feature nel processo di classificazione; attraverso il segno del peso si può anche comprendere verso quale classe è rilevante: se minore di zero allora è rilevante per la classe negativa (insuccesso), se maggiore di zero lo è per la classe positiva (successo).

5.1 In-domain

Nella seguente tabella sono riportati i risultati dei classificatori addestrati con le feature linguistiche, e valutati nello scenario in-domain.

-	**accuracy**	**baseline_mf**	**baseline_u**
HPTOT	65.03%	50.16%	50.81%
HP2011	62.69%	52.14%	50.89%
LOTR	62.24%	56.43%	53.11%

Tabella 5.1: Risultati classificazione linguistica in-domain.

I risultati presentati nella tabella 5.1 mostrano che, i classificatori basati sul modello linguistico e utilizzati per delle predizioni in-domain, riescano a raggiungere dei miglioramenti significativi rispetto alle baseline. L'accuracy più bassa è stata registrata dal training set HP2011, comunque mantenendo uno scarto maggiore di 10 punti percentuali rispetto a entrambe le baseline. Il classificatore basato sul corpus LOTR è invece quello che accumula lo scarto minore dalla baseline_mf, ovvero 6 punti percentuali, prestazione comunque soddisfacente considerando che è il corpus più piccolo e più sbilanciato tra i tre

(vedi tabelle 2.1 e 2.2). Invece i risultati migliori derivano dal classificatore basato sul corpus più grande, HPTOT, che registra uno scarto di 15 punti percentuali su entrambe le baseline.

HPTOT			
#	Nome feature	direzione	Coef
1	avg_token_per_clause	ins	4.706
2	n_tokens	succ	3.334
3	char_per_tok	succ	2.875
4	n_prepositional_chains	succ	2.444
5	avg_max_depth	ins	1.805
6	n_sentences	succ	1.805
7	max_links_len	succ	1.665
8	upos_dist_SYM	ins	1.594
9	upos_dist_VERB	ins	1.375
10	upos_dist_SCONJ	succ	1.300
11	dep_dist_amod	ins	1.299
12	dep_dist_aux:pass	ins	1.262
13	upos_dist_PART	ins	1.225
14	dep_dist_obj	succ	1.194
15	dep_dist_dislocated	succ	1.191
16	upos_dist_PUNCT	succ	1.066
17	avg_max_links_len	ins	1.065
18	subordinate_propositi..	ins	1.056
19	upos_dist_CCONJ	succ	1.015
20	dep_dist_case	succ	0.974
21	tokens_per_sent	ins	0.973
22	dep_dist_cc	ins	0.963
23	verbs_form_dist_Fin	ins	0.934
24	dep_dist_acl	succ	0.906
25	ttr_lemma_chunks_2..	succ	0.893
26	upos_dist_ADJ	succ	0.892
27	verbs_num_pers_dist..	ins	0.868
28	upos_dist_NUM	ins	0.861
29	dep_dist_root	ins	0.825
30	dep_dist_conj	ins	0.824

HP2011			
#	Nome feature	direzione	Coef
1	n_sentences	succ	2.661
2	n_tokens	succ	2.376
3	n_prepositional_chains	succ	1.958
4	char_per_tok	succ	1.882
5	verbs_num_pers_dist_..	ins	1.802
6	dep_dist_iobj	succ	1.660
7	dep_dist_obj	succ	1.624
8	tokens_per_sent	ins	1.591
9	upos_dist_SCONJ	succ	1.589
10	avg_max_depth	ins	1.560
11	ttr_lemma_chunks_100	succ	1.370
12	upos_dist_ADV	ins	1.330
13	upos_dist_PUNCT	succ	1.188
14	verbal_head_per_sent	ins	1.077
15	verbs_num_pers_dist_..	ins	1.076
16	dep_dist_dep	succ	1.036
17	dep_dist_fixed	ins	1.034
18	dep_dist_dislocated	succ	1.028
19	verbs_num_pers_dist_..	succ	0.984
20	dep_dist_compound	ins	0.978
21	ttr_form_chunks_100	ins	0.954
22	verbs_num_pers_dist_..	succ	0.929
23	dep_dist_aux:pass	succ	0.925
24	avg_subordinate_chain..	ins	0.877
25	upos_dist_PART	ins	0.859
26	avg_token_per_clause	ins	0.855
27	prep_dist_5	succ	0.824
28	prep_dist_4	ins	0.787
29	dep_dist_flat:name	ins	0.783
30	max_links_len	succ	0.767

LOTR			
#	Nome feature	direzione	Coef
1	avg_token_per_clause	ins	2.019
2	tokens_per_sent	ins	1.246
3	char_per_tok	succ	1.229
4	ttr_lemma_chunks_1..	ins	1.104
5	dep_dist_obj	succ	1.020
6	verb_edges_dist_5	succ	1.003
7	dep_dist_parataxis	ins	0.998
8	aux_form_dist_Ger	ins	0.986
9	prep_dist_5	succ	0.945
10	verbs_mood_dist_Cnd	ins	0.935
11	dep_dist_flat	succ	0.913
12	verbal_root_perc	ins	0.911
13	aux_num_pers_dist_..	succ	0.909
14	dep_dist_nsubj:pass	ins	0.908
15	verbs_form_dist_Part	succ	0.859
16	dep_dist_csubj	ins	0.849
17	upos_dist_PUNCT	succ	0.831
18	dep_dist_orphan	succ	0.788
19	dep_dist_amod	ins	0.766
20	verb_edges_dist_4	succ	0.762
21	upos_dist_VERB	ins	0.760
22	dep_dist_punct	succ	0.722
23	dep_dist_nsubj	succ	0.719
24	ttr_form_chunks_100	ins	0.702
25	dep_dist_advcl	ins	0.683
26	upos_dist_NOUN	succ	0.682
27	verbs_mood_dist_Imp	succ	0.669
28	upos_dist_NUM	ins	0.654
29	verbs_form_dist_Fin	ins	0.634
30	verb_edges_dist_3	ins	0.628

Tabella 5.2: Prime 30 feature per peso di ogni corpus per la classificazione linguistica in-domain.

I pesi assegnati dai classificatori (vedi tabella 5.2) hanno restituito dei risultati che in buona parte supportano le osservazioni emerse nella fase di studio della significatività statistica delle feature (sezione 3.2). Tra le feature più importanti per determinare la classe successo ritroviamo il numero totale di frasi, il numero totale di token (per i corpora HPTOT e HP2011) e l'elevata presenza di segni di punteggiatura. Inoltre anche i pesi del classificatore suggeriscono che le frasi più lunghe e complesse siano generalmente una caratteristica della classe insuccesso (avg_token_per_clause, tokens_per_sent, avg_max_depth). Infine, una caratteristica considerata molto importante per la classe successo in tutti e tre corpora è la media di caratteri per token (char_per_tok), che non era tra le 30 feature statisticamente più significative calcolate attraverso il rank-biserial.

Nelle seguenti tabelle sono riportati i risultati dei classificatori addestrati sui modelli lessicali e valutati in-domain.

-	**accuracy**	**baseline_mf**	**baseline_u**
forma	69.56%	50.16%	50.45%
lemma	69.50%	50.16%	49.38%
pos	61.33%	50.16%	50.03%
carattere	63.52%	50.16%	49.79%
totale	69.95%	50.16%	49.85%

Tabella 5.3: Risultati classificazione lessicale HPTOT in-domain.

-	**accuracy**	**baseline_mf**	**baseline_u**
forma	67.61%	52.14%	49.90%
lemma	67.56%	52.14%	50.22%
pos	60.63%	52.14%	50.49%
carattere	62.79%	52.14%	49.74%
totale	67.85%	52.14%	50.22%

Tabella 5.4: Risultati classificazione lessicale HP2011 in-domain.

-	**accuracy**	**baseline_mf**	**baseline_u**
forma	69.97%	56.43%	50.75%
lemma	67.28%	56.43%	47.63%
pos	62.88%	56.43%	51.28%
carattere	66.53%	56.43%	51.28%
totale	69.32%	56.43%	52.14%

Tabella 5.5: Risultati classificazione lessicale LOTR in-domain.

Come si può osservare nelle tabelle 5.3, 5.4 e 5.5 i risultati derivati dai modelli lessicali sono mediamente migliori rispetto a quelli del modello linguistico. In tutti e tre i corpora

vediamo che gli n-grammi di forma e lemma permettono di raggiungere una accuracy più alta rispetto a gli n-grammi di parti del discorso (pos) e caratteri (prefissi e suffissi); con l'eccezione del corpus LOTR, dove notiamo un ottimo risultato, 66.53% e +10% dalla baseline_mf, anche per gli n-grammi di caratteri. Si può anche notare come in generale il modello totale non riesca, in nessuno dei casi, a distanziarsi significativamente dai modelli di forma e lemma.

Gli esiti migliori derivano anche in questo caso dal corpus HPTOT, che per i modelli di forme, lemmi e totale sfiora il 70% di accuracy e il +20% da entrambe le baseline. Il classificatore addestrato con il corpus LOTR, che con il modello linguistico aveva restituito i risultati peggiori, con i modelli lessicali registra un importante incremento delle prestazioni, sfiorando il 70% di accuracy e uno scarto di 15 punti percentuali rispetto alla baseline_mf, con il modello basato sulle forme e con il modello totale.

```
1  def main():
2      # prompt per scegliere il training set
3      dataset = input('scegli dataset: ')
4      tipo = input('forma, lemma, pos, carattere o totale: ')
5
6      # costruisco il training set
7      featuredicts, labels = get_corpus(dataset, tipo)
8
9      # trasformo la lista di dizionari in una matrice sparsa
       compressa
10     vectorizer = DictVectorizer()
11     features = vectorizer.fit_transform(featuredicts)
12
13     # normalizzo l'intervallo di valori delle feature
14     scaler = MaxAbsScaler()
15     features_norm = scaler.fit_transform(features)
16
17     # definisco il classificatore e svolgo la 10-fold
       crossvalidation
18     classifier = LinearSVC(max_iter=10000)
```

```
19      res = list(cross_val_score(classifier, features_norm, labels, cv
    =10, scoring='accuracy'))
20      acc = sum(res) / len(res)
21
22      # ricavo la baseline_mf
23      dummy_mf = DummyClassifier(strategy='most_frequent')
24      dummy_mf.fit(X=None, y=labels)
25      baseline_mf = dummy_mf.score(X=None, y=labels)
26
27      # ricavo la baseline_u
28      dummy_u = DummyClassifier(strategy='uniform')
29      dummy_u.fit(X=None, y=labels)
30      baseline_u = dummy_u.score(X=None, y=labels)
31
32      # stampo i risultati
33      with open(f'risclassificazione2/{dataset}_ngrammi.csv', 'a',
    encoding='utf-8') as f:
34          f.write(f'{dataset}_{tipo}\t{acc}\t{baseline_mf}\t{
    baseline_u}\n')
```

Code 5.1: Codice utilizzato per gli esperimenti di classificazione lessicale in-domain

Nelle due tabelle seguenti sono riportate le distribuzioni dei diversi tipi di n-grammi, utilizzati nel modello lessicale totale, tra le 1000 feature più importanti per la classe successo, e le 1000 più importanti per la classe insuccesso.

-	**forme**	**lemmi**	**pos**	**prefissi**	**suffissi**
HPTOT	26.7%	24.5%	16.9%	19.1%	12.8%
HP2011	21.4%	19.7%	31.2%	16.6%	10.2%
LOTR	19.9%	22.3%	29.7%	17.1%	11.0%

Tabella 5.6: Distribuzione delle 1000 feature più importanti per la classe successo.

-	forme	lemmi	pos	prefissi	suffissi
HPTOT	22.3%	22.2%	25.0%	15.8%	14.7%
HP2011	19.6%	18.6%	33.9%	15.8%	12.1%
LOTR	18.8%	19.0%	34.9%	16.0%	11.3%

Tabella 5.7: Distribuzione delle 1000 feature più importanti per la classe insuccesso.

5.2 Out-domain

Di seguito sono riportati i risultati dei sistemi di classificazione addestrati con il modello linguistico e valutati sui testi al di fuori del proprio dominio. Quindi il classificatore addestrato con il corpus HPTOT, è stato valutato attraverso il corpus LOTR, e viceversa.

training	test	accuracy	baseline_mf	baseline_u
LOTR	HPTOT	56.76%	50.16%	49.30%
HPTOT	LOTR	59.22%	56.43%	48.28%

Tabella 5.8: Risultati della classificazione linguistica out-domain.

I risultati della tabella 5.8 mostrano una importante diminuzione delle prestazioni dei classificatori sui testi esterni al loro dominio, mantenendo comunque un discreto scarto positivo (dai 3 ai 9 punti percentuali) rispetto alle baseline.

```
1     def cross_set_validation(self, cross_set: list) -> dict:
2         # addestro il classificatore ed effettuo la predizione
3         classificatore = svm.LinearSVC()
4         classificatore.fit(self.featuresvectorslist, self.
      resultslist)
5         risclassificazione = classificatore.predict(cross_set[0])
6
7         # se dovesse esserci una incongruenza tra il numero di
      risultati reali e quelli predetti genero una eccezione
8         if len(risclassificazione) != len(cross_set[1]):
```

```
            raise Exception

        # calcolo la baseline per most frequency con il dummy classifier
        dummy_mf = DummyClassifier(strategy='most_frequent')
        dummy_mf.fit(self.featuresvectorslist, self.resultslist)
        baseline_mf = dummy_mf.score(X=cross_set[0], y=cross_set[1])

        # calcolo la baseline_u
        dummy_u = DummyClassifier(strategy='uniform')
        dummy_u.fit(self.featuresvectorslist, self.resultslist)
        baseline_u = dummy_u.score(X=cross_set[0], y=cross_set[1])

        # calcolo manualmente l'accuracy
        esatti = 0
        for i in range(len(risclassificazione)):
            if risclassificazione[i] == cross_set[1][i]:
                esatti += 1
        accuracy = esatti / len(risclassificazione)

        # restituisco i risultati con un dizionario
        return {'accuracy': accuracy, 'baseline_mf': baseline_mf, 'baseline_u': baseline_u}
```

Code 5.2: Metodo della classe ”ClassificazioneLinguistica” utilizzata per gli esperimenti linguistici out-domain e cross-time.

