evaluation

April 27, 2020

1 Comparativa de Word2Vec, GloVe y FastText para medir la similaridad semántica entre pares de oraciones

1.1 Pablo Valdunciel Sánchez

1.2 Modelos

Utilizamos el modelo *KeyedVectors* de la librería *gesim* para cargar los vectores pre-entrenados de los diferentes modelos

[2]:
```
from gensim.models import KeyedVectors
```

Cargamos los vectores pre-entrenados con modelos Word2Vec, GloVe y FastText. Los vectores pre-entrenados de cada modelo utilizados son:

Modelo	Vectores pre-entrenados	Vocabulario	Dimensión de los vectores	Idioma
Word2Vec	GoogleNews-vectors-negative300.bin.gz	3 millones	300	Inglés
GloVe	Common Crawl (840B tokens,2.2M vocab, cased, 300d vectors)	2,2 millones	300	Inglés
FastText	rawl-300d-2M.vec.zip: 2 million word vectors trained on Common Crawl (600Btokens)	2 millones	300	Inglés

En el caso de los vectores de GloVe, se ha utilizado la función *glove2word2vec* de la librería *gensim* para convertir el archivo al formato Word2Vec.

[2]:
```
PATH_WORD2VEC = './data/embedding/word2vec/GoogleNews-vectors-negative300.bin'
PATH_GLOVE = './data/embedding/glove/glove.840B.300d.w2v.txt'
PATH_FASTTEXT = './data/embedding/fasttext/crawl-300d-2M.vec'
```

Cargamos los tres conjuntos vectores, lo que puede llevar varios minutos.

[3]:
```
word2vec = KeyedVectors.load_word2vec_format(PATH_WORD2VEC, binary=True)
```

[4]:
```
glove = KeyedVectors.load_word2vec_format(PATH_GLOVE, binary=False)
```

[5]:
```
fasttext = KeyedVectors.load_word2vec_format(PATH_FASTTEXT, binary=False)
```

1.3 Datos

Haciendo uso de las funciones del módulo *load.py* cargamos los conjuntos de **test** de las tareas STS12, STS13, STS14, STS15, STS16 y SICK-R.

Tarea	Nº instancias	Referencia
STS12	3108	SemEval 2012 - STS
STS13	1500	SemEval 2013 - STS
STS14	3750	SemEval 2014 - STS
STS15	3000	SemEval 2015 - STS
STS16	1186	SemEval 2016 - STS
SICK-R	9927	SICK Relatedness

Todas las funciones del módulo *load.py* llevan a cabo un preprocesamiento de las oraciones según los parámetros que se indiquen. Por defecto, el preprocesamiento de las oraciones se basa en realizar una *tokenización/segmentación* de las mismas. A mayores, existen las siguientes posibilidades:

- **lowercase**: hacer que todas las palabras estén en minúscula.
- **stop_words**: eliminar las palabras que no aportan casi significado semántico como determinantes, preprosiciones, etc.
- **punctuation**: elminar los símbolos de puntuación.
- **only_ascci**: eliminar las palabras que no estén formadas por caracteres ASCII.
- **lemmatization**: sustituir las palabras por su lexema.

El preprocesamiento del texto está implementado en la función *preprocess* en el módulo *utils.py*. La función *preprocess* hace uso de la librería spaCy para llevar a cabo el preprocesamiento.

[3]:
```
from load import load_sts_12, load_sts_13, load_sts_14, load_sts_15,␣
↪load_sts_16, load_SICK
from load import load_frequencies
```

En este caso, tan solo se realiza la *tokenización/segmentación* de las oraciones y una conversión a minúscula (parámetro *lowercase* igual a True) para lograr uniformidad.

[1]:
```
PATH_STS = './data/datasets/STS'
PATH_SICK = './data/datasets/SICK'
PATH_FREQUENCIES = './data/frequencies.tsv'
PREPROCESSING =  {'lowercase':  True,
                  'stop_words': False,
                  'punctuation': False,
                  'only_ascii': False,
                  'lemmatization': False
                 }
```

Cargamos también las frecuencias de las palabras en el corpus para poder aplicar el SIF.

