Word Analogy and Removal Of Bias from Word Embeddings

January 22, 2024

[1]:
```python
### v1.1
```

[2]:
```python
# Importing the necessary packages

import numpy as np
from w2v_utils import *
```

[3]:
```python
# We will use 50-dimensional GloVe vectors to represent words

words, word_to_vec_map = read_glove_vecs('data/glove.6B.50d.txt')

# words - A set of words in the vocabulary
# word_to_vec_map - A dictionary mapping words to their GloVe vector␣
↪representation

# print(type(words))
# print(type(word_to_vec_map))
```

0.0.1 Completion of Word Analogy

[4]:
```python
def cosine_similarity(u, v):
    """
    Cosine similarity reflects the degree of similarity between u and v

    Arguments:
        u -- a word vector of shape (n,)
        v -- a word vector of shape (n,)

    Returns:
        cosine_similarity
    """

    # Special case - We consider the case u = [0, 0], v=[0, 0]
    if np.all(u == v):
        return 1
```

```
    # Compute the dot product between u and v ( 1 line)
    dot = np.dot(u,v)
    # Compute the L2 norm of u ( 1 line)
    norm_u = np.sqrt(np.sum(np.dot(u,u)))

    # Compute the L2 norm of v ( 1 line)
    norm_v = np.sqrt(np.sum(np.dot(v,v)))

    # Avoid division by 0
    # If the difference between norm_u * norm_v and 0 is less than the absolute␣
↪tolerance = 1e-32, then return 0
    # If the above case, then either one of them is a zero vector and the␣
↪cosine_similarity between a zero vector and the
    # other given vector = 0. A zero vector is orthogonal to any other vector.
    if np.isclose(norm_u * norm_v, 0, atol=1e-32):
        return 0

    # Now we compute the cosine similarity
    cosine_similarity = dot / (norm_u * norm_v)

    return cosine_similarity
```

[5]:
```
def complete_analogy(word_a, word_b, word_c, word_to_vec_map):
    """
    Performs the word analogy task as explained above: a is to b as c is to␣
↪____.

    Arguments:
    word_a -- a word, string
    word_b -- a word, string
    word_c -- a word, string
    word_to_vec_map -- dictionary that maps words to their corresponding␣
↪vectors.

    Returns:
    best_word --  the word such that v_b - v_a is close to v_best_word - v_c,␣
↪as measured by cosine similarity
    """

    # First we convert words to lowercase
    word_a, word_b, word_c = word_a.lower(), word_b.lower(), word_c.lower()

    # Get the word embeddings e_a, e_b and e_c
    e_a, e_b, e_c = word_to_vec_map[word_a], word_to_vec_map[word_b],␣
↪word_to_vec_map[word_c]
```

```
    words = word_to_vec_map.keys()
    # We initialize max_consine_similarity to be a large negative number
    max_cosine_sim = -100
    # We initialize the best word with None
    best_word = None

    # We loop over the whole word vector set
    for w in words:
        # To avoid best_word being one the input words, we skip the input word_c
        # We skip word_c from query
        if w == word_c:
            continue

        # Computing cosine similarity between the vector (e_b - e_a) and the␣
↪vector ((w's vector representation) - e_c)
        cosine_sim = cosine_similarity(e_b - e_a, word_to_vec_map[w] - e_c)

        # If the cosine_sim is more than the max_cosine_sim seen so far,
            # then: set the new max_cosine_sim to the current cosine_sim and␣
↪the best_word to the current word
        if cosine_sim > max_cosine_sim:
            max_cosine_sim = cosine_sim
            best_word = w

    return best_word
```

[6]:
```
triads_to_try = [('italy', 'italian', 'spain'), ('india', 'delhi', 'japan'),␣
↪('man', 'woman', 'boy'), ('small', 'smaller', 'large')]
for triad in triads_to_try:
    print ('{} -> {} :: {} -> {}'.format( *triad, complete_analogy(*triad,␣
↪word_to_vec_map)))
```

```
italy -> italian :: spain -> spanish
india -> delhi :: japan -> tokyo
man -> woman :: boy -> girl
small -> smaller :: large -> smaller
```

0.0.2 Debiasing Word Vectors

[7]:
```
# Calculating the 'Bias' direction

e1 = word_to_vec_map['woman'] - word_to_vec_map['man']
e2 = word_to_vec_map['mother'] - word_to_vec_map['father']
e3 = word_to_vec_map['girl'] - word_to_vec_map['boy']
g = (e1 + e2 + e3)/3
print(g)
```

```
# 'g' is an approximate representation of 'gender'
```

```
[ 0.07656667  0.34967667 -0.40057667 -0.03130333  0.0088      0.72586333
  0.10256     0.14906333  0.4780662  -0.22850987  0.05957667 -0.68663
  0.62210033  0.10395     0.17747667  0.09556867 -0.49258333 -0.17066233
  0.46930033  0.02196333  0.28145667  0.50513333  0.17144733  0.40154767
  0.24039333  0.1646     -0.17984667  0.24042667  0.05689333 -0.31423
 -0.10933333  0.26355967  0.06100667 -0.01156405 -0.12236333 -0.188245
 -0.13215057 -0.068186    0.05624667 -0.29555567 -0.09669533 -0.29559667
  0.62465867 -0.40130167  0.03330667 -0.24831667  0.26381667 -0.28738333
  0.03020433  0.054106  ]
```