Nella seguenti tabelle sono invece riportati i risultati ottenuti out-domain dai classificatori a modello lessicale.

tipo	training	test	accuracy	baseline_mf	baseline_u
forma	LOTR	HPTOT	49.87%	50.16%	50.11%
lemmma	LOTR	HPTOT	50.24%	50.16%	51.24%
pos	LOTR	HPTOT	50.36%	50.16%	51.60%
carattere	LOTR	HPTOT	49.84%	50.16%	50.75%
totale	LOTR	HPTOT	50.66%	50.16%	49.92%

Tabella 5.9: Risultati valutazione out-domain dei classificatori basati sui modelli lessicali, addestrati con il corpus LOTR e valutati con il corpus HPTOT.

tipo	training	test	accuracy	baseline_mf	baseline_u
forma	HPTOT	LOTR	47.74%	56.43%	50.64%
lemma	HPTOT	LOTR	44.09%	56.43%	50.03%
pos	HPTOT	LOTR	54.93%	56.43%	50.43%
carattere	HPTOT	LOTR	51.71%	56.43%	51.46%
totale	HPTOT	LOTR	43.45%	56.43%	51.53%

Tabella 5.10: Risultati valutazione out-domain dei classificatori basati sui modelli lessicali, addestrati con il corpus HPTOT e valutati con il corpus LOTR.

I classificatori addestrati sui modelli lessicali si sono rivelati completamente inefficaci nella predizione dei testi esterni al proprio dominio: l'accuracy non supera mai significativamente la baseline_mf, risultando spesso inferiore.

5.3 Cross-time

In questa sezione saranno mostrati i risultati della classificazione cross-time: il corpus HP2011 è stato utilizzato per l'addestramento del classificatore, mentre per la valutazione sono stati utilizzati a turno i corpora di testi di Harry Potter degli altri anni.
Nelle due tabelle successive sono riportati i risultati del modello linguistico e del modello lessicale totale.

-	**accuracy**	**baseline_mf**	**baseline_u**
2003	55.38%	70.76%	53.84%
2004	59.78%	87.34%	50.75%
2005	66.34%	85.49%	49.38%
2006	65.67%	84.92%	49.14%
2007	64.69%	81.37%	50.24%
2008	65.31%	65.18%	49.05%
2009	65.50%	63.19%	49.18%
2010	64.04%	70.65%	49.72%
2012	61.76%	38.99%	49.74%
2013	62.04%	24.38%	51.60%
2014	61.05%	20.84%	48.26%
2015	61.13%	23.40%	50.83%
2016	57.24%	21.97%	48.29%
2017	64.56%	20.27%	50.00%
2018	62.56%	21.65%	44.03%
2019	59.55%	19.60%	50.24%
2020	57.89%	36.45%	47.56%

Tabella 5.11: Risultati classificazione linguistica cross-time.

-	**accuracy**	**baseline_mf**	**baseline_u**
2003	70.76%	70.76%	49.23%
2004	87.34%	87.34%	50.60%
2005	85.49%	85.49%	49.22%
2006	84.73%	84.92%	48.51%
2007	81.10%	81.37%	49.86%
2008	65.18%	65.18%	50.62%
2009	63.50%	63.19%	49.50%
2010	70.70%	70.65%	48.81%
2012	39.10%	38.99%	50.72%
2013	24.50%	24.38%	50.75%
2014	21.12%	20.84%	51.13%
2015	23.55%	23.40%	50.75%
2016	22.06%	21.97%	45.24%
2017	20.57%	20.27%	48.79%
2018	22.75%	21.65%	51.00%
2019	19.60%	19.60%	48.52%
2020	36.25%	36.45%	48.14%

Tabella 5.12: Risultati classificazione lessicale cross-time.

I risultati di questi esperimenti di classificazione sono difficili da interpretare, a causa dell'eccessivo sbilanciamento tra le classi nei corpora utilizzati per la valutazione. Emerge però che il classificatore a modello lessicale abbia restituito sempre valori quasi identici alla baseline_mf, limitandosi quindi ad assegnare la classe più frequente. Invece il classificatore linguistico, a prescindere dalla variazione della baseline_mf, ha ottenuto dei risultati più stabili e significativi.

Per valutare quantitativamente la differenza tra le prestazioni dei due sistemi di classificazione, sono state calcolate la media aritmetica e la deviazione standard.

-	**media**	**deviazione**
linguistica	62.02	3.16
lessicale	49.31	26.45

Tabella 5.13: Media aritmetica e deviazione standard dell'accuracy dei risultati della classificazione cross-time.

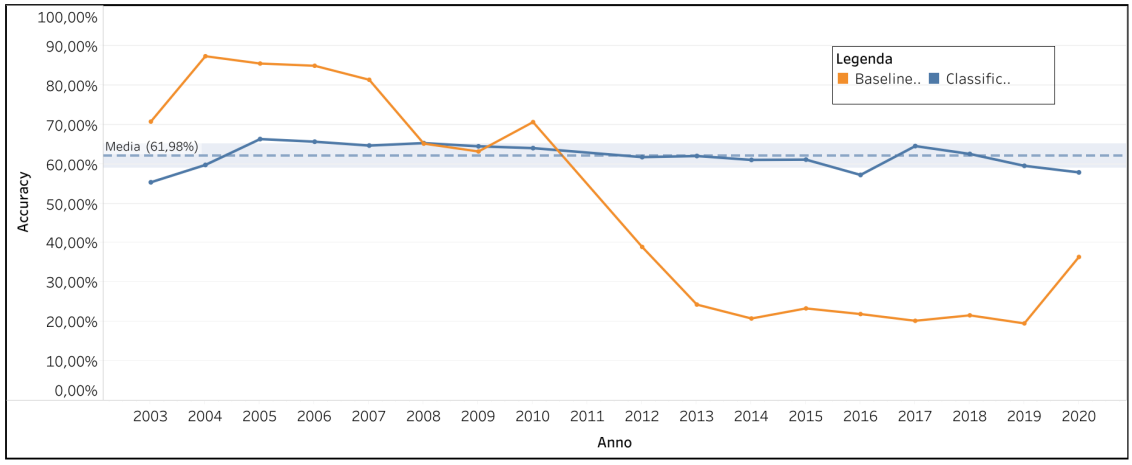

Figura 5.1: Grafico dei risultati della classificazione cross-time a modello linguistico.

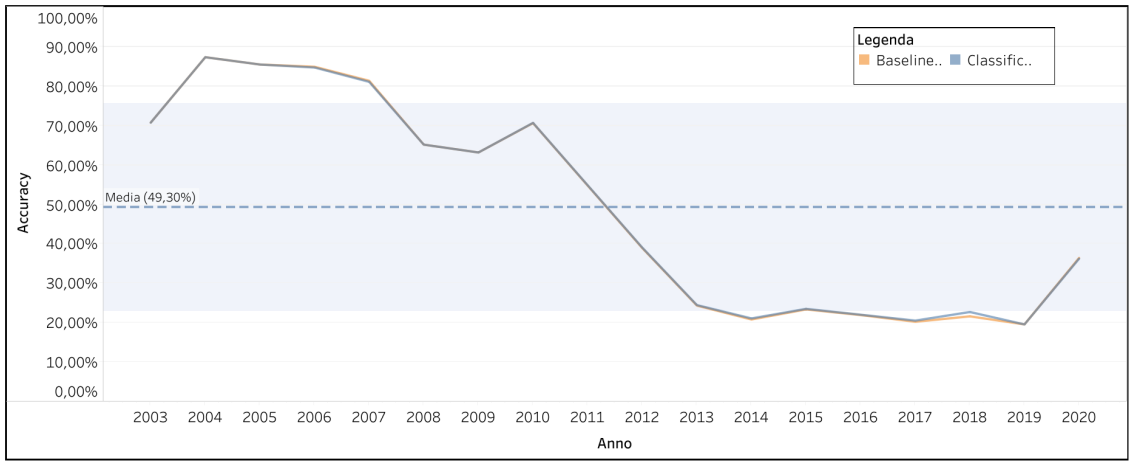

Figura 5.2: Grafico dei risultati della classificazione cross-time a modello lessicale.

```
def main():
    # prompt per scegliere il training e il test set
    training = input('training set: ')
    test = input('test set: ')
    tipo = input('tipo (forma, lemma, pos, carattere, totale): ')

    # ricavo i corpora
    training_featuredicts, training_results = get_corpus(training,
    tipo)
    test_featuredicts, test_results = get_corpus(test, tipo)

    # trasformo i dizionari di feature in matrici sparse compresse
    vectorizer = DictVectorizer()
    training_features = vectorizer.fit_transform(
    training_featuredicts)
    test_features = vectorizer.fit_transform(test_featuredicts)

    # ricavo le dimensioni delle matrici
    training_shape = training_features.get_shape()
    test_shape = test_features.get_shape()

    # modifico le dimensioni delle matrici, per renderle della
    stessa lunghezza
    if training_shape[1] > test_shape[1]:
        test_features.resize((test_shape[0], training_shape[1]))
    else:
        training_features.resize((training_shape[0], test_shape[1]))

    # normalizzo l'intervallo di valori delle feature
    scaler = MaxAbsScaler()
    training_features_norm = scaler.fit_transform(training_features)
    test_features_norm = scaler.fit_transform(test_features)

    # addestro il classificatore e lo valuto
    classifier = LinearSVC()
```

```
33      classifier.fit(X=training_features_norm, y=training_results)
34      res = classifier.score(X=test_features_norm, y=test_results)
35
36      # calcolo la baseline_mf
37      dummy_mf = DummyClassifier(strategy='most_frequent')
38      dummy_mf.fit(X=training_features_norm, y=training_results)
39      baseline_mf = dummy_mf.score(X=test_features_norm, y=
        test_results)
40
41      # calcolo la baseline_u
42      dummy_u = DummyClassifier(strategy='uniform')
43      dummy_u.fit(X=training_features_norm, y=training_results)
44      baseline_u = dummy_u.score(X=test_features_norm, y=test_results)
45
46      # stampo i risultati
47      with open(f'risclassificazione_cross-anni.csv', 'a', encoding='
        utf-8') as f:
48          f.write(f'{training}\t{test}\t{tipo}_ngrammi\t{res}\t{
        baseline_mf}\t{baseline_u}\n')
```

Code 5.3: Codice utilizzato per gli esperimenti lessicali out-domain e cross-time

5.4 Il lessico del successo

Questa sezione riguarda il classificatore addestrato esclusivamente con il modello lessicale di uni-grammi di token. Lo scopo di questo esperimento è stato quello di estrarre i pesi delle feature del classificatore, ottenendo così una classifica del lessico correlato al successo.

Di seguito sono presentati i primi 50 token per peso verso la classe successo di ogni corpus.

HPTOT			HP2011			LOTR		
#	Token	Coef	#	Token	Coef	#	Token	Coef
1	^^	1.1104	1	Hermione	0.5255	1	Thorin	0.2169
2	Hermione	1.0837	2	Granger	0.3428	2	Kili	0.1696
3	moretto	0.8468	3	Scorpius	0.3194	3	po’	0.1569
4	Trattato	0.8406	4	spiazzato	0.2783	4	com’	0.1346
5	tutore	0.8177	5	Malfoy	0.2678	5	capito	0.1336
6	sommessamente	0.7959	6	–	0.2577	6	appena	0.1318
7	Granger	0.7927	7	,	0.2559	7	Fili	0.1295
8	XD	0.7875	8	Dipartimento	0.2493	8	,	0.1203
9	*****	0.7542	9	sì	0.2414	9	titolo	0.1191
10	domandato	0.7340	10	Lo	0.2398	10	E	0.1185
11	Tonks	0.7323	11	Facebook	0.2372	11	dimentichi	0.1182
12	indetto	0.6960	12	Teddy…	0.2363	12	tutte	0.1169
13	Ch	0.6774	13	u	0.2322	13	colmo	0.1164
14	shot	0.6725	14	difficile	0.2316	14	Quanto	0.1163
15	chap	0.6721	15	mano	0.2300	15	E’	0.1161
16	Spoiler	0.6654	16	freddo	0.2272	16	-Angolino	0.1142
17	,	0.6536	17	ripetendo	0.2231	17	Nano	0.1135
18	indugiate	0.6382	18	25	0.2219	18	labbro	0.1101
19	Forse	0.6206	19	aveva	0.2197	19	Faramir	0.1094
20	Fine	0.6182	20	sé	0.2195	20	sentiti	0.1082
21	ringraziare	0.6066	21	benché	0.2186	21	costume	0.1079
22	Harry	0.6020	22	interpellò	0.2180	22	inutilmente	0.1077
23	biondino	0.6002	23	Lucius	0.2161	23	!	0.1074
24	Colpiva	0.5925	24	comparire	0.2151	24	esattamente	0.1043
25	stories	0.5874	25	Durante	0.2139	25	scivolassero	0.1038
26	sottolineava	0.5831	26	approvò	0.2136	26	io	0.1020
27	ceffone	0.5823	27	George	0.2130	27	»	0.0999
28	scatole	0.5823	28	sicuramente	0.2099	28	«	0.0987
29	dite	0.5793	29	su	0.2083	29	macchiati	0.0978
30	nuovo	0.5727	30	extra	0.2069	30	cui	0.0973
31	Co…	0.5682	31	rese	0.2068	31	Credo	0.0972
32	Adoro	0.5595	32	totalmente	0.2062	32	autrice	0.0952
33	Prestavolto	0.5573	33	pezzo	0.2049	33	inizialmente	0.0941
34	Scorpius	0.5558	34	correggere	0.2044	34	l’	0.0932
35	sesso	0.5550	35	profumo	0.2043	35	mano	0.0931
36	accetterà	0.5526	36	panico	0.2033	36	ammirare	0.0919
37	interrogativamente	0.5485	37	Place	0.2029	37	meravigliose	0.0916
38	sedici	0.5459	38	apprestò	0.2027	38	Avrebbe	0.0912
39	Frassino	0.5451	39	anticipò	0.2023	39	lisciando	0.0911
40	Dolores	0.5429	40	posticipare	0.2020	40	Amico	0.0910
41	improvvisamente	0.5426	41	A	0.2016	41	tuo	0.0908
42	appartenevano..	0.5407	42	gesto	0.1999	42	me…	0.0908
43	Domandò	0.5367	43	Arriverà	0.1998	43	pausa	0.0908
44	SOLO	0.5360	44	esempio	0.1992	44	trattando	0.0906
45	!	0.5358	45	Cioè	0.1985	45	giurò	0.0905
46	****	0.5341	46	rispondere	0.1974	46	Sorridendo	0.0893
47	Immergo	0.5323	47	u.	0.1974	47	Mai	0.0892
48	sfiorandole	0.5312	48	Arrivate	0.1971	48	tremendamente	0.0892
49	scheda	0.5310	49	.	0.1965	49	farò	0.0892
50	catturata	0.5286	50	Draco	0.1963	50	camminò	0.0891

Tabella 5.14: Prime 50 feature per peso di ogni corpus per la classificazione linguistica in-domain.