[3]:
```
freqs = load_frequencies(PATH_FREQUENCIES)
```

[4]:
```
sts12 = load_sts_12(PATH_STS, PREPROCESSING, verbose=True)
sts13 = load_sts_13(PATH_STS, PREPROCESSING, verbose=True)
sts14 = load_sts_14(PATH_STS, PREPROCESSING, verbose=True)
sts15 = load_sts_15(PATH_STS, PREPROCESSING, verbose=True)
sts16 = load_sts_16(PATH_STS, PREPROCESSING, verbose=True)
sick = load_SICK(PATH_SICK, PREPROCESSING, verbose=True)
```

```
***** TASK: STS12 *****

Preprocessing -MSRpar-
-MSRpar- preprocessed correctly
Preprocessing -MSRvid-
-MSRvid- preprocessed correctly
Preprocessing -SMTeuroparl-
-SMTeuroparl- preprocessed correctly
Preprocessing -surprise.OnWN-
-surprise.OnWN- preprocessed correctly
Preprocessing -surprise.SMTnews-
-surprise.SMTnews- preprocessed correctly


***** TASK: STS13 (-SMT) ***

Preprocessing -FNWN-
-FNWN- preprocessed correctly
Preprocessing -headlines-
-headlines- preprocessed correctly
Preprocessing -OnWN-
-OnWN- preprocessed correctly


***** TASK: STS14 *****

Preprocessing -deft-forum-
-deft-forum- preprocessed correctly
Preprocessing -deft-news-
-deft-news- preprocessed correctly
Preprocessing -headlines-
-headlines- preprocessed correctly
Preprocessing -images-
-images- preprocessed correctly
Preprocessing -OnWN-
-OnWN- preprocessed correctly
```

```
Preprocessing -tweet-news-
-tweet-news- preprocessed correctly


***** TASK: STS15 *****

Preprocessing -answers-forums-
-answers-forums- preprocessed correctly
Preprocessing -answers-students-
-answers-students- preprocessed correctly
Preprocessing -belief-
-belief- preprocessed correctly
Preprocessing -headlines-
-headlines- preprocessed correctly
Preprocessing -images-
-images- preprocessed correctly


***** TASK: STS16 *****

Preprocessing -answer-answer-
-answer-answer- preprocessed correctly
Preprocessing -headlines-
-headlines- preprocessed correctly
Preprocessing -plagiarism-
-plagiarism- preprocessed correctly
Preprocessing -postediting-
-postediting- preprocessed correctly
Preprocessing -question-question-
-question-question- preprocessed correctly


***** Task: SICK-Relatedness*****
```

1.4 Métodos

Los métodos para calcular la similaridad semántica entre dos oraciones se encuentran en el módulo *methods.py* y son:

- **avg_cosine**: el vector de una oración se obtiene haciendo la media (*average*) de los vectores de las palabras de esa oración. La similaridad entre dos vectores se calcula utilizando la similitud coseno.
- **sif_cosine**: el vector de una oración se obtiene haciendo una media ponderada (*Smooth Inverse Frequency*, SIF) de los vectores de las palabras de esa oración. La similaridad entre dos vectores se calcula utilizando la similitud coseno.
- **wmd**: la similariad entre dos oraciones se calcula como el contrario de la distancia *Word Mover's Distance* entre las mismas. El modelo *KeyedVectors* de *gensim* incorpora el cálculo

de esta distancia (ver método). Esta distancia no está acotada y tomará el valor *infinito* cuando ninguna de las palabras de alguna de las dos oraciones este presente en el modelo, por ello se ha acotado en el intervalo [0, 100].

[4]:
```
from functools import partial
from methods import avg_cosine, wmd, sif_cosine
```

[15]:
```
METHODS = [
    ("W2V + AVG", partial(avg_cosine, model=word2vec)),
    ("W2V + SIF", partial(sif_cosine, model=word2vec, frequencies=freqs, a=0.
 ↪001)),
    ("W2V + WMD", partial(wmd, model=word2vec)),
    ("GLOVE + AVG", partial(avg_cosine, model=glove)),
    ("GLOVE + SIF", partial(sif_cosine, model=glove, frequencies=freqs, a=0.
 ↪001)),
    ("GLOVE + WMD", partial(wmd, model=glove)),
    ("FT + AVG", partial(avg_cosine, model=fasttext)),
    ("FT + SIF", partial(sif_cosine, model=fasttext, frequencies=freqs, a=0.
 ↪001)),
    ("FT + WMD", partial(wmd, model=fasttext))
]
```

1.5 Evaluación

En el módulo *utils.py* se encuentra la función *evaluate* que calcula los coeficientes de correlación de Pearson y Spearman entre las puntuaciones *Gold Standard* de una tarea y las mediadas de similariad proporcionadas por uno o varios métodos.

[6]:
```
from utils import evaluate
from scipy.stats import rankdata, chi2
from Orange.evaluation import compute_CD, graph_ranks
import numpy as np
import pprint
import matplotlib.pyplot as plt
```

[18]:
```
N = 6  # Número de conjuntos de datos
K = 9  # Número de métodos
```

[22]:
```
sts12_pearson, sts12_spearman = evaluate(sts12, METHODS)
sts13_pearson, sts13_spearman = evaluate(sts13, METHODS)
sts14_pearson, sts14_spearman = evaluate(sts14, METHODS)
sts15_pearson, sts15_spearman = evaluate(sts15, METHODS)
sts16_pearson, sts16_spearman = evaluate(sts16, METHODS)
sick_pearson, sick_spearman = evaluate(sick, METHODS)
```

1.5.1 Coeficientes de correlación de Pearson

En la siguiente tabla podemos observar los coeficientes de correlación de Pearson obtenidos por cada uno de los métodos sobre cada tarea.

