# Documentacion

November 7, 2017

## 1 Expansión de consultas con Word2Vec

Antes de ejecutar este notebook asegurate de que elasticsearch está iniciado. Además los tweets deben estar indexados. Para indexarlos ejecuta el script 'index.py'.

### 1.1 Importando las librerías

Primero vamos a importar las librerías que vamos a usar.

```
In [1]: import elasticsearch.helpers
        from elasticsearch import Elasticsearch
        from elasticsearch_dsl import Search, Q
        from gensim.models import word2vec
```

Las consultas que realizaremos a lo largo de la práctica son:

```
In [2]: index = '2008-feb-02-04-en'
        queries = ['American Football Conference', 'David Tyree','defensive end', 'Eli
        Manning','football', 'Glendale',
                   'Laurence Maroney', 'Miami Dolphins', 'Michael Strahan', 'National Football
        Conference',
                   'National Football League', 'New England Patriots', 'New York', 'New York
        Giants', 'NFL',
                   'Plaxico Burress', 'quarterback', 'Randy Moss', 'running back', 'Super Bowl',
                   'University of Phoenix', 'Wide receiver', 'XLII']
```

La mitad de estas consultas serán para entrenar el modelo de Word2Vec, mientras que la otra mitad se usarán para realizar la expansión de consultas.

```
In [3]: queries_train = queries[:len(queries)//2]
        queries_test = queries[len(queries)//2:]
```

### 1.2 Entrenamiento de Word2Vec

Ahora vamos a comparar el resultado que obtenemos al entrenar el modelo con distinto número de documentos.

#### 1.2.1 Prueba 1: Utilizando las consultas de entrenamiento

```
In [4]: client = Elasticsearch()
        documents = []
        for query in queries_train:
            s = Search(using=client, index=index).query("match", text=query)

            # al llamar a s.scan() utilizamos las características de scroll
```

```
            # de elasticsearch, por lo que obtenemos todos los resultados
            for tweet in s.scan():
                documents.append(tweet.text)
        print("Numero de documentos: {}".format(len(documents)))
```

```
Numero de documentos: 14544
```

Una vez que tenemos los documentos de entrenamiento, necesitamos preparalos para que el modelo de Word2Vec los pueda procesar. Primero necesitamos separar cada tweet en una lista de palabras. Después, eliminaremos las palabras vacías de cada tweet para mejorar los resultados obtenidos con Word2Vec. Finalmente, eliminaremos símbolos de puntuación pegados a las palabras (por ejemplo, 'football,') para evitar añadir ruido al modelo:

```
In [5]: import string
        from stop_words import get_stop_words
        stop_words = get_stop_words('english')

        # creamos una tabla de traducción que usaremos para eliminar los simbolos de puntuación.
        remove_punctuation_map = dict((ord(char), None) for char in string.punctuation)
        documents = [[word.translate(remove_punctuation_map) # eliminar simb. puntuacion de cada
        palabra
                      for word in tweet.split(' ') if word not in stop_words] #  eliminar
        palabras vacías
                      for tweet in documents]

In [6]: model = word2vec.Word2Vec(documents, size=100, window=5, min_count=5, workers=4)

In [7]: model.most_similar('football')
```

```
Out[7]: [('good', 0.9967594146728516),
         ('watching', 0.9956277012825012),
         ('watch', 0.9956172704696655),
         ('Im', 0.9951895475387573),
         ('just', 0.9949032068252563),
         ('hate', 0.9948126673698425),
         ('think', 0.9946979284286499),
         ('get', 0.9940967559814453),
         ('game', 0.9934204816818237),
         ('really', 0.9933381080627441)]
```

### 1.2.2 Prueba 2: Utilizando el top 100 de resultados de cada consulta

```
In [8]: documents = []
        for query in queries:
            s = Search(using=client, index=index).query("match", text=query)

            s = s[0:100]
            for tweet in s:
                documents.append(tweet.text)
        documents = [[word.translate(remove_punctuation_map) # eliminar simb. puntuacion de cada
        palabra
                      for word in tweet.split(' ') if word not in stop_words] # eliminar
        palabras vacías
                      for tweet in documents]
        print("Numero de documentos: {}".format(len(documents)))
        model = word2vec.Word2Vec(documents, size=100, window=5, min_count=5, workers=4)
```

```
Numero de documentos: 2191
```

```
In [9]: model.most_similar('football')
```

```
Out[9]: [('I', 0.9997382164001465),
         ('University', 0.9997354745864868),
         ('', 0.9997296333312988),
         ('quarterback', 0.9997286200523376),
         ('hour', 0.9997243881225586),
         ('can', 0.9997240900993347),
         ('game', 0.9996951818466187),
         ('Plaxico', 0.9996931552886963),
         ('Vick', 0.9996829032897949),
         ('Eli', 0.9996739625930786)]
```