Si notano tra le prime posizioni della tabella 5.14 diversi nomi e cognomi di persona ap-

partenenti a dei personaggi delle opere madri, che probabilmente suscitano un particolare interesse nei fan: Hermione Granger, Draco Malfoy e Scorpius Malfoy per Harry Potter; Thorin, Fili e Kili per Il Signore Degli Anelli. Inoltre, per tutti e tre i corpora, diversi segni di punteggiatura, in particolare la virgola, sono tra i token più importanti per il successo; questo rafforza l'osservazione che i testi di successo siano caratterizzati da un maggiore utilizzo di punteggiatura rispetto a quelli di insuccesso.

5.5 Discussione dei risultati

Dai risultati degli esperimenti di classificazione sono emerse diverse osservazioni interessanti, che sono esposte in questa sezione.

I classificatori addestrati con i modelli lessicali si sono dimostrati più efficaci nella predizione in-domain, in particolare per il corpus LOTR. Il modello linguistico si è comunque rivelato efficace nella predizione in-domain.

Nello scenario out-domain invece le prestazioni si sono notevolmente abbassate, però è stato osservato che: i classificatori a modello lessicale hanno restituito dei risultati pessimi; invece i classificatori basati su feature linguistiche hanno preservato un discreto scarto (dai 3 ai 9 punti percentuali) rispetto alle baseline, dimostrando quindi una maggiore capacità di generalizzazione.

La maggiore affidabilità del modello linguistico è emersa anche nello scenario cross-time: ha ottenuto risultati mediamente migliori (62.02% contro il 49.31%) e anche molto più stabili (deviazione standard di 3.16 rispetto a quella di 26.45 del modello lessicale).

Infine l'analisi dei pesi dei classificatori ha sia rafforzato le osservazioni emerse durante le analisi statistiche, riguardanti alcune feature apparentemente correlate al successo (vedi sez. 3.2), che mostrato ulteriori caratteristiche importanti per l'assegnazione della classe successo, come l'utilizzo di parole mediamente più lunghe e la presenza nel racconto di alcuni personaggi che sembrano suscitare interesse nel lettore.

6. Conclusioni

Questo studio si è concentrato sulle caratteristiche del successo e la predizione automatica di esso, per i racconti del genere fanfiction.
Per questo scopo è stato costruito un dataset di oltre 55000 testi, distinti attorno a due temi (Harry Potter e Il Signore Degli Anelli). Il dataset è stato analizzato per stabilire una definizione di successo e insuccesso: le storie che hanno ricevuto meno di due recensioni al proprio primo capitolo sono state considerate di insuccesso, mentre le storie che hanno ottenuto più di 6 recensioni sono state considerate di successo.
Sono stati utilizzati degli strumenti di TAL (Trattamento Automatico del Linguaggio) per estrarre le feature linguistiche e lessicali dai testi. Ne è conseguita l'analisi della significatività statistica delle caratteristiche linguistiche attraverso il Mann-Whitney U test. Per le feature significative è stato calcolato il rank-biserial, in modo da ottenere una classifica delle feature statisticamente correlate al successo della storia.
In seguito, si è scelto di interpretare la predizione automatica del successo di un racconto come un task di classificazione, tra le due classi "successo" e "insuccesso". L'obiettivo di questa fase dello studio è stato quello di comprendere se, attraverso le feature estratte, fosse possibile addestrare un classificatore lineare per ottenere delle prestazioni interessanti, valutandolo su tre diversi scenari: all'interno del proprio dominio (in-domain), all'esterno del proprio dominio tematico (out-domain), all'esterno del proprio dominio temporale (cross-time). Inoltre sono state analizzate le differenze nel comportamento tra i classificatori basati sul modello linguistico, e quelli basati sul modello lessicale.
Le osservazioni emerse durante le diverse fasi dello studio permettono di rispondere alle domande di ricerca poste nell'introduzione:

1. Esistono delle caratteristiche significative per il successo di un racconto?

Nella sezione 3.2, attraverso il Mann-Whitney U test, è stata analizzata la significatività statistica delle 144 feature linguistiche estratte, riportando 124 feature significative per il corpus contenente tutti i testi a tema Harry Potter, 85 per il corpus formato dai testi a tema

Harry Potter scritti nel 2011, 64 per i testi a tema Il Signore Degli Anelli, e 45 feature significative per tutti e tre i corpora.

2. Quali tra queste caratteristiche sono maggiormente correlate al successo?

Il calcolo del rank-biserial (sezione 3.2) e l'analisi dei pesi assegnati dai classificatori (sezione 5.1) hanno permesso di ricavare le caratteristiche più importanti per il successo di un racconto, di seguito sono elencate le più interessanti:

- maggiore lunghezza dei testi in termini di token e frasi;
- token mediamente più lunghi;
- frasi più brevi e semplici;
- utilizzo più frequente dei segni di punteggiatura;
- presenza di verbi all'imperativo, verbi coniugati alla seconda persona e interiezioni, che fanno desumere una maggiore frequenza dell'uso del discorso diretto;
- presenza nel racconto di personaggi precisi derivati dall'opera madre, apparentemente interessanti per i lettori.

3. È possibile prevedere il successo di un racconto attraverso queste caratteristiche?

La previsione del successo di un racconto si è dimostrata possibile: i risultati degli esperimenti di classificazione descritti nel capitolo 5 hanno riportato delle buone prestazioni, sia per il modello linguistico (fino a 15 punti percentuali in più rispetto alla baseline_mf) che per i modelli lessicali (fino a 20 punti percentuali in più rispetto alla baseline_mf), nella predizione del successo su testi dello stesso dominio.

4. Le caratteristiche cambiano rispetto al dominio tematico dei testi?

I risultati degli esperimenti di classificazione presentati nella sezione 5.2 hanno mostrato che l'efficacia delle feature varia particolarmente rispetto al dominio tematico dei testi: le prestazioni della classificazione out-domain sono state decisamente più deboli, solo il modello linguistico è riuscito a migliorare l'accuratezza dalla baseline_mf (dai 3 ai 6 punti percentuali).

5. Le caratteristiche cambiano nel tempo?

Per rispondere a questa domanda i classificatori sono stati valutati nello scenario cross-time, presentato nella sezione 4.3, dove i testi di Harry Potter pubblicati nel 2011 sono stati usati come training set fisso per il classificatore, mentre per la valutazione si sono alternati i restanti testi, raggruppati per anno.
I risultati presentati nella sezione 5.3 mostrano che anche rispetto al tempo varia l'efficacia delle feature nella predizione del successo: l'accuracy delle prove di classificazione cross-time è decisamente più bassa rispetto a quella delle classificazioni in-domain; anche in questo caso solo i modelli linguistici hanno preservato un minimo scarto rispetto alla baseline_mf.

6. Per la predizione del successo le caratteristiche linguistiche e lessicali si comportano diversamente?

Nei risultati degli esperimenti di classificazione svolti in questo studio (capitolo 5), entrambi i gruppi di caratteristiche si sono dimostrati efficaci nella predizione del successo in-domain, con un leggero vantaggio per i modelli lessicali. Nello scenario out-domain i modelli che sfruttano solo caratteristiche linguistiche si sono rilevati più efficienti, riuscendo a mantenere un minimo scarto dalle baseline. Anche nello scenario cross-time sono stati questi sistemi a restituire i risultati migliori: hanno riportato una maggiore accuracy media (62.02% contro il 49.31%) e dei risultati molto più stabili (3.16 di deviazione standard rispetto a 26.45). Si può quindi osservare che le feature linguistiche abbiano dimostrato una maggiore capacità di generalizzazione, mentre quelle lessicali si siano rivelate strettamente legate al dominio tematico e temporale dei testi.

Questa tesi non vuole e non può essere una ricerca conclusiva sull'argomento, essendo innanzi tutto un elaborato di laurea triennale. Nonostante ciò, i risultati sono stati promettenti, quindi questo potrebbe essere uno studio pilota che mostra una interessante direzione di ricerca da approfondire con degli studi avanzati. Per i possibili sviluppi futuri per esempio potrebbe essere analizzato un diverso dataset, più ampio e contenente testi di generi differenti; oppure, si potrebbe definire in modo differente il successo di un racconto; inol-

tre, si potrebbero utilizzare altri algoritmi di apprendimento automatico, come i *neural language model*.

Bibliografia

Dominique Brunato, Andrea Cimino, Felice Dell'Orletta, Giulia Venturi, and Simonetta Montemagni. 2020. *Profiling-UD: a Tool for Linguistic Profiling of Texts*. In "Proceedings of the Twelfth Language Resources and Evaluation Conference", pp. 7145–7151, Marseille, France. European Language Resources Association.

Fan, Chang Rong-En, Hsieh Kai-Wei, Wang Cho-Jui, Lin Xiang-Rui, and Chih-Jen. 2008. *LIBLINEAR: a library for large linear classification*. In "Journal of Machine Learning Research", 9, 1871-1874. 10.1145/1390681.1442794.

Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. 2003. *A Practical Guide to Support Vector Classification*.

Simonetta Montemagni. 2013. *Tecnologie linguistico-computazionali e monitoraggio della lingua italiana*. In: "Studi Italiani di Linguistica Teorica e Applicata (SILTA) Anno XLII", Numero 1, pp. 145-172.

Milan Straka, Jan Hajič, and Jana Straková. 2016. *UDPipe: Trainable Pipeline for Processing CoNLL-U Files Performing Tokenization, Morphological Analysis, POS Tagging and Parsing*. In "Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)", pages 4290–4297, Portorož, Slovenia. European Language Resources Association (ELRA).

Sitografia

[1] *Documentazione Python, voce collections*. URL: https://docs.python.org/3/library/collections.html#module-collections.

[2] *Documentazione Python, voce Regular expression operations*. URL: https://docs.python.org/3/library/re.html.

[3] *Documentazione scikit-learn, voce DictVectorizer*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.DictVectorizer.html.

[4] *Documentazione scikit-learn, voce DummyClassifier*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html.

[5] *Documentazione scikit-learn, voce LinearSVC*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html.

[6] *Documentazione scikit-learn, voce MaxAbsScaler*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MaxAbsScaler.html.

[7] *Documentazione scikit-learn, voce MinMaxScaler*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.

[8] *Documentazione scikit-learn, voce Support Vector Machine*. URL: https://scikit-learn.org/stable/modules/svm.html.

[9] *Documentazione Scipy*. URL: https://docs.scipy.org/doc/scipy/index.html#.

[10] *EFP Fanfiction*. URL: https://efpfanfic.net/.

[11] *Italian Natural Language Processing Lab*. URL: http://www.italianlp.it/.

[12] *Italian Natural Language Processing Lab, voce Profiling-UD*. URL: http://www.italianlp.it/demo/profiling-ud/.

[13] *LIBLINEAR*. URL: https://www.csie.ntu.edu.tw/~cjlin/liblinear/.

[14] *Scrapy*. URL: scrapy.org.

[15] *Universal Dependencies*. URL: https://universaldependencies.org/.

[16] *Universal Dependencies, voce CoNLL-U Format*. URL: https://universaldependencies.org/format.html.

[17] *Wikipedia, voce Mann-Whitney U test*. URL: https://en.wikipedia.org/wiki/Mann-Whitney_U_test.

[18] *Wikipedia, voce p-value*. URL: https://en.wikipedia.org/wiki/P-value.