[32]:
```
pearson_corr = np.array([ np.array(list(sts12_pearson.values())),
                          np.array(list(sts13_pearson.values())),
                          np.array(list(sts14_pearson.values())),
                          np.array(list(sts15_pearson.values())),
                          np.array(list(sts16_pearson.values())),
                          np.array(list(sick_pearson.values()))
                        ])
```

Corpus Método	W2V + AVG	W2V + SIF	W2V + WMD	GLOVE + AVG	GLOVE + SIF	GLOVE + WMD	FT + AVG	FT + SIF	FT + WMD
STS12	0,5577	0,5670	0,4735	0,5503	0,5887	0,5511	0,6006	0,6213	0,5535
STS13	0,6402	0,7227	0,5212	0,5431	0,7004	0,4865	0,6396	0,7431	0,5043
STS14	0,6868	0,7279	0,6122	0,5625	0,7069	0,5803	0,6664	0,7356	0,5921
STS15	0,7049	0,7490	0,6839	0,6006	0,7319	0,6796	0,6999	0,7613	0,6888
STS16	0,6391	0,7191	0,6408	0,5025	0,6847	0,6133	0,6326	0,7315	0,6312
SICK-R	0,7012	0,7318	0,6125	0,6517	0,7215	0,5995	0,6980	0,7400	0,6045

1.5.2 Rankings

Calculamos los rankings para cada tarea y los rankings promedios.

[35]:
```
ranks = np.array([np.array(rankdata((-1)*pearson_corr[task])) for task in␣
↪range(pearson_corr.shape[0])])
avg_ranks = np.average(ranks, axis=0)
```

En la siguiente tabla podemos observar los rankings obtenidos:

Corpus Método	W2V + AVG	W2V + SIF	W2V + WMD	GLOVE + AVG	GLOVE + SIF	GLOVE + WMD	FT + AVG	FT + SIF	FT + WMD
STS12	5	4	9	8	3	7	2	1	6
STS13	4	2	7	6	3	9	5	1	8
STS14	4	2	6	9	3	8	5	1	7
STS15	4	2	7	9	3	8	5	1	6
STS16	5	2	4	9	3	8	6	1	7
SICK-R	4	2	7	6	3	9	5	1	8
Promedio	4,33	2,33	6,67	7,83	3,00	8,17	4,67	1,00	7,00

1.5.3 Test de Friedman

Se plantea el siguiente contraste de hipótesis para determinar si los diferentes métodos son equivalentes:

- H0: los métodos son equivalentes
- H1: los métodos no son equivalentes

Para resolver este contraste de hipótesis se realiza el Test de Friedman sobre los rankings promedios. Aún sin realizar el test, estos rankings promedios proporcionan ya una buena comparación de los diferentes métodos.

Posición	Ranking promedio	Método
1	1	FT + SIF
2	2,33	W2V + SIF
3	3	GLOVE + SIF
4	4,33	W2V + AVG
5	4,67	FT + AVG
6	6,67	W2V + WMD
7	7	FT + WMD
8	7,83	GLOVE + AVG
9	8,17	GLOVE + WMD

[7]:
```
sqr_avg_ranks = np.array(list(map(lambda r: r**2, avg_ranks)))
friedman = (12*N) / (K*(K+1)) * (sum(sqr_avg_ranks) - (K*(K+1)**2/4))
p_value = 1 - chi2.cdf(friedman, K-1)

print("p-valor = {:4f}".format(p-value))
```

```
p-valor = 0.000028
```

El p-valor obtenido para el Test de Friedman es **0,000028**, por lo que la hipótesis nula, H0, sobre la equivalencia de los métodos se rechaza a los niveles de confianza habituales.