[8]:
```
print ('List of names and their similarities with constructed vector:')

# girls and boys name
name_list = ['john', 'marie', 'sophie', 'ronaldo', 'priya', 'rahul',␣
 ↪'danielle', 'reza', 'katy', 'yasmin']

for w in name_list:
    print (w, cosine_similarity(word_to_vec_map[w], g))
```

```
List of names and their similarities with constructed vector:
john -0.30873091089769905
marie 0.34257515107827113
sophie 0.4116200252265308
ronaldo -0.29083978511732383
priya 0.1964679344860046
rahul -0.194921476386334
danielle 0.2923957653171286
reza -0.1679382162425299
katy 0.3113243060566435
yasmin 0.19658379893678699
```

[9]:
```
# In the above case, we observe that female first names tend to have a positive␣
 ↪cosine similarity with our constructed vector,
# while male first names tend to have a negative cosine similarity and this␣
 ↪seems acceptable
```

[10]:
```
# Trying with other words
```

[11]:
```
print('Other words and their similarities:')
word_list = ['lipstick', 'guns', 'science', 'arts', 'literature',␣
 ↪'warrior','doctor', 'tree', 'receptionist',
             'technology',  'fashion', 'teacher', 'engineer', 'pilot',␣
 ↪'computer', 'singer']
for w in word_list:
```

```
    print (w, cosine_similarity(word_to_vec_map[w], g))
```

```
Other words and their similarities:
lipstick 0.4136681512625245
guns -0.08755154639507806
science -0.058374259643848236
arts 0.01176046812578374
literature 0.023169467893751714
warrior -0.16564638100307946
doctor 0.07721412726656676
tree 0.035380421070982306
receptionist 0.30167259871100655
technology -0.1619210846255818
fashion 0.1416547219136271
teacher 0.10545901736578718
engineer -0.22639944157426764
pilot -0.03699357317847414
computer -0.1682103192173514
singer 0.20093000793226248
```

[12]:
```
# We first make the embeddings of words to have L2 norm = 1

word_to_vec_map_unit_vectors = {}
for word in word_to_vec_map.keys():
    embedding = word_to_vec_map[word]
    word_to_vec_map_unit_vectors[word] = embedding/np.linalg.norm(embedding)
```

[13]:
```
e1 = word_to_vec_map_unit_vectors['woman'] - word_to_vec_map_unit_vectors['man']
e2 = word_to_vec_map_unit_vectors['mother'] -␣
 ↪word_to_vec_map_unit_vectors['father']
e3 = word_to_vec_map_unit_vectors['girl'] - word_to_vec_map_unit_vectors['boy']
g_unit = (e1 + e2 + e3)/3
print(g_unit)
```

```
[ 0.01506562  0.05674597 -0.06807928  0.00160754 -0.014503    0.12191253
  0.02812728  0.02154482  0.08825992 -0.04282351  0.00805539 -0.12824188
  0.11472482  0.0173499   0.02193565  0.02041312 -0.08110868 -0.0375994
  0.0869276  -0.00104698  0.05802643  0.07449733  0.03111171  0.06634303
  0.03654503  0.05869284 -0.0254643   0.04347939  0.00634609 -0.05206027
 -0.04707763  0.05191345  0.01418016 -0.00448297 -0.02789399 -0.03850989
 -0.0247095  -0.00942581  0.00506779 -0.04515941 -0.01451673 -0.0586928
  0.11671609 -0.07113848 -0.00264706 -0.03825909  0.05018548 -0.0357483
  0.00161449  0.01005749]
```

[14]:
```
def neutralize(word, g, word_to_vec_map):
    """
```

```
    Removes the bias of "word" by projecting it on the space orthogonal to the␣
↪bias axis.
    This function ensures that gender neutral words are zero in the gender␣
↪subspace.

    Arguments:
        word -- string indicating the word to debias
        g -- numpy-array of shape (50,), corresponding to the bias axis (such␣
↪as gender)
        word_to_vec_map -- dictionary mapping words to their corresponding␣
↪vectors.

    Returns:
        e_debiased -- neutralized word vector representation of the input "word"
    """

    # We select the word vector representation of "word" using the␣
↪word_to_vec_map
    e = word_to_vec_map[word]

    # We compute e_biascomponent
    e_biascomponent = np.linalg.norm(e) * cosine_similarity(e,g) * (g/np.linalg.
↪norm(g))

    # We neutralize e by subtracting e_biascomponent from it
    # e_debiased should be equal to its orthogonal projection.
    e_debiased = e - e_biascomponent

    return e_debiased
```