Appendice A: codice

In questa appendice sarà riportato il codice citato nella relazione.

clean.py

La funzione definita in questo script, data una stringa, effettua delle sostituzioni che ne ripuliscono il contenuto, restituendo in output la stringa ripulita.

```
1  import re
2  def cleantext(rawtext: str) -> str:
3      cleaned = re.sub(r'(\.) ', r'\1', rawtext)
4      cleaned = re.sub(r'\u2028', r'\n', cleaned)  # carattere
       problematico per il tool
5      cleaned = re.sub(r'\u0085', r'\n', cleaned)  # carattere
       problematico per il tool
6      cleaned = re.sub(r'([a-zA-Z,;:\(\)\[\]ùèéàòìÈ\u00a0 ])\r\n', r'
       \1 ', cleaned)
7      cleaned = re.sub(r'([a-zA-Z,;:\(\)\[\]ùèéàòìÈ\u00a0 ])\n', r'\1
       ', cleaned)  # stessa cosa solo per \n
8      cleaned = re.sub(r'\u2026', r'...', cleaned)  # sistemo i
       puntini di sospensione
9      cleaned = re.sub(r'(\.\.\.\.+)', r'...', cleaned)  # sistemo i
       punti di sospensione maggiori di 3
10     cleaned = re.sub(r'([\.,;:!?])(\S)', r'\1 \2', cleaned)  #
       aggiungo uno spazio se non è presente dopo: . , ; : ! ?
11     cleaned = re.sub(r'\s([\.,;:!?])', r'\1', cleaned)  # rimuovo
       gli spazi se sono presenti prima di: . , ; : ! ?
12     cleaned = re.sub(r'  +', r' ', cleaned)  # rimuovo gli spazi
       ripetuti
13     return cleaned
```

Code 6.1: Codice utilizzato per gli esperimenti lessicali out-domain e cross-time

listofdicts.py

La classe definita in questo script è servita a gestire il dataset iniziale, per poterlo analizzare. Data una lista di dizionari, i metodi permettono diverse manipolazioni, trattando la struttura dati come una tabella.

```
1  import json
2  import csv
3  import pyarrow as pa
4  import pyarrow.parquet as pq
5  import matplotlib.pyplot as plt
6
7
8  class listofdicts:
9
10
11      dataset = ''
12      file_in = ''
13
14      def __init__(self, Input, json_in: bool = False):
15          if json_in == True:
16              self.file_in = Input
17              with open(Input, 'r', encoding='utf-8') as raw:
18                  # assegno all'attributo dataset il valore dell'
    oggetto json del file di input.
19                  dataset = json.loads(raw.read())
20          else:
21              # in questo modo invece posso dare direttamente una
    lista di dizionari
22              dataset = Input
23
24          i = 0
25          keys = set(dataset[0].keys())
26          if type(dataset) != list:
27              raise TypeError
28          while i <= (len(dataset)) - 1:
```

```
29              if type(dataset[i]) != dict:
30                  raise TypeError
31              if set(dataset[i].keys()) != keys:
32                  raise TypeError
33              i += 1
34
35          self.dataset = dataset
36
37      def __repr__(self):
38          # metodo che imposta la rappresentazione del nostro oggetto.
39          return '{}'.format(self.dataset)
40
41      def tocsv(self, file_output: str, columns: list, separator: str
    = ','):
42          # metodo per creare un un file .csv con le colonne
    selezionate nel parametro "columns".
43          with open(file_output, 'w', newline='', encoding='utf-8') as
     file_out:
44              writer = csv.writer(file_out, delimiter=separator)
45
46              # prima scrivo l'intestazione
47              writer.writerow([col for col in columns])
48
49              # poi scrivo ogni oggetto contenuto nel dataset
50              for item in self.dataset:
51                  writer.writerow([item[col] for col in columns])
52
53      def toparquet(self, file_output: str):
54          # crea un file parquet, che permettte una compressione più
    efficente rispetto al csv
55
56          # prima devo convertire il mio dataset in una tabella
    pyarrow
57          pyarrowtab = pa.Table.from_pylist(self.dataset)
58          pq.write_table(pyarrowtab, file_output, compression='BROTLI'
    )
```

```
59
60      def splitfor(self, key: str, dateonlyyear: bool = False) -> dict
        :
61              # divide il dataset in diversi dataset  in base al
        valore di un campo. Il risultato sarà un dizionario di liste di
        dizionari. (ogni lista di dizionari la possiamo considerare un
        dataset e possiamo trasformarla a sua volta in un oggetto
        listofdicts)
62          result = {}
63          field = key
64
65          if dateonlyyear == True:
66              # per normalizzare la data al solo anno, aggiungo al mio
         dataset una colonna chiamata "dataNORM"
67              for item in self.dataset:
68                  appoggio = item[key].split('/')
69                  namenewfield = '{}NORM'.format(key)
70                  item[namenewfield] = appoggio[len(appoggio) - 1]
71                  field = namenewfield
72
73          # creo un set dei valori per i quali abbiamo suddiviso il
        dataset, in modo da averli unici.
74          listofvalues = {item[field] for item in self.dataset}
75
76          for val in listofvalues:
77              result[val] = []
78              for item in self.dataset:
79                  if item[field] == val:
80                      result[val].append(item)
81
82          return result
83
84      def groupby(self, field: str) -> list:
85          # questo metodo raggruppa le righe per il valore di un campo
        , quindi associa al valore di quel campo il numero di righe che
        lo contengono. Il risultato sara una lista di dizionari, ognuno
```

```
    con due coppie chiave:valore.
86          result = []
87          # anche qui, in modo simile al metodo splitfor, prima
    estraggo il set dei valori di quel campo.
88          listofvalues = {item[field] for item in self.dataset}
89          # poi per ogni valore del campo conto le righe che hanno
    quel valore
90          for val in listofvalues:
91              count = 0
92              for item in self.dataset:
93                  if item[field] == val:
94                      count += 1
95              result.append({field: val, 'tot': count})
96          return result
97
98      def tobarchart(self, x: str, y: str, path_output: str, title:
    str, colorRGBA: list = [0, 0, 255, 0.8],
99                     displayedx: str = False, displayedy: str = False)
    :
100         # questo metodo salva le immagini .png dei grafici a barre
    costruiti dalle colonne selezionate attraverso i parametri "x" e
     "y".
101         # per semplificare il processo di selezione delle colonne
    sfrutto pyarrow.
102         pyarrowtab = pa.Table.from_pylist(self.dataset)
103         assex = [str(el) for el in pyarrowtab.column(x).to_pylist()]
104         assey = pyarrowtab.column(y).to_pylist()
105
106         label_x = displayedx if displayedx != False else x
107         label_y = displayedy if displayedy != False else y
108
109         plt.figure(figsize=(24, 10))
110         plt.bar(assex, assey, color=(
111         float(colorRGBA[0]) / 255, float(colorRGBA[1]) / 255, float(
    colorRGBA[2]) / 255, float(colorRGBA[3])))
112         plt.title(title)
```

```
113         plt.xlabel(label_x)
114         plt.ylabel(label_y)
115         for index in range(len(assex)):
116             plt.text(assex[index], assey[index], assey[index], size
    =12)
117         plt.savefig(path_output)
118         plt.close()
119 
120     def sortbykey(self, key: str, reverse: bool = False):
121         # ordina le righe del dataset in base al valore di un campo.
122         self.dataset.sort(key=lambda d: d[key], reverse=reverse)
```

Code 6.2: Codice utilizzato per gli esperimenti lessicali out-domain e cross-time

datasetToCharts.py

In questo file si sfrutta la classe *listofdicts* per generare dei grafici e dei file csv che mostrino per ogni anno la distribuzione delle recensioni nei capitoli del dataset.

```
1  import listofdicts as lod
2  
3  
4  def main():
5      LOR = lod.listofdicts('dataset/DatasetLOR.json', json_in=True)
6      HP = lod.listofdicts('dataset/DatasetHP.json', json_in=True)
7  
8      crea_csv_charts(LOR, 'LOR', barcolor=[0, 200, 255, 0.39])
9      crea_csv_charts(HP, 'HP', barcolor=[255, 0, 0, 0.39])
10 
11 
12 def crea_csv_charts(dataset, title: str, barcolor: list):
13     # suddivido il dataset per anno con il metodo splitfor, il
    risultato sarà un dizionario con ogni chiave che rappresenta un
    anno.
14     datesplitted = dataset.splitfor('Data', dateonlyyear=True)
15     # poi trasformo il dizionario in una lista di liste di dizionari
    .
```

```
16      listeperanno = [value for key, value in datesplitted.items()]
17
18      # per ogni lista:
19      for lista in listeperanno:
20          listalod = lod.listofdicts(lista)
21          # raggruppo per il campo "This_Rec"
22          listagrouped = listalod.groupby("This_Rec")
23          # ordino i dizionari(righe) all'interno per il valore del
    campo "This_Rec"
24          listagroupedsorted = sorted(listagrouped, key=lambda d: d['
    This_Rec'])
25          # prendo l'anno di riferimento di tutte le righe di quella
    lista
26          anno = lista[0]['DataNORM']
27
28          listagroupedsortedlod = lod.listofdicts(listagroupedsorted)
29
30          # creo i file csv con il metodo tocsv
31          outputnamecsv = 'graficiperanno/{}/dati/{}{}chartdata.csv'.
    format(title, anno, title)
32          listagroupedsortedlod.tocsv(outputnamecsv, columns=["
    This_Rec", 'tot'], separator='|')
33
34          # creo i grafici a barre con il metodo tobarchart.
35          outputnameplot = 'graficiperanno/{}/grafici/{}plot{}.png'.
    format(title, title, anno)
36          listagroupedsortedlod.tobarchart("This_Rec", 'tot',
    outputnameplot, '{} anno {}'.format(title, anno),
37                                              displayedx='numero
    recensioni primo capitolo', displayedy='numero storie',
    colorRGBA=barcolor)
38
39  if __name__ == "__main__":
40      main()
```

Code 6.3: Codice utilizzato per gli esperimenti lessicali out-domain e cross-time

ngram_classes.py

In questo file sono state definite le classi utilizzate per ricavare le feature lessicali (ngrammi) dai documenti.

```
1  from collections import namedtuple
2
3  # definisco la classe token, come sottoclasse della tupla, con gli
       attributi: forma, lemma e pos
4  Token = namedtuple('Token', ['forma', 'lemma', 'pos'])
5
6
7  class Corpus:
8      # oggetto che rappresenta un corpus di documenti, con il metodo
       get_feature_dicts si possono estrarre i dizionari di feature di
       ngrammi
9      __slots__ = ('docslist', 'resultslist')
10     # docslist è la lista di oggetti document
11     # resultslist è la lista di risultati
12
13
14     def __init__(self, docslist: list, resultslist: list) -> None:
15         if len(docslist) != len(resultslist):
16             raise Exception
17         self.docslist = docslist
18         self.resultslist = resultslist
19
20     def get_feature_dicts(self, element: str, t_type: str, n_ngram:
       int) -> list:
21         if element == 'lessico':
22             return [document.get_word_ngrams(t_type, n_ngram) for
       document in self.docslist]
23         if element == 'carattere':
24             return [document.get_char_pref_suff_ngrams(n_ngram) for
       document in self.docslist]
25
```

```
26
27
28
29  class Document:
30      # oggetto che rappresenta il documento, i suoi metodi permettono
        di estrarre i dizionari di feature di ngrammi
31      __slots__ = 'sentenceslist'
32      # sentenceslist è la lista delle frasi del documento, ogni frase
        sarà un oggetto sentence
33
34
35      def __init__(self, lineslist: list) -> None:
36          sentenceslist = []  # conterrà la lista delle frasi
37          tmp_frase = []  # lista d'appoggio
38          # iteriamo sulle righe del file conll-u
39          for line in lineslist:
40              if line[0].isdigit():  # se la riga inizia con un numero
        significa che contiene una parola
41                  splitted_line = line.strip().split('\t')
42                  if '-' not in splitted_line[0]:  # se l'id della
       parola non contiene un trattino la memorizziamo
43                      tmp_frase.append(Token(forma=splitted_line[1],
44                                             lemma=splitted_line[2],
45                                             pos=splitted_line[3]))
46              if line == '\n':  # se la riga è vuota significa che la
       frase è finita
47                  sentenceslist.append(Sentence(tmp_frase))
48                  tmp_frase = []
49          self.sentenceslist = sentenceslist
50
51      def get_word_ngrams(self, t_type: str, n_ngram: int) -> dict:
52          # estrae gli ngrammi di forma, lemma o pos del documento
53          wordcount = 0
54          featuredict = {}
55          if t_type == 'forma':
56              t_label = 'f'
```

```
57          if t_type == 'lemma':
58              t_label = 'l'
59          if t_type == 'pos':
60              t_label = 'p'
61          for sentence in self.sentenceslist:
62              word_list = sentence.get_wordlist(t_type)
63              wordcount += len(word_list)
64
65              for i in range(0, len(word_list) - n_ngram + 1):
66                  ngramma = word_list[i: i + n_ngram] # usiamo il list
     slicing per selezionare l'ngramma
67                  nomefeature = f'{t_label.upper()}_{n_ngram}_' + '_'.
    join(ngramma)
68                  featuredict[nomefeature] = featuredict.get(
    nomefeature, 0) + 1 # se la feature è presente nel dizionario
    allora il conteggio aumenta di 1, se non è presente allora viene
     aggiunta
69
70          return {featurename: float(value)/float(wordcount) for
    featurename, value in featuredict.items()}  # normalizzo
    dividendo per il numero di token totale del documento e
    restituisco il dizionario di feature
71
72      def get_char_ngrams(self, n_ngram: int) -> dict:
73          # estrae gli ngrammi di carattere del documento
74          featuredict = {}
75          charcount = 0
76          for sentence in self.sentenceslist:
77              word_list = sentence.get_wordlist('forma')
78              char_list = ' '.join(word_list)
79              charcount += len(char_list)
80
81              for i in range(0, len(char_list) - n_ngram + 1):
82                  ngramma = char_list[i: i + n_ngram] # usiamo il list
     slicing per prendere l'ngramma, ricorda che la fine è esclusa
83                  nomefeature = f'{n_ngram}_' + '_'.join(ngramma)
```

```
84                  featuredict[nomefeature] = featuredict.get(
    nomefeature, 0) + 1
85
86          return {featurename: float(value)/float(charcount) for
    featurename, value in featuredict.items()}
87
88
89      def get_char_pref_suff_ngrams(self, n_ngram: int) -> dict:
90          # estrae gli ngrammi di prefissi e suffissi del documento
91          featuredict = {}
92          charcount = 0
93          for sentence in self.sentenceslist:
94              word_list = sentence.get_wordlist('forma')
95              char_list = ' ' + ' '.join(word_list)  # metto anche uno
     spazio prima, così sarà preso il prefisso della prima parola.
96              charcount += len(char_list)
97
98              for i in range(1, len(char_list) - n_ngram + 1):  # il
    range comincia da 1 perchè devo considerare lo spazio che ho
    aggiunto prima
99                  ngramma = char_list[i: i + n_ngram] # usiamo il list
     slicing per prendere l'ngramma, la fine è esclusa
100                 flag = ' ' not in ngramma
101                 try:
102                     if flag and char_list[i-1] == ' ':
103                         nomefeature = f'p{n_ngram}_' + ''.join(
    ngramma)
104                         featuredict[nomefeature] = featuredict.get(
    nomefeature, 0) + 1
105
106                     elif flag and char_list[i + n_ngram] == ' ':
107                         nomefeature = f's{n_ngram}_' + ''.join(
    ngramma)
108                         featuredict[nomefeature] = featuredict.get(
    nomefeature, 0) + 1
109                 except IndexError:
```

```
110                     pass
111
112         return {featurename: float(value)/float(charcount) for
    featurename, value in featuredict.items()}
113
114
115 class Sentence:
116     __slots__ = 'tokenslist'
117     # tokenslist è la lista dei token della frase
118
119     def __init__(self, tokenslist: list):
120         self.tokenslist = tokenslist
121
122     def __repr__(self):
123         return str(self.tokenslist)
124
125     def get_wordlist(self, t_type: str):  # mi serve per ottenere la
    lista delle parole, o dei lemmi, o dei pos
126         return [getattr(token, t_type) for token in self.tokenslist]
     # getattr mi permette di prendere l'attributo dandogli il nome
    come una stringa
```