1.5.4 Test de Nemenyi

Una vez rechazada la hipótesis nula H0 de la equivalencia de los métodos, se realiza el test *post-hoc* de Nemenyi para determinar entre que pares de métodos existen diferencias significativas. Considerando un nivel de confianza del 95%, alpha = 0.05, y sabiendo que el número de métodos es k=*9* y el número de conjuntos de datos es N=*6*, la distancia crítica de Nemenyi calculada es:

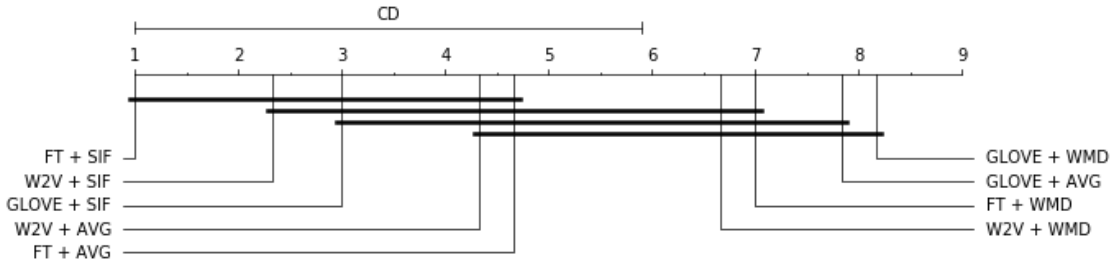
[37]:
```
cd_nemenyi = compute_CD(avg_ranks, N, alpha='0.05', test='nemenyi')
print("Distancia crítica de Nemenyi (alpha = 0.05) = ", cd_nemenyi)
```

```
Distancia crítica de Nemenyi (alpha = 0.05) =  4.904265743437033
```

Representamos las diferencias significativas en base a la distancia crítica obtenida.

[38]:
```
graph_ranks(avg_ranks, list(sts12_pearson.keys()), cd=cd_nemenyi, width=9,␣
↪textspace=1)
```

```
plt.show()
```



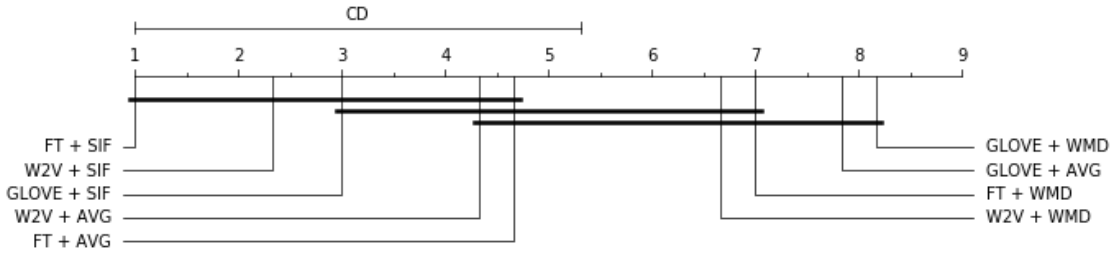
1.5.5 Test de Bonferroni-Dunn

Dado que el método *FT + SIF* ha obtenido la primera posición para todos los conjuntos de datos, parece razonable utiizar el Test de Bonferroni-Dunn para comparar este método con los demás. El Test de Bonferroni-Dunn se basa tambien en el calculo de una distancia crıtica cuya expresión es muy similar a la distancia crítica de Nemenyi. Considerando un nivel de confianza del 95%, $=$ 0,05, y sabiendo que el numero de metodos es $K=9$ y el numero de conjuntos de datos es $N=6$, la distancia crıtica de Bonferroni-Dunn calculada es:

[39]:
```
cd_bonferroni_dunn = compute_CD(avg_ranks, N, alpha='0.05',␣
↪test='bonferroni-dunn')
print("Distancia crítica de Bonferroni-Dunn (alpha = 0.05) = ",␣
↪cd_bonferroni_dunn)
```

```
Distancia crítica de Bonferroni-Dunn (alpha = 0.05) =  4.307022173149333
```

[40]:
```
graph_ranks(avg_ranks, list(sts12_pearson.keys()), cd=cd_bonferroni_dunn,␣
↪width=9, textspace=1)
plt.show()
```



La distancia crítica del Test de Bonferroni-Dunn es **4.3070**. Seleccionando el método *FT + SIF* como método de control, se comprueba cómo este método es significativamente mejor que *W2V + WMD*, *FT + WMD*, *GLOVE + WMD* y *GLOVE + AVG*.

1.6 Conclusiones

1. Combinar los vectores de palabras pre-entrenados de FastText realizando una media SIF es el método más preciso para medir la similaridad semántica entre pares de documentos entre los combinaciones evaluadas.

2. Entre los conjuntos de palabras pre-entrenados evaluados, se puede observar como el conjunto de GloVe obtiene peores resultados que el conjunto de Word2Vec o FastText, independientemente del método utilizado para calcular la similaridad semántica.

3. Aunque realizar una media no ponderada de los vectores de palabras es un buen punto de partida para obtener una vectorización de una oración, aplicar una ponderación SIF (*Smooth Inverse Frequency*) es una mejor alternativa.

4. Entre las posibilidades para medir la similaridad, el inverso de la distancia *Word Mover's Distance* como medida proporciona los peores resultados.