### 1.2.3 Prueba 3: Utilizando toda la colección

```
In [32]: documents = []
         s = Search(using=client, index=index)

         for tweet in s.scan():
             documents.append(tweet.text)
         documents = [[word.translate(remove_punctuation_map) # eliminar simb. puntuacion de cada
         palabra
                       for word in tweet.split(' ') if word not in stop_words] # eliminar
         palabras vacías
                       for tweet in documents]
         print("Numero de documentos: {}".format(len(documents)))
         model = word2vec.Word2Vec(documents, size=100, window=5, min_count=5)
```

```
Numero de documentos: 390000
```

```
In [33]: model.most_similar('football')
```

```
Out[33]: [('sports', 0.8406401872634888),
          ('rugby', 0.822288990020752),
          ('game', 0.8177592754364014),
          ('soccer', 0.7996732592582703),
          ('commercials', 0.7861671447753906),
          ('halftime', 0.7768101692199707),
          ('SB', 0.7765822410583496),
          ('winning', 0.77300626039505),
          ('patriots', 0.7709023952484131),
          ('pregame', 0.7703537344932556)]
```

## 1.3 Dibujando vectores de palabras

Ahora vamos a dibujar un gráfico que nos ayude a visualizar el modelo que utiliza Word2Vec para determinar que palabras son similares a una palabra dada. En el constructor de Word2Vec uno de los parámetros que se le introdujo fue 'size=100'. Esto quiere decir que cada palabra de los documentos de entrenamiento se representa en el modelo utilizándo un vector de 100 dimensiones.

```
In [12]: len(model.wv['football'])
```

```
Out[12]: 100
```

Las palabras que suelan aparecer en el mismo contexto tendrán unos vectores parecidos, mientras que las palabras que no tengan nada en común tenderán a estar alejadas. Para poder ver esta relación vamos a reducir la dimensionalidad de algunos vectores a 2 dimensiones utilizando la técnica de PCA (Principal Component Analysis):

```
In [34]: from sklearn.decomposition import PCA

         X_train = [model.wv[word] for word in model.wv.vocab]

         words = ['football', 'superbowl', 'queen', 'sister', 'brother', 'king',
                  'mother', 'father', 'Europe', 'Asia', 'space', 'bowl', 'super']
         X = [model.wv[word] for word in words]

         pca = PCA(n_components=2)
         pca.fit(X_train)
         result = pca.transform(X)
         print(result)
```

```
[[-4.41699481  3.51100563]
 [-5.16985036  4.04712219]
 [-0.37515158  0.57880411]
 [-2.07448958  1.44980465]
 [-2.51391966  1.7688225 ]
 [-0.85924824  0.95537341]
 [-0.54096789  2.29067224]
 [ 0.18359422  2.47843291]
 [ 1.08165519  0.36616536]
 [ 0.98202308  0.05303514]
 [-2.28814674  5.46711759]
 [-4.50576676  3.89280608]
 [-4.41440813  4.10941419]]
```

Ahora que tenemos los vectores en 2 dimensiones, podemos realizar un scatter plot y dibujar cada palabra en el plano:

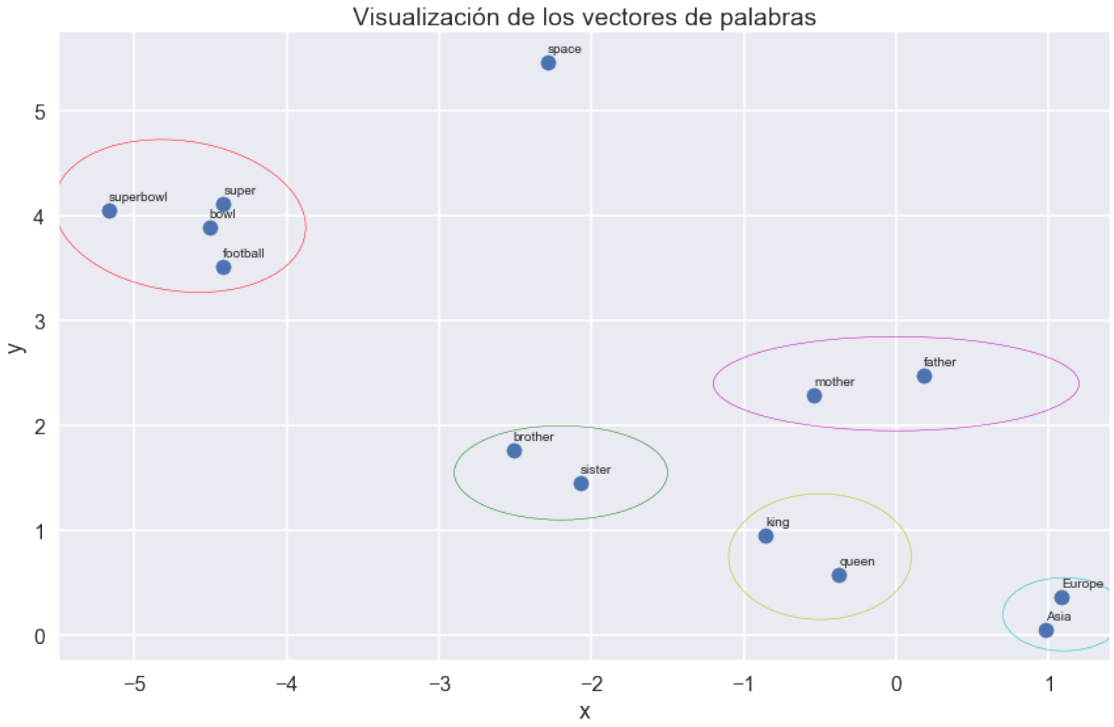
```
In [36]: from matplotlib.patches import Ellipse
         import matplotlib as mpl
         import matplotlib.pyplot as plt
         %matplotlib inline

         plt.style.use('seaborn-poster')

         plt.title('Visualización de los vectores de palabras')
         plt.xlabel('x')
         plt.ylabel('y')
         plt.scatter(result[:,0], result[:,1])
         for i, word in enumerate(words):
             plt.annotate(word, xy=(result[i,0],result[i,1] + 0.1))

         # dibujando elipses para que nos ayuden a visualizar las relaciones
         fig = plt.gcf()
         fig.gca().add_artist(mpl.patches.Ellipse(xy=(-4.7,4), width=1.7, height=1.4, angle=155,
         color='r', fill=False))
         fig.gca().add_artist(mpl.patches.Ellipse(xy=(-2.2,1.55), width=1.4, height=0.9,
         angle=180, color='g', fill=False))
         fig.gca().add_artist(mpl.patches.Ellipse(xy=(1.1,0.2), width=0.8, height=0.7, angle=180,
         color='c', fill=False))
         fig.gca().add_artist(mpl.patches.Ellipse(xy=(0,2.4), width=2.4, height=0.9, angle=180,
         color='m', fill=False))
         fig.gca().add_artist(mpl.patches.Ellipse(xy=(-0.5,0.75), width=1.2, height=1.2,
         angle=180, color='y', fill=False))
         plt.show()
```

```
plt.show()
```




## 1.4 Expansión de consultas

```
In [15]: start_query = 'Super Bowl'
         num_expanded_words = 5
         final_query = []

         # separamos la query original en términos y sacamos
         # las 5 palabras más similares de cada término
         for word in start_query.split(' '):
             expanded_words = [tuple[0] for tuple in
         model.most_similar(word)[:num_expanded_words]]
             expanded_words.append(word)
             final_query.append(expanded_words)

In [16]: queries = []
         for expansion in final_query:
             queries.append(Q('match', text=' '.join(expansion)))
         q = Q('bool', should=queries)
         s = Search(using=client, index=index).query(q)
         for tweet in s:
             print(tweet.text)

souper bowl.
Puppy Bowl VI!!!!
not watching the stupor bowl.
Not watching the Stupor Bowl.
Stupor Bowl at the Crows
Let's hope it's not the Stupor Bowl.
```

```
Going to a presuper  bowl party.. Lemoncello all around!!j
@thomasgvl Does your mom think it's the Souper Bowl?
"Souper" Bowl today! http://tinyurl.com/2zdx7x/20080202/NEWS01/80201025
Going to bed soon... Scholar Bowl in the morning!
```

```
In [17]: s = Search(using=client, index=index).query('match', text='super bowl')
         for tweet in s:
             print(tweet.text)
```

```
Super Bowl, baby, Super Bowl ... go Pats!
super bowl? super martinis!
super bowl? or puppy bowl?
super bowl?!
Super Bowl!!!!!
Super Bowl
Super bowl :)
Super Bowl.
super bowl!!
Super Tuesday not Super Bowl.
```

```
In [18]: model.most_similar('sun')
```

```
Out[18]: [('rain', 0.893586277961731),
          ('cold', 0.8926393985748291),
          ('sunshine', 0.8836721181869507),
          ('warm', 0.8819676637649536),
          ('wind', 0.8775304555892944),
          ('wet', 0.8622894287109375),
          ('outside', 0.8585378527641296),
          ('mountains', 0.8509751558303833),
          ('shining', 0.8475022315979004),
          ('mountain', 0.8448079228401184)]
```

# 2 Titulo 1

## 2.1 Titulo 2

### 2.1.1 Titulo 3

asofmspaofmpasmfo

```
In [19]: print(plt.style.available)
```

```
['_classic_test', 'bmh', 'classic', 'dark_background', 'fivethirtyeight', 'ggplot',
'grayscale', 'seaborn-bright', 'seaborn-colorblind', 'seaborn-dark-palette', 'seaborn-
dark', 'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook',
'seaborn-paper', 'seaborn-pastel', 'seaborn-poster', 'seaborn-talk', 'seaborn-ticks',
'seaborn-white', 'seaborn-whitegrid', 'seaborn']
```

```
In [ ]:
```