Code 6.4: Codice utilizzato per gli esperimenti lessicali out-domain e cross-time

analisi_statistiche.py

Questo script serve a calcolare il Mann Whitney U test, il rank-biserial, e estrarre i pesi delle feature linguistiche.

```
1 from class_ling_classes import ClassificazioneLinguistica
2 import json
3 import csv
4
5
6 def main():
```

```
7     with open('../4-provaclassificazione/corpusdivisiperprove/
      HP2011features_samples.json', 'r', encoding='utf-8') as f:
8         HP2011corpus = json.load(f)
9     with open('../4-provaclassificazione/corpusdivisiperprove/
      HPTOTfeatures_samples.json', 'r', encoding='utf-8') as f:
10        HPTOTcorpus = json.load(f)
11    with open('../4-provaclassificazione/corpusdivisiperprove/
      LORfeatures_samples.json', 'r', encoding='utf-8') as f:
12        LORcorpus = json.load(f)
13
14    HPTOTdataset = HPTOTcorpus['ins<=1, succ>=7']
15    HP2011dataset = HP2011corpus['ins<=1, succ>=7']
16    LORdataset = LORcorpus['ins<=1, succ>=7']
17
18    # prendo i nomi ordinati delle feature
19
20    with open('../3-pre_prove_classificazione/featuresvectorsHP.csv'
      , 'r', encoding='utf-8') as f:
21        reader = csv.reader(f)
22        intestazioneHP = reader.__next__()[0].split('\t')[1:]
23
24    with open('../3-pre_prove_classificazione/featuresvectorsLOR.csv
      ', 'r', encoding='utf-8') as f:
25        reader = csv.reader(f)
26        intestazioneLOR = reader.__next__()[0].split('\t')[1:]
27
28    # ricavo e stampo il coef per tutti e tre i corpora
29    LORdatasetobj = ClassificazioneLinguistica(LORdataset[0],
      LORdataset[1])
30    LORcoef = LORdatasetobj.get_coef()[0]
31    coef_csv('coef_results/LORcoef.csv', LORcoef, intestazioneLOR)
32
33    HPTOTdatasetobj = ClassificazioneLinguistica(HPTOTdataset[0],
      HPTOTdataset[1])
34    HPTOTcoef = HPTOTdatasetobj.get_coef()[0]
35    coef_csv('coef_results/HPTOTcoef.csv', HPTOTcoef, intestazioneHP
```

```
    )
36
37      HP2011datasetobj = ClassificazioneLinguistica(HP2011dataset[0],
    HP2011dataset[1])
38      HP2011coef = HP2011datasetobj.get_coef()[0]
39      coef_csv('coef_results/HP2011coef.csv', HP2011coef,
    intestazioneHP)
40
41      # calcolo il Mann Whitney U test per tutti e tre i corpora
42      HP2011_mwu_res = HP2011datasetobj.mann_whitney_u(intestazioneHP)
43      HPTOT_mwu_res = HPTOTdatasetobj.mann_whitney_u(intestazioneHP)
44      LOR_mwu_res = LORdatasetobj.mann_whitney_u(intestazioneHP)
45
46      # stampo i p-value e il rank-biserial
47      pvalue_csv('risultati_statistici/HP2011_pvalue.csv',
    HP2011_mwu_res)
48      pvalue_csv('risultati_statistici/HPTOT_pvalue.csv',
    HPTOT_mwu_res)
49      pvalue_csv('risultati_statistici/LOR_pvalue.csv', LOR_mwu_res)
50
51      pvalue_csv_total('risultati_statistici/pvalue_total.csv', [
    HPTOT_mwu_res, HP2011_mwu_res, LOR_mwu_res])
52
53      rankbiserial_csv('risultati_statistici/HP2011_rankbiserial.csv',
        HP2011_mwu_res)
54      rankbiserial_csv('risultati_statistici/HPTOT_rankbiserial.csv',
    HPTOT_mwu_res)
55      rankbiserial_csv('risultati_statistici/LOR_rankbiserial.csv',
    LOR_mwu_res)
56
57
58  def rankbiserial_csv(filepath: str, mwu_res: list) -> None:
59      with open(filepath, 'w', encoding='utf-8', newline='') as f:
60          writer = csv.writer(f, dialect='excel', delimiter='\t')
61          writer.writerow(['Nome feature', 'rank_biserial', 'p-value'
    ])
```

```
62          output = [(feature_dict['feature name'], feature_dict['
    rank_biserial'], feature_dict['p-value'])
63                  for feature_dict in mwu_res if feature_dict['p-
    value'] < 0.05]
64          output.sort(key=lambda x: x[1], reverse=True)
65          writer.writerows(output)
66
67
68  def pvalue_csv(filepath: str, mwu_res: list) -> None:
69      with open(filepath, 'w', encoding='utf-8', newline='') as f:
70          writer = csv.writer(f, dialect='excel', delimiter='\t')
71          writer.writerow(['Nome feature', 'p-value'])
72          output = [(feature_dict['feature name'], feature_dict['p-
    value']) for feature_dict in mwu_res]
73          output.sort(key=lambda x: x[1], reverse=False)
74          writer.writerows(output)
75
76
77
78  def pvalue_csv_total(filepath: str, mwu_ress: list) -> None:
79      total_list = [(feature_dict_1['feature name'], feature_dict_1['p
    -value'], feature_dict_2['p-value'], feature_dict_3['p-value'])
80                  for feature_dict_1,feature_dict_2, feature_dict_3
    in zip(mwu_ress[0], mwu_ress[1], mwu_ress[2])]
81      with open(filepath, 'w', encoding='utf-8', newline='') as f:
82          writer = csv.writer(f, dialect='excel', delimiter='\t')
83          writer.writerow(['Nome feature', 'p-value HPTOT', 'p-value
    HP2011', 'p-value LOTR'])
84          writer.writerows(total_list)
85
86
87  def coef_csv(filepath: str, coef_list: list, header: list) -> None:
88      with open(filepath, 'w', encoding='utf-8', newline='') as f:
89          writer = csv.writer(f, dialect='excel', delimiter='\t')
90          writer.writerow(['Nome feature', 'coefOLD'])
91          output = [(feature, value) for feature, value in zip(header,
```

```
    coef_list)]
92          output.sort(key=lambda x: x[1], reverse=True)
93          writer.writerows(output)
94
95
96  if __name__ == '__main__':
97      main()
```

class_ling_classes.py

In questo file è definita la classe *ClassificazioneLinguistica*, utilizzata per gestire gli esperimenti di classificazione a modello linguistico e le analisi statistiche delle feature.

```
1   from sklearn import svm
2   from sklearn.dummy import DummyClassifier
3   import scipy.stats as sp
4   from multiprocessing import Pool
5
6
7   class ClassificazioneLinguistica:
8       @staticmethod
9       def crossvalida(arg):
10          fold = arg[0]
11          selffolds = arg[1]
12          i = selffolds.index(fold)
13          test = selffolds[i][0]
14          training = [[], []]
15          for fold in selffolds:
16              if selffolds.index(fold) == i:
17                  continue
18              training[0] += fold[0]
19              training[1] += fold[1]
20          classificatore = svm.LinearSVC()
21          classificatore.fit(training[0], training[1])
```

```
22          risclassificazione = classificatore.predict(test)
23          return risclassificazione
24
25
26
27      featuresvectorslist = [] # lista dei vettori di feature
28      resultslist = [] # lista delle etichette di classe associate
29      classes = {} # insieme delle classi
30      folds = [] # sottoliste di vettori di feature per la n-fold
    crossvalidation
31      __tmpcrossvalidationresult = [] # risultati della n-fold
    crossvalidation
32
33      def __init__(self, features: list, results: list) -> None:
34          if len(features) != len(results):
35              raise Exception
36          self.featuresvectorslist = features
37          self.resultslist = results
38          self.classes = set(results)
39
40      def corpusbalance(self, ratio=False) -> str | float:
41          """Restituisce il bilanciamento del corpus"""
42          result = ''
43          values = [(class_, self.resultslist.count(class_)) for
    class_ in self.classes]
44          total = len(self.resultslist)
45          for couple in values:
46              result += f'class {couple[0]}: {couple[1] / total}\t|\t'
47          if ratio:
48              return min(values[0][1], values[1][1]) / max(values
    [0][1], values[1][1])
49          return result
50
51      def preparecrossvalidation(self, nfold) -> None:
52          """Prepara una n-fold crossvalidation"""
53          self.folds = [([], []) for n in range(nfold)]
```

```
54          c = 0
55          for i in range(len(self.featuresvectorslist)):
56              if c >= nfold: c = 0
57              self.folds[c][0].append(self.featuresvectorslist[i])
58              self.folds[c][1].append(self.resultslist[i])
59              c += 1
60
61      def crossvalidation(self) -> None:
62          """Effettua la n-fold crossvalidation"""
63          result = []
64          for i in range(len(self.folds)):
65              test = self.folds[i][0]
66              training = [[], []]
67              for fold in self.folds:
68                  if self.folds.index(fold) == i:
69                      continue
70                  training[0] += fold[0]
71                  training[1] += fold[1]
72              classificatore = svm.LinearSVC()
73              classificatore.fit(training[0], training[1])
74              risclassificazione = classificatore.predict(test)
75              result.append(list(risclassificazione))
76          self.__tmpcrossvalidationresult = result
77
78      def multiprocess_crossvalidation(self) -> None:
79          """Effettua la n-fold crossvalidation in multiprocessing"""
80          input = [(fold, self.folds) for fold in self.folds]
81
82          with Pool(4) as p:
83              self.__tmpcrossvalidationresult = p.map(self.crossvalida
    , input)
84
85
86      def accuracy(self):
87          """Restituisce l'accuracy della n-fold crossvalidation"""
88          numesatti = 0
```

```
89          for fold, foldresult in zip(self.folds, self.
    __tmpcrossvalidationresult):
90              for i in range(len(fold[1])):
91                  if foldresult[i] == fold[1][i]:
92                      numesatti += 1
93          return numesatti / len(self.resultslist)
94
95      def mostfrequencybaseline(self):
96          """Calcola la baseline_mf"""
97          return max([self.resultslist.count(class_) for class_ in
    self.classes]) / len(self.resultslist)
98
99      def dummymostfrequencybaseline(self):
100         """Calcola la baselin_mf con il DummyClassifier"""
101         result = []
102         for i in range(len(self.folds)):
103             test = self.folds[i][0]
104             training = [[], []]
105             for fold in self.folds:
106                 if self.folds.index(fold) == i:
107                     continue
108                 training[0] += fold[0]
109                 training[1] += fold[1]
110             classificatore = DummyClassifier(strategy='most_frequent
    ')
111             classificatore.fit(training[0], training[1])
112             risclassificazione = classificatore.predict(test)
113             result.append(list(risclassificazione))
114         dummycrossvalidationresult = result
115
116         numesatti = 0
117         for fold, foldresult in zip(self.folds,
    dummycrossvalidationresult):
118             for i in range(len(fold[1])):
119                 if foldresult[i] == fold[1][i]:
120                     numesatti += 1
```

```
121         return numesatti / len(self.resultslist)
122
123
124
125     def get_coef(self) -> list:
126         """Restituisce la lista dei pesi delle feature"""
127         cls = svm.LinearSVC()
128         features = self.featuresvectorslist
129         results = self.resultslist
130         cls.fit(X=features, y=results)
131         return cls.coef_
132
133     def mann_whitney_u(self, featureslabels: list):
134         """Calcola il Mann Whitney U test e il rank-biserial"""
135         if len(self.featuresvectorslist[0]) != len(featureslabels):
136             raise Exception
137         response = []
138         for i in range(len(featureslabels)):
139             element = {
140                 'feature name': featureslabels[i],
141                 'class1': [],
142                 'class2': [],
143                 'p-value': None,
144                 'U': None
145             }
146             for vector, result in zip(self.featuresvectorslist, self
    .resultslist):
147                 if result == 0:
148                     element['class1'].append(vector[i])
149                 else:
150                     element['class2'].append(vector[i])
151             element['U'], element['p-value'] = sp.mannwhitneyu(x=
    element['class1'], y=element['class2'])
152
153             # qui calcolo il rank biserial
154             lenX = len(element['class1'])
```

```
155             lenY = len(element['class2'])
156             rank_biserial = 1 - ((2 * element['U']) / (lenX * lenY))
157             # uso il valore assoluto
158             element['rank_biserial'] = abs(rank_biserial)
159 
160             response.append(element)
161         return response
162 
163     def cross_set_validation(self, cross_set: list) -> dict:
164         """Valuta il classificatore con un corpus diverso da quello
    di addestramento (utilizzato per valutazione out-domain e cross-
    time)"""
165         # addestro il classificatore ed effettuo la predizione
166         classificatore = svm.LinearSVC()
167         classificatore.fit(self.featuresvectorslist, self.
    resultslist)
168         risclassificazione = classificatore.predict(cross_set[0])
169 
170         # se dovesse esserci una incongruenza tra il numero di
    risultati reali e quelli predetti genero una eccezione
171         if len(risclassificazione) != len(cross_set[1]):
172             raise Exception
173 
174         # calcolo la baseline per most frequency con il dummy
    classifier
175         dummy_mf = DummyClassifier(strategy='most_frequent')
176         dummy_mf.fit(self.featuresvectorslist, self.resultslist)
177         baseline_mf = dummy_mf.score(X=cross_set[0], y=cross_set[1])
178 
179         # calcolo la baseline_u
180         dummy_u = DummyClassifier(strategy='uniform')
181         dummy_u.fit(self.featuresvectorslist, self.resultslist)
182         baseline_u = dummy_u.score(X=cross_set[0], y=cross_set[1])
183 
184         # calcolo manualmente l'accuracy
185         esatti = 0
```

```
186         for i in range(len(risclassificazione)):
187             if risclassificazione[i] == cross_set[1][i]:
188                 esatti += 1
189         accuracy = esatti / len(risclassificazione)
190 
191         # restituisco i risultati con un dizionario
192         return {'accuracy': accuracy, 'baseline_mf': baseline_mf, '
    baseline_u': baseline_u}
```

Appendice B: tabelle

In questa appendice saranno mostrati per intero le tabelle riportate in forma abbreviata nel corpo della relazione.