[15]:
```
word = "receptionist"
print("cosine similarity between " + word + " and g, before neutralizing: ",␣
↪cosine_similarity(word_to_vec_map[word], g))

e_debiased = neutralize(word, g_unit, word_to_vec_map_unit_vectors)
print("cosine similarity between " + word + " and g_unit, after neutralizing:␣
↪", cosine_similarity(e_debiased, g_unit))
```

```
cosine similarity between receptionist and g, before neutralizing:
0.30167259871100655
cosine similarity between receptionist and g_unit, after neutralizing:
-7.560738707307337e-17
```

[16]:
```
# In the above case, we observe that the second result is close to 0. (i.e␣
↪e_debiased and g_unit are almost orthogonal)
```

0.0.3 Equalization

[17]:
```
def equalize(pair, bias_axis, word_to_vec_map):
    """
    Debias gender specific words by following the equalize method described in␣
↪the figure above.

    Arguments:
    pair -- pair of strings of gender specific words to debias, e.g.␣
↪("actress", "actor")
    bias_axis -- numpy-array of shape (50,), vector corresponding to the bias␣
↪axis, e.g. gender
    word_to_vec_map -- dictionary mapping words to their corresponding vectors

    Returns
    e_1 -- word vector corresponding to the first word
    e_2 -- word vector corresponding to the second word
    """

    # Selecting word vector representation of "word" using word_to_vec_map
    w1, w2 = pair
    e_w1, e_w2 = word_to_vec_map[w1], word_to_vec_map[w2]

    # Computing the mean of e_w1 and e_w2
    mu = (e_w1 + e_w2)/2

    # Computing the projections of mu over the bias axis and the orthogonal axis
    mu_B = (np.dot(mu, bias_axis)/np.linalg.norm(bias_axis)) * (bias_axis) / np.
↪linalg.norm(bias_axis)
    mu_orth = mu - mu_B

    # Compute e_w1B and e_w2B (i.e The components of e_w1 and e_w2 along the␣
↪bias_axis)
    e_w1B = (np.dot(e_w1, bias_axis)/np.linalg.norm(bias_axis)) * (bias_axis) /␣
↪np.linalg.norm(bias_axis)
    e_w2B = (np.dot(e_w2, bias_axis)/np.linalg.norm(bias_axis)) * (bias_axis) /␣
↪np.linalg.norm(bias_axis)

    # Recentering the 'e_w1B' and 'e_w2B' such that 'corrected_e_w1B' and␣
↪'corrected_e_w2B' are equidistant from the
    # bias axis and hence equidistant from the 'neutralized' words.
    corrected_e_w1B = np.sqrt(1-(np.linalg.norm(mu_orth)**2)) * (e_w1B - mu_B)/
↪np.linalg.norm(e_w1B - mu_B)
    corrected_e_w2B = np.sqrt(1-(np.linalg.norm(mu_orth)**2)) * (e_w2B - mu_B)/
↪np.linalg.norm(e_w2B - mu_B)
```

```
    # Step 6: Debias by equalizing e1 and e2 to the sum of their corrected␣
↪projections (2 lines)
    e1 = corrected_e_w1B + mu_orth
    e2 = corrected_e_w2B + mu_orth

    ### END CODE HERE ###

    return e1, e2
```

[18]:
```
# Note

# After 'Neutralization', words like 'babysitter', 'nurse', 'doctor',␣
↪'homemaker' would lie on
# the 49-dimensional axis which is orthogonal to the bias axis.

# The 'corrected_e_w1B' and 'corrected_e_w2B' help in recentering the 'e_w1B'␣
↪and 'e_w2B' such that 'corrected_e_w1B' and
# 'corrected_e_w2B' are equidistant from the bias axis and hence equidistant␣
↪from the 'neutralized' words.

# The resultant 'e1' and 'e2' are such that they have undergone equalization (i.
↪e now they are equidistant from the bias axis)
# and their meanings have not changed.
```

[19]:
```
print("cosine similarities before equalizing:")
print("cosine_similarity(word_to_vec_map[\"man\"], gender) = ",␣
↪cosine_similarity(word_to_vec_map["man"], g))
print("cosine_similarity(word_to_vec_map[\"woman\"], gender) = ",␣
↪cosine_similarity(word_to_vec_map["woman"], g))
print()
e1, e2 = equalize(("man", "woman"), g_unit, word_to_vec_map_unit_vectors)
print("cosine similarities after equalizing:")
print("cosine_similarity(e1, gender) = ", cosine_similarity(e1, g_unit))
print("cosine_similarity(e2, gender) = ", cosine_similarity(e2, g_unit))
```

```
cosine similarities before equalizing:
cosine_similarity(word_to_vec_map["man"], gender) =  -0.024358754123475775
cosine_similarity(word_to_vec_map["woman"], gender) =  0.3979047171251496

cosine similarities after equalizing:
cosine_similarity(e1, gender) =  -0.24211016258888443
cosine_similarity(e2, gender) =  0.24211016258888443
```

[ ]:
```
# Now the magnitude of angles between the bias axis and the embedding vectors␣
↪'e1' and 'e2' are equal.
```

0.0.4 References

- The debiasing algorithm is from Bolukbasi et al., 2016, Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings
- The GloVe word embeddings were due to Jeffrey Pennington, Richard Socher, and Christopher D. Manning. (https://nlp.stanford.edu/projects/glove/)