Rank-biserial HPTOT

Nome feature	Rank-biserial	Direzione
n_sentences	0.32617615735022243	succ
n_tokens	0.257471533190915	succ
upos_dist_PUNCT	0.23685668520022474	succ
n_prepositional_chains	0.23008786639102874	succ
avg_max_depth	0.22278988978109382	ins
dep_dist_punct	0.22274976334374041	succ
avg_max_links_len	0.21320861381799605	ins
dep_dist_root	0.21273132879328882	succ
tokens_per_sent	0.21273132879328882	ins
verbal_head_per_sent	0.21234503362321555	ins
verbal_root_perc	0.2034849651469013	ins
dep_dist_conj	0.17907487512933673	ins
upos_dist_INTJ	0.17659174045157833	succ
dep_dist_discourse	0.16382340537903928	succ
aux_num_pers_dist_Sing+2	0.1490058359352162	succ
ttr_lemma_chunks_200	0.14172289345825262	succ
ttr_form_chunks_200	0.13964809456796645	succ
aux_tense_dist_Pres	0.13744110509743168	succ
dep_dist_obl	0.1360879859085491	ins
verbs_mood_dist_Imp	0.13280077598084117	succ
verbs_num_pers_dist_Plur+2	0.13145228449523672	succ

upos_dist_DET	0.13003990827419898	ins
subordinate_proposition_dist	0.12657880531259402	ins
principal_proposition_dist	0.12657500398490118	succ
verbs_tense_dist_Pres	0.1265066509186239	succ
verbs_tense_dist_Imp	0.12451575866366849	ins
aux_tense_dist_Imp	0.12430691584519238	ins
verbs_mood_dist_Cnd	0.12314402241338362	succ
verbs_num_pers_dist_Sing+2	0.12061700719204116	succ
dep_dist_cc	0.11786244355057884	ins
upos_dist_CCONJ	0.11591300583659625	ins
upos_dist_ADP	0.11416407045036037	ins
dep_dist_det	0.11334404615161264	ins
aux_num_pers_dist_Plur+2	0.10912964871978015	succ
avg_links_len	0.10771290451350524	ins
dep_dist_case	0.10767246523396601	ins
lexical_density	0.1074574954307923	succ
aux_tense_dist_Fut	0.10259964655207887	succ
dep_dist_compound	0.10174444816680739	succ
verb_edges_dist_0	0.10090129125435143	succ
avg_verb_edges	0.09655715285561284	ins
verbs_form_dist_Ger	0.09634382399824448	succ
aux_num_pers_dist_Sing+1	0.09565707637337184	succ
verb_edges_dist_1	0.09519355050216127	succ
dep_dist_nsubj	0.0917610637465418	ins
upos_dist_X	0.09041042368441543	succ
verbs_mood_dist_Ind	0.08886563848386708	ins
ttr_lemma_chunks_100	0.0871594562288095	succ
verbs_num_pers_dist_Sing+1	0.08503303247516747	succ

dep_dist_csubj	0.08467706528907815	succ
upos_dist_SYM	0.08447550638111878	succ
verbs_num_pers_dist_Plur+3	0.08348109322312292	ins
dep_dist_acl:relcl	0.08188837233593915	ins
char_per_tok	0.081555921437937	succ
aux_num_pers_dist_Plur+1	0.07962646430507259	succ
dep_dist_flat:foreign	0.07949143453441987	succ
dep_dist_iobj	0.07693419167470039	succ
ttr_form_chunks_100	0.07345101267960008	succ
verbs_tense_dist_Fut	0.07123030537434782	succ
dep_dist_det:poss	0.06639805643073315	ins
prep_dist_3	0.06506625182927095	succ
upos_dist_NOUN	0.06505653601345984	ins
dep_dist_parataxis	0.06453981516528118	succ
subj_post	0.06417988727906465	succ
dep_dist_expl:pass	0.06289242953312402	succ
subj_pre	0.061907006160960565	ins
dep_dist_appos	0.060413515273534	succ
dep_dist_amod	0.05844294595529642	ins
verbs_num_pers_dist_Plur+1	0.0574780993352092	succ
max_links_len	0.05490170226995694	succ
verb_edges_dist_4	0.05377716447067593	ins
dep_dist_expl:impers	0.05357826767263807	succ
aux_num_pers_dist_Plur+3	0.05078838237758321	ins
subordinate_dist_4	0.05040025147317384	succ
aux_mood_dist_Sub	0.04845740115314334	succ
aux_num_pers_dist_Sing+3	0.04756655056404502	ins
dep_dist_xcomp	0.04735681644047318	ins

upos_dist_ADJ	0.04722264255881958	ins
upos_dist_AUX	0.04715002775559474	ins
avg_token_per_clause	0.04671801040491863	succ
dep_dist_vocative	0.044054867513931506	succ
upos_dist_PROPN	0.04096016177883999	succ
verb_edges_dist_5	0.04069274663940803	ins
dep_dist_aux:pass	0.039717387341337895	ins
verbs_mood_dist_Sub	0.03961404907438493	succ
dep_dist_nmod	0.036817121878806036	ins
obj_pre	0.03667251795672832	succ
dep_dist_ccomp	0.03641106452687648	succ
verbs_form_dist_Part	0.03640041608719047	ins
dep_dist_fixed	0.03620473034857019	ins
upos_dist_VERB	0.03536479039821461	ins
obj_post	0.03499680535004046	ins
aux_tense_dist_Past	0.03399676693538112	ins
subordinate_dist_5	0.03335427762578891	succ
dep_dist_dislocated	0.031997215444827476	succ
upos_dist_SCONJ	0.03197776610515701	succ
verbs_tense_dist_Past	0.030541779153108184	ins
dep_dist_aux	0.029646135585614575	ins
upos_dist_ADV	0.028843730794883782	succ
aux_form_dist_Ger	0.027350340253064043	succ
aux_form_dist_Fin	0.027277548369356697	succ
aux_mood_dist_Imp	0.027264001712439145	succ
dep_dist_cop	0.027176459024532473	ins
verb_edges_dist_3	0.02662567969687024	ins
dep_dist_nummod	0.024934395813080767	succ

avg_subordinate_chain_len	0.02404915277259745	ins
subordinate_dist_1	0.02358532586456652	succ
dep_dist_advcl	0.02335438340186391	ins
verb_edges_dist_2	0.022495365980851223	succ
dep_dist_obj	0.02217190486868148	succ
dep_dist_advmod	0.021785515255684285	succ
aux_mood_dist_Cnd	0.021211957475279042	succ
dep_dist_flat	0.020217762716544962	succ
prep_dist_4	0.019335606779141368	succ
aux_mood_dist_Ind	0.018497053959232623	ins
verbs_num_pers_dist_Sing+3	0.017996966871051345	ins
dep_dist_flat:name	0.016536312607778547	succ
upos_dist_NUM	0.01592052703495994	succ
verb_edges_dist_6	0.01472510391850057	ins
verbs_num_pers_dist_Plur+	0.008346605909015148	succ
upos_dist_PART	0.005768927961605552	ins
prep_dist_5	0.0034936562569866103	succ
verbs_num_pers_dist_+3	0.0023964183654371363	succ
dep_dist_orphan	0.0017058162887269024	succ

Rank-biserial HP2011

Nome feature	**Rank-biserial**	**Direzione**
n_sentences	0.2499288054145401	succ
upos_dist_PUNCT	0.2151108162906854	succ
dep_dist_punct	0.20464791222233403	succ
n_tokens	0.18784513742402353	succ
n_prepositional_chains	0.1860887201060899	succ
dep_dist_root	0.17643496168566042	succ

tokens_per_sent	0.17643496168566042	ins
verbal_head_per_sent	0.17604445725283835	ins
avg_max_links_len	0.17546211112266907	ins
avg_max_depth	0.17213856029485064	ins
dep_dist_conj	0.1677270538856328	ins
ttr_lemma_chunks_200	0.16281732748315625	succ
dep_dist_cc	0.1496444818085756	ins
dep_dist_discourse	0.1491007882277977	succ
upos_dist_CCONJ	0.1480401167989096	ins
ttr_form_chunks_200	0.14339697930326512	succ
upos_dist_INTJ	0.14170848815668835	succ
verbal_root_perc	0.12564380652580076	ins
principal_proposition_dist	0.1216830570528682	succ
subordinate_proposition_dist	0.12159154145798556	ins
ttr_lemma_chunks_100	0.12085714301954853	succ
avg_links_len	0.11540514408590408	ins
verbs_num_pers_dist_Plur+2	0.11204493017814843	succ
verbs_form_dist_Ger	0.1099636609193736	succ
char_per_tok	0.10818564364736827	succ
verbs_mood_dist_Imp	0.10798840196150317	succ
verb_edges_dist_1	0.10749785563613856	succ
aux_num_pers_dist_Sing+2	0.10715197216106975	succ
dep_dist_obl	0.10649885776032342	ins
aux_tense_dist_Pres	0.10311505442904212	succ
avg_verb_edges	0.10116992172290318	ins
lexical_density	0.09850119055536322	succ
dep_dist_acl:relcl	0.09394530550577151	ins
verbs_tense_dist_Imp	0.09189075197966412	ins

aux_tense_dist_Imp	0.09145562658908868	ins
verbs_tense_dist_Pres	0.090767270158015	succ
upos_dist_PRON	0.08881417957406024	ins
verb_edges_dist_0	0.08648280558392918	succ
aux_num_pers_dist_Plur+2	0.08637537423341479	succ
aux_mood_dist_Sub	0.0792280631150658	succ
ttr_form_chunks_100	0.07704817301335842	succ
verbs_mood_dist_Cnd	0.07676765782034867	succ
upos_dist_X	0.07491773543807834	succ
prep_dist_3	0.0728975713125326	succ
upos_dist_ADV	0.07238229872395419	ins
aux_num_pers_dist_Plur+3	0.07123409063909159	ins
upos_dist_PROPN	0.07034053464433687	succ
dep_dist_compound	0.06654007956740982	succ
dep_dist_advmod	0.06632322739692698	ins
dep_dist_fixed	0.06556268164566648	ins
dep_dist_iobj	0.06480952535237772	succ
verbs_num_pers_dist_Sing+2	0.06190234206028322	succ
dep_dist_det:poss	0.06152121655488685	ins
dep_dist_appos	0.06002087237667142	succ
dep_dist_csubj	0.06001831448737338	succ
dep_dist_flat:foreign	0.059031537638204234	succ
upos_dist_DET	0.05855349654940745	ins
dep_dist_parataxis	0.05830083392875307	succ
dep_dist_nsubj	0.0576960352147462	ins
verbs_mood_dist_Ind	0.05624599619272397	ins
verb_edges_dist_5	0.056157606906983126	ins
upos_dist_SYM	0.05492243058598967	succ

upos_dist_ADP	0.05429290560876909	ins
dep_dist_xcomp	0.05182766874537803	ins
aux_tense_dist_Fut	0.05106342826513144	succ
verb_edges_dist_4	0.049226295329350966	ins
avg_token_per_clause	0.048458928539962476	succ
avg_prepositional_chain_len	0.048282434178403166	succ
dep_dist_expl:impers	0.04782798251313192	succ
aux_num_pers_dist_Plur+1	0.047053226265771464	succ
subj_post	0.04631882782733454	succ
subj_pre	0.046109649324745705	ins
dep_dist_expl:pass	0.04602296529853689	succ
dep_dist_obl:agent	0.045824870982905885	succ
dep_dist_case	0.045521903206058534	ins
verbs_tense_dist_Fut	0.04470593652000865	succ
dep_dist_expl	0.039782283831354404	ins
dep_dist_nummod	0.0392337586819026	succ
verbs_num_pers_dist_Plur+3	0.03861986525039174	ins
dep_dist_flat	0.034993062435804134	succ
subordinate_dist_4	0.032189899975160086	succ
aux_mood_dist_Imp	0.030808355544338695	succ
dep_dist_vocative	0.027659309608625837	succ
dep_dist_dislocated	0.019815115761543378	succ
verbs_num_pers_dist_Plur+	0.010270493951160198	succ

Rank-biserial LOTR

Nome feature	**Rank-biserial**	**Direzione**
verbal_root_perc	0.28620127741669643	ins
verbal_head_per_sent	0.25099271385491395	ins

n_sentences	0.22215718593717804	succ
avg_max_depth	0.21710464702466803	ins
upos_dist_PUNCT	0.21460881454981362	succ
dep_dist_punct	0.20667646893554847	succ
dep_dist_root	0.20357657944520402	succ
tokens_per_sent	0.20357657944520402	ins
upos_dist_INTJ	0.19244132686508453	succ
avg_max_links_len	0.19151885219801823	ins
verbs_mood_dist_Imp	0.189809698627058	succ
dep_dist_conj	0.17802824551873986	ins
dep_dist_discourse	0.17633313978534904	succ
verbs_num_pers_dist_Sing+2	0.1616859278128453	succ
obj_pre	0.15889040813650757	succ
obj_post	0.15889040813650745	ins
n_tokens	0.15748562437955382	succ
n_prepositional_chains	0.15352413418494448	succ
dep_dist_compound	0.15070520144599076	succ
dep_dist_cc	0.14799865140759327	ins
upos_dist_CCONJ	0.14601790631028866	ins
avg_token_per_clause	0.1459289366723482	succ
verbs_tense_dist_Pres	0.14545131019498403	succ
dep_dist_obl	0.142445072955103	ins
dep_dist_advcl	0.14134934162467916	ins
upos_dist_VERB	0.13985558822978517	ins
subordinate_proposition_dist	0.13451740995336126	ins
principal_proposition_dist	0.13451740995336114	succ
aux_tense_dist_Pres	0.13171720766450012	succ
dep_dist_appos	0.12653823821386423	succ

aux_num_pers_dist_Plur+3	0.12442637996591066	ins
dep_dist_iobj	0.12412669276442712	succ
upos_dist_DET	0.1240892318642417	ins
dep_dist_flat:name	0.12330255296034764	succ
verbs_mood_dist_Ind	0.12237539568075828	ins
upos_dist_ADJ	0.12162149506452646	ins
upos_dist_ADP	0.12131712525051985	ins
aux_num_pers_dist_Sing+1	0.11915844087733429	succ
verbs_tense_dist_Imp	0.11634419075090374	ins
dep_dist_amod	0.11574949896045994	ins
dep_dist_det	0.11532338122085073	ins
upos_dist_SYM	0.1150236940193673	succ
aux_tense_dist_Imp	0.1148410721309634	ins
verb_edges_dist_0	0.11419955421528782	succ
upos_dist_PROPN	0.11112776040008243	succ
verbs_num_pers_dist_Sing+1	0.10999456816947306	succ
verbs_form_dist_Part	0.1050544119575193	succ
aux_num_pers_dist_Plur+2	0.10179999625391001	succ
dep_dist_case	0.10172507445353918	ins
aux_num_pers_dist_Sing+2	0.10030156024649273	succ
avg_subordinate_chain_len	0.09892487216467805	ins
subordinate_dist_1	0.09876098072636685	succ
dep_dist_aux:pass	0.09452321639288996	ins
verb_edges_dist_6	0.0923036580569031	succ
verbs_num_pers_dist_Plur+2	0.0909550656502276	succ
dep_dist_aux	0.0852844218846579	succ
avg_links_len	0.0842214688418963	ins
verbs_num_pers_dist_Plur+1	0.08398733821573734	succ

verbs_num_pers_dist_Plur+3	0.08327089849969105	ins
dep_dist_nsubj:pass	0.08099514881342595	ins
dep_dist_det:poss	0.08090617917548548	ins
dep_dist_expl:pass	0.06114555432767055	succ
dep_dist_flat	0.04035007211223285	succ
dep_dist_orphan	0.009852216748768461	succ

Pesi classificazione modello linguistico HPTOT

Nome feature	**Coef**	**Direzione**
avg_token_per_clause	4.706262360928698	ins
n_tokens	3.3338805789123516	succ
char_per_tok	2.875414748423346	succ
n_prepositional_chains	2.4438614911154364	succ
avg_max_depth	1.8048199172816854	ins
n_sentences	1.8046483612407866	succ
max_links_len	1.664724549393337	succ
upos_dist_SYM	1.5938334517982125	ins
upos_dist_VERB	1.3753783997100657	ins
upos_dist_SCONJ	1.2996142207197003	succ
dep_dist_amod	1.2989012432352463	ins
dep_dist_aux:pass	1.2616274631473485	ins
upos_dist_PART	1.2249080407788506	ins
dep_dist_obj	1.193571783211394	succ
dep_dist_dislocated	1.1906318368223479	succ
upos_dist_PUNCT	1.066059932927527	succ
avg_max_links_len	1.0646464245093192	ins
subordinate_proposition_dist	1.0561194220560974	ins
upos_dist_CCONJ	1.0147160605809347	succ

dep_dist_case	0.974314620155371	succ
tokens_per_sent	0.9730268517867137	ins
dep_dist_cc	0.9626232881467991	ins
verbs_form_dist_Fin	0.9338864764501476	ins
dep_dist_acl	0.9056813302387807	succ
ttr_lemma_chunks_200	0.8934797256883672	succ
upos_dist_ADJ	0.8919242103299239	succ
verbs_num_pers_dist_Plur+1	0.8675701201701032	ins
upos_dist_NUM	0.8613627192291904	ins
dep_dist_root	0.8249965674250243	ins
dep_dist_conj	0.8243339499681716	ins
verbal_head_per_sent	0.8093974828015863	ins
verbal_root_perc	0.8083809144076597	ins
verbs_num_pers_dist_+3	0.7813095109499076	succ
dep_dist_aux	0.7445630133416812	ins
dep_dist_nmod	0.7386170445275724	ins
verbs_tense_dist_Fut	0.7313225576655473	succ
aux_mood_dist_Imp	0.6666032753910787	ins
verbs_num_pers_dist_Plur+2	0.6639448595872911	succ
upos_dist_AUX	0.6470699785558199	ins
dep_dist_dep	0.6463180988408261	succ
dep_dist_obl:agent	0.641136109908121	ins
dep_dist_iobj	0.6345661488883536	succ
verbs_tense_dist_Past	0.6228805724483066	succ
verbs_tense_dist_Pres	0.6161006756892298	succ
aux_tense_dist_Past	0.6008204604722869	ins
upos_dist_NOUN	0.5919178176226666	succ
verbs_num_pers_dist_Plur+	0.5882193005453825	succ

lexical_density	0.5665157205254172	ins
verbs_form_dist_Ger	0.5629471969546008	succ
dep_dist_acl:relcl	0.5487241547445733	succ
upos_dist_ADP	0.548182940853045	ins
dep_dist_ccomp	0.5362124552641312	succ
aux_form_dist_Part	0.5242836504276454	succ
dep_dist_compound	0.5122235045793039	succ
ttr_form_chunks_200	0.5037247591732503	ins
dep_dist_flat	0.49088779092617046	ins
subj_pre	0.4638498425344773	succ
aux_num_pers_dist_Plur+2	0.4371372710960777	succ
aux_num_pers_dist_Plur+3	0.4332278934886269	ins
verbs_num_pers_dist_+	0.4323014219944128	succ
aux_num_pers_dist_Plur+1	0.42252290225264044	ins
prep_dist_2	0.4081732753248112	ins
dep_dist_cop	0.4017873204633444	ins
subj_post	0.40039823392188056	succ
subordinate_dist_3	0.40027340488232066	ins
dep_dist_nsubj:pass	0.37754689653956575	succ
avg_prepositional_chain_len	0.3746271941999527	succ
dep_dist_mark	0.37120485522183905	succ
avg_verb_edges	0.36021632174572443	succ
verbs_mood_dist_Imp	0.35259472307642165	succ
prep_dist_4	0.34732046194137345	ins
subordinate_dist_5	0.3367570810578993	succ
verbs_form_dist_Part	0.3360502126168436	succ
upos_dist_ADV	0.3284458159151846	succ
dep_dist_expl	0.32317637129898125	succ

subordinate_pre	0.312049049692215	ins
aux_tense_dist_Imp	0.2874923465543451	ins
dep_dist_appos	0.2851403260756518	ins
dep_dist_obl	0.2694788199721943	ins
dep_dist_orphan	0.26889361344495677	succ
verbs_num_pers_dist_Sing+3	0.23808036045817577	succ
avg_subordinate_chain_len	0.23319563287665795	succ
dep_dist_advcl	0.23203899664283703	succ
dep_dist_nummod	0.2316541352980966	succ
upos_dist_PROPN	0.229672235280227	succ
verbs_mood_dist_Sub	0.22062390065585563	ins
aux_num_pers_dist_Sing+1	0.2165259024284683	ins
dep_dist_csubj	0.21061875419310103	succ
verbs_tense_dist_Imp	0.20584971282316714	succ
dep_dist_fixed	0.20334103311183754	ins
prep_dist_3	0.20062521343616216	ins
upos_dist_INTJ	0.19918151224370628	succ
verbs_num_pers_dist_Sing+1	0.19905457706178012	succ
aux_mood_dist_Cnd	0.19577232543929302	ins
upos_dist_X	0.19044456133976637	ins
principal_proposition_dist	0.18827280012509953	succ
dep_dist_vocative	0.18533649369309502	succ
dep_dist_flat:foreign	0.18147473196199898	ins
dep_dist_flat:name	0.17566201893640726	succ
verbs_num_pers_dist_Sing+2	0.17275756568846662	ins
verb_edges_dist_2	0.16755159744404485	ins
aux_num_pers_dist_Sing+2	0.16102511280301138	ins
prep_dist_5	0.15946458658223636	ins

subordinate_post	0.15371785106590205	ins
dep_dist_expl:pass	0.13772057243769878	ins
prep_dist_1	0.13207841964424405	ins
verbs_num_pers_dist_Plur+3	0.1310838364314433	ins
dep_dist_det	0.1251736022739138	ins
ttr_lemma_chunks_100	0.12246874911621564	succ
verbs_mood_dist_Cnd	0.12220321650333839	succ
aux_form_dist_Fin	0.11577887181070656	succ
verb_edges_dist_0	0.11164568530502222	succ
verb_edges_dist_4	0.10900224259035116	ins
dep_dist_punct	0.09946057971952099	succ
aux_form_dist_Inf	0.09287891589917263	succ
obj_pre	0.0915476117842982	succ
dep_dist_det:predet	0.0842530907937512	ins
aux_num_pers_dist_Sing+3	0.08387343569669799	ins
verb_edges_dist_6	0.07504752829002159	succ
verb_edges_dist_5	0.0745546156061307	succ
aux_form_dist_Ger	0.0678413988785679	ins
subordinate_dist_1	0.06539677865646015	ins
ttr_form_chunks_100	0.059360310768969	succ
verbs_mood_dist_Ind	0.05348705942431627	ins
verb_edges_dist_1	0.04961740271743552	ins
aux_tense_dist_Pres	0.04861035758183248	succ
aux_tense_dist_Fut	0.04033552612876493	ins
upos_dist_DET	0.03361837458214702	succ
verb_edges_dist_3	0.03095918558818172	ins
dep_dist_advmod	0.026611901421625485	succ
verbs_form_dist_Inf	0.02393176415153512	ins

avg_links_len	0.023775004402598044	succ
aux_mood_dist_Sub	0.02077127405859966	ins
dep_dist_expl:impers	0.01671191393186687	ins
dep_dist_det:poss	0.016423588678291866	succ
dep_dist_xcomp	0.012668682518704729	succ
dep_dist_nsubj	0.009840803937953278	succ
obj_post	0.007615254790913561	ins
subordinate_dist_4	0.006608780851439963	ins
dep_dist_discourse	0.004875257271677718	succ
upos_dist_PRON	0.0039606358286517086	ins
aux_mood_dist_Ind	0.0031088993155906405	succ
dep_dist_parataxis	0.0011941570243117368	succ
subordinate_dist_2	0.00014867012321470067	ins

Pesi classificazione modello linguistico HP2011

Nome feature	**Coef**	**Direzione**
n_sentences	2.660827879343441	succ
n_tokens	2.375579085966855	succ
n_prepositional_chains	1.9575244911704421	succ
char_per_tok	1.8824984183683628	succ
verbs_num_pers_dist_Plur+1	1.8015304616047791	ins
dep_dist_iobj	1.6602313850178452	succ
dep_dist_obj	1.623736259663214	succ
tokens_per_sent	1.5912820922146744	ins
upos_dist_SCONJ	1.5890060625096813	succ
avg_max_depth	1.5600550422454202	ins
ttr_lemma_chunks_100	1.3695572145409192	succ
upos_dist_ADV	1.329501137289342	ins

upos_dist_PUNCT	1.1880910817135746	succ
verbal_head_per_sent	1.0770433096335454	ins
verbs_num_pers_dist_+3	1.0763505269482876	ins
dep_dist_dep	1.036439969867439	succ
dep_dist_fixed	1.034114826135655	ins
dep_dist_dislocated	1.02797768710867	succ
verbs_num_pers_dist_Plur+2	0.983819642122602	succ
dep_dist_compound	0.9780875261018823	ins
ttr_form_chunks_100	0.9536277790364731	ins
verbs_num_pers_dist_Plur+	0.9292860294082683	succ
dep_dist_aux:pass	0.9248754190738636	succ
avg_subordinate_chain_len	0.8774589474050709	ins
upos_dist_PART	0.8590437104366425	ins
avg_token_per_clause	0.8549562231385879	ins
prep_dist_5	0.8239285630916411	succ
prep_dist_4	0.7871421384376169	ins
dep_dist_flat:name	0.7825558504136122	ins
max_links_len	0.766976477723723	succ
upos_dist_ADP	0.7522523334608353	succ
dep_dist_amod	0.7463313160253584	ins
verbs_form_dist_Ger	0.7398876045634493	succ
dep_dist_flat	0.7145025116984209	ins
upos_dist_PRON	0.7081524422868567	ins
verbs_form_dist_Fin	0.6863058287576507	ins
dep_dist_expl:pass	0.6832695702359272	succ
upos_dist_NUM	0.6827496266816692	ins
avg_max_links_len	0.6650600174078042	ins
dep_dist_mark	0.6307422002498355	succ

aux_mood_dist_Imp	0.6068843211874442	succ
dep_dist_case	0.5762034651345018	succ
verbs_mood_dist_Cnd	0.5373195296013662	ins
prep_dist_3	0.5259163129668144	succ
dep_dist_conj	0.5195905306249523	ins
upos_dist_INTJ	0.5187883882481422	ins
dep_dist_discourse	0.5149987640012365	succ
dep_dist_nmod	0.5140470143613176	ins
ttr_lemma_chunks_200	0.5048099234180102	succ
dep_dist_flat:foreign	0.4849270319910382	ins
verbs_form_dist_Part	0.47851623458752607	succ
aux_mood_dist_Cnd	0.47048085519012295	ins
avg_verb_edges	0.46986292514839473	succ
dep_dist_parataxis	0.4598754476231835	ins
dep_dist_cc	0.45247360578441564	ins
dep_dist_obl	0.45229244460620077	ins
dep_dist_punct	0.4494355558403125	succ
dep_dist_advcl	0.44640572819283086	ins
upos_dist_ADJ	0.4444046078722569	succ
verbs_form_dist_Inf	0.4430823007091052	ins
aux_num_pers_dist_Plur+2	0.44261932782667635	succ
dep_dist_ccomp	0.43863160619744745	succ
dep_dist_aux	0.43332792082117877	ins
verb_edges_dist_1	0.4283660928634027	succ
dep_dist_vocative	0.4251916308534875	succ
lexical_density	0.4251435574245712	ins
dep_dist_nsubj:pass	0.4188753144690638	ins
dep_dist_advmod	0.4145308249722877	succ

verbs_mood_dist_Imp	0.4143689948320975	ins
subordinate_proposition_dist	0.4134822552129687	ins
dep_dist_det	0.4111790878165546	succ
dep_dist_orphan	0.4011567163828773	succ
verb_edges_dist_0	0.38752534331742733	ins
subordinate_dist_4	0.3854548569975137	succ
upos_dist_X	0.36649681982630894	ins
subordinate_dist_2	0.33598715957433495	succ
verbs_num_pers_dist_Sing+2	0.3341484112172557	ins
verbal_root_perc	0.33034632466646147	ins
avg_links_len	0.32948296190929205	succ
upos_dist_PROPN	0.31966133512949946	ins
aux_num_pers_dist_Sing+1	0.3155565257696813	ins
aux_tense_dist_Past	0.3129851610526535	succ
aux_form_dist_Ger	0.3086531442613346	ins
prep_dist_2	0.3057295132958554	ins
dep_dist_det:predet	0.30007963434041274	ins
upos_dist_VERB	0.2975803927949885	ins
verbs_tense_dist_Past	0.296898643550676	ins
dep_dist_acl	0.2831077590624652	succ
upos_dist_SYM	0.28051317751977023	ins
upos_dist_NOUN	0.2785753762108627	ins
verbs_num_pers_dist_Sing+1	0.2651126178769086	ins
dep_dist_nummod	0.26479307370417204	succ
principal_proposition_dist	0.2558687633759308	succ
dep_dist_xcomp	0.24396899050754647	ins
verbs_mood_dist_Sub	0.2404993292672185	ins
aux_form_dist_Fin	0.23843200102861054	succ

verb_edges_dist_6	0.2297529431481949	ins
prep_dist_1	0.2239034494553761	ins
verbs_num_pers_dist_+	0.22291313101377863	succ
aux_num_pers_dist_Sing+3	0.20465456350550118	succ
verbs_num_pers_dist_Plur+3	0.19043974641601377	ins
aux_mood_dist_Sub	0.1903513189050966	succ
aux_tense_dist_Imp	0.1898175349308323	ins
verbs_tense_dist_Fut	0.1836832514663673	succ
dep_dist_csubj	0.1798772508176485	succ
upos_dist_CCONJ	0.17429298479928215	ins
obj_pre	0.1594034280949876	ins
verb_edges_dist_4	0.1581257459189747	ins
verbs_tense_dist_Imp	0.1529020786723387	ins
aux_num_pers_dist_Plur+3	0.14594551523983793	ins
aux_mood_dist_Ind	0.1393210121151976	ins
dep_dist_appos	0.13280833339866857	ins
aux_tense_dist_Pres	0.1326496518093241	succ
avg_prepositional_chain_len	0.13208976640203654	succ
subordinate_pre	0.13208025320524283	ins
verb_edges_dist_2	0.11420341138061413	ins
dep_dist_obl:agent	0.11273713403591752	succ
verbs_tense_dist_Pres	0.1085039789196149	succ
subordinate_dist_3	0.10268971120585134	succ
dep_dist_det:poss	0.08819230521894514	ins
verbs_mood_dist_Ind	0.0874302729627578	ins
subordinate_dist_1	0.08742230485022362	ins
dep_dist_cop	0.08120732192797521	ins
aux_tense_dist_Fut	0.06838350514390627	ins

dep_dist_nsubj	0.06700462178286884	succ
upos_dist_AUX	0.05945615290350042	succ
verbs_num_pers_dist_Sing+3	0.05944771900013806	ins
dep_dist_expl	0.058749554603001276	succ
dep_dist_acl:relcl	0.05841918871667599	succ
ttr_form_chunks_200	0.057071241662634616	ins
aux_form_dist_Part	0.05597542820808047	succ
upos_dist_DET	0.05017617998489181	succ
subj_pre	0.04965960776721386	succ
aux_num_pers_dist_Sing+2	0.04012314981685202	succ
dep_dist_expl:impers	0.03980557841237364	ins
aux_num_pers_dist_Plur+1	0.03846122735224889	ins
verb_edges_dist_3	0.03765064500476629	ins
dep_dist_root	0.037240674212281374	ins
subj_post	0.03526177919132333	succ
subordinate_dist_5	0.03418988555828326	ins
subordinate_post	0.025533238631805497	ins
aux_form_dist_Inf	0.02389843907503156	succ
verb_edges_dist_5	0.01413199088657071	ins
obj_post	0.0017899362579541177	succ

Pesi classificazione modello linguistico LOTR

Nome feature	**Coef**	**Direzione**
avg_token_per_clause	2.0192351777084823	ins
tokens_per_sent	1.245976383454497	ins
char_per_tok	1.2285961850404086	succ
ttr_lemma_chunks_100	1.1037082746859261	ins
dep_dist_obj	1.0204533963384783	succ

verb_edges_dist_5	1.0030965119705282	succ
dep_dist_parataxis	0.9977418978051626	ins
aux_form_dist_Ger	0.9862001439933517	ins
prep_dist_5	0.9454029972348055	succ
verbs_mood_dist_Cnd	0.9351807731460552	ins
dep_dist_flat	0.9127786822518695	succ
verbal_root_perc	0.9105977647334872	ins
aux_num_pers_dist_Plur+2	0.9089533711690122	succ
dep_dist_nsubj:pass	0.9082335803940429	ins
verbs_form_dist_Part	0.8591585350520594	succ
dep_dist_csubj	0.8490100187815741	ins
upos_dist_PUNCT	0.8305244209324065	succ
dep_dist_orphan	0.7875863585317071	succ
dep_dist_amod	0.7660873543161234	ins
verb_edges_dist_4	0.7619823507082427	succ
upos_dist_VERB	0.76009022362502	ins
dep_dist_punct	0.7218621176913533	succ
dep_dist_nsubj	0.7186324354250969	succ
ttr_form_chunks_100	0.7021141474175575	ins
dep_dist_advcl	0.682783214252818	ins
upos_dist_NOUN	0.6815190140754338	succ
verbs_mood_dist_Imp	0.6692176148590963	succ
upos_dist_NUM	0.6541646083831432	ins
verbs_form_dist_Fin	0.6340200372752152	ins
verb_edges_dist_3	0.6279541513628	ins
aux_num_pers_dist_Plur+3	0.6267000408510517	ins
verbs_tense_dist_Fut	0.6261009971602941	ins
verb_edges_dist_6	0.6143812425986469	succ

n_prepositional_chains	0.6078739527351642	succ
verbs_num_pers_dist_Plur+2	0.602021432862807	ins
dep_dist_expl	0.5951173603871615	succ
verbs_num_pers_dist_Plur+	0.5888026632349652	succ
n_sentences	0.5844590179974481	ins
n_tokens	0.5806446912747549	succ
avg_links_len	0.5468963838853624	ins
dep_dist_conj	0.5418304674001148	ins
obj_pre	0.5102843527424656	succ
upos_dist_SYM	0.4950685826470057	succ
max_links_len	0.49412817952624194	succ
dep_dist_fixed	0.4869246537023071	ins
ttr_lemma_chunks_200	0.45281331679969594	succ
verb_edges_dist_0	0.44982448547043463	succ
verbs_tense_dist_Pres	0.44874114331674636	succ
upos_dist_ADV	0.4403656896181873	ins
verbs_num_pers_dist_+3	0.43197958071010245	succ
verb_edges_dist_1	0.4189519658810652	ins
upos_dist_X	0.3972175073356771	ins
verbs_num_pers_dist_Sing+1	0.39059005389199764	succ
upos_dist_PART	0.37944340409583777	ins
aux_form_dist_Inf	0.3457711700519857	succ
dep_dist_flat:name	0.3413319197053054	succ
dep_dist_iobj	0.3401051614597383	succ
dep_dist_nummod	0.3393320989294486	ins
verbs_num_pers_dist_+	0.33751470346050255	succ
dep_dist_root	0.33648501337767	ins
dep_dist_det:predet	0.3230698055880784	ins

upos_dist_ADP	0.3213322613015254	succ
dep_dist_flat:foreign	0.3165453289925302	ins
dep_dist_compound	0.30689022000009897	ins
aux_form_dist_Fin	0.30448666346002495	ins
dep_dist_det	0.300307143466109	succ
aux_mood_dist_Sub	0.2981448343546595	succ
aux_mood_dist_Cnd	0.28898763759247476	ins
dep_dist_case	0.28744341271323504	succ
dep_dist_vocative	0.2859230356285797	ins
verbs_tense_dist_Past	0.27994589493717476	succ
dep_dist_obl	0.27869530835209083	ins
subordinate_dist_3	0.275739184868896	succ
avg_prepositional_chain_len	0.27557782687794624	succ
verbs_form_dist_Ger	0.2749680094299949	ins
dep_dist_appos	0.27475261387324046	succ
dep_dist_ccomp	0.2709461857351366	ins
upos_dist_PRON	0.26913479107372285	ins
verbs_num_pers_dist_Sing+3	0.2631272292586487	succ
upos_dist_INTJ	0.2605981173587123	succ
aux_form_dist_Part	0.2563176852369001	succ
aux_mood_dist_Imp	0.2520998879205296	ins
dep_dist_expl:pass	0.24937843108447091	succ
subordinate_proposition_dist	0.24929490902274382	succ
subj_post	0.2346414724639699	succ
lexical_density	0.2244476661976863	succ
dep_dist_acl:relcl	0.22384769410129235	succ
obj_post	0.22164841207365527	ins
verbs_tense_dist_Imp	0.2201343716085928	ins

dep_dist_advmod	0.2173506803795412	succ
upos_dist_CCONJ	0.21455419993919042	ins
aux_num_pers_dist_Sing+3	0.2087236652422898	ins
dep_dist_nmod	0.19756356974583428	ins
ttr_form_chunks_200	0.19168763769246022	succ
subordinate_dist_1	0.19074031672381686	succ
dep_dist_discourse	0.18136470953991068	succ
aux_tense_dist_Imp	0.18092511330028727	ins
subordinate_post	0.17860332346852095	succ
prep_dist_4	0.17782233927811608	succ
aux_tense_dist_Pres	0.1738848113623105	ins
upos_dist_AUX	0.17319152126503193	ins
dep_dist_expl:impers	0.17277250521285548	succ
upos_dist_DET	0.16335560175849323	succ
avg_verb_edges	0.16081700723424258	succ
verbs_num_pers_dist_Sing+2	0.14250062844949504	ins
verbs_mood_dist_Ind	0.1419842206758172	succ
aux_tense_dist_Past	0.13307294920133256	succ
aux_tense_dist_Fut	0.12370949264065448	succ
dep_dist_det:poss	0.12235956111762421	ins
dep_dist_acl	0.12001353166867702	succ
verbs_form_dist_Inf	0.11957617951467822	succ
subordinate_dist_4	0.11626391969356384	succ
prep_dist_3	0.11415329142747692	ins
upos_dist_SCONJ	0.10622818363437359	succ
dep_dist_mark	0.10509470344366399	ins
dep_dist_aux:pass	0.10348516765192657	ins
dep_dist_cop	0.09628767496522529	ins

verbs_num_pers_dist_Plur+3	0.09165022814485539	succ
upos_dist_PROPN	0.08534106565822043	succ
subordinate_dist_5	0.08051019794007057	succ
upos_dist_ADJ	0.07772479454090638	ins
subordinate_pre	0.07370681389143512	succ
aux_num_pers_dist_Sing+1	0.07261937048737677	succ
prep_dist_2	0.0702292367257284	ins
avg_subordinate_chain_len	0.0630515380340934	succ
dep_dist_aux	0.05830467311952935	succ
subj_pre	0.058153775897591575	ins
aux_num_pers_dist_Plur+1	0.05486445327738027	ins
verb_edges_dist_2	0.05415117612992755	succ
dep_dist_dislocated	0.04713683068716848	ins
avg_max_depth	0.046493033021610196	ins
prep_dist_1	0.04599188872154847	succ
principal_proposition_dist	0.03934103164608335	succ
dep_dist_cc	0.037943800905650624	succ
subordinate_dist_2	0.03456617172968182	ins
avg_max_links_len	0.02923060676315057	succ
verbs_mood_dist_Sub	0.020096182763168168	ins
aux_num_pers_dist_Sing+2	0.019591998237696962	ins
aux_mood_dist_Ind	0.018750396212357662	ins
dep_dist_obl:agent	0.018614423611625974	succ
verbs_num_pers_dist_Plur+1	0.013210901242691054	ins
verbal_head_per_sent	0.011601012632107367	ins
dep_dist_xcomp	0.004088748240033343	succ
dep_dist_dep	0.0	succ