Universidad Nacional del Altiplano

Facultad de Ingeniería Mecánica Eléctrica, Electrónica y Sistemas

Escuela Profesional de Ingeniería de Sistemas

PRÁCTICA DE LABORATORIO N°3 - TRIVIA

Curso

Tópicos Avanzados en Inteligencia Computacional

Estudiante

SALCCA LAGAR, Dany

Docente

Ing. RUELAS ACERO, Donia Alizandra

Enlace a la práctica de laboratorio 03:

https://colab.research.google.com/drive/1hdrl5obWsvKrPmC00TYHwiV7bhjqX6IN?usp=sharing

PRÁCTICA DE LABORATORIO Nº 03

Datos

- Apellidos y Nombres: Salcca Lagar, Dany
- Código: 191849

Librerias y Datos necesarios

Herramientas necesarias

In [18]:
```
import numpy as np
import matplotlib.pyplot as plt
import pickle
import pandas as pd
import math
```

Cargando el word embeddings

In [19]:
```
word_embeddings = pickle.load(open( "word_embeddings_subset.p", "rb" ) )
len(word_embeddings) #  243 palabras
```

Out[19]:
```
243
```

Cargando los datos con los cuales se validara el modelo

In [20]:
```
data = pd.read_csv('capitals.txt', delimiter=' ')
data.columns = ['capital1', 'pais1', 'capital2', 'pais2']

# imprimir los primeros cinco elementos en el DataFrame
data.head(5)
```

Out[20]:

	capital1	pais1	capital2	pais2
0	Athens	Greece	Bangkok	Thailand
1	Athens	Greece	Beijing	China
2	Athens	Greece	Berlin	Germany
3	Athens	Greece	Bern	Switzerland
4	Athens	Greece	Cairo	Egypt

Funciones Necesarios

Función de coseno similitud

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

In [21]:
```
def similitud_cos(vector1, vector2):

    similitud = -10

    dot = np.dot(vector1, vector2)
    norma = np.linalg.norm(vector1)
    normb = np.linalg.norm(vector2)
    similitud = dot / (norma * normb)

    return similitud
```

Prueba de la función

In [22]:
```
oil = word_embeddings['oil']
gas = word_embeddings['gas']

similitud_cos(oil, gas)
```

Out[22]: 0.7105981

Funcion de distancia euclidiana

$$d(\mathbf{A}, \mathbf{B}) = d(\mathbf{A}, \mathbf{B}) = \sqrt{(A_1 - B_1)^2 + (A_2 - B_2)^2 + \cdots + (A_n - B_n)^2}$$
$$= \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2}$$

In [23]:
```
def distancia_Eu(punto1, punto2):

    distancia_p = 0

    #calculando lo que esta dentro de la raiz de la funcion utilizada para el calculo de la distancia
    for i in range(len(punto1)):
        distancia_p += (punto1[i] - punto2[i])**2

    #sacando la raiz del calculo anterior
    distancia = math.sqrt(distancia_p)

    return distancia
```

Prueba de la función

In [24]:
```
distancia_Eu(oil, gas)
```

Out[24]: 2.188501083795935

Funcion para obtener el vector correspodiente de las palabras

In [25]:
```
def get_vectors(embeddings, words):

    # Obtener la longitud del vector de embeddings
    vector_length = len(list(embeddings.values())[0])

    # Crear una matriz numpy de ceros para almacenar los vectores de embeddings de las palabras
    vectors = np.zeros((len(words), vector_length))

    # Recorrer la lista de palabras y obtener los vectores de embeddings correspondientes
    for i, word in enumerate(words):
            vectors[i] = embeddings[word]

    return vectors
```

Modelo y Prueba del modelo

Modelo de predicción

In [26]:
```
def modelo_pas(cap1, pas1, cap2, embeddings):

    # almacenar la ciudad 1, el país 1 y la ciudad 2 en un conjunto llamado grupo
    group = set((cap1, pas1, cap2))

    # Obtener el vector de la capital 1
    cap1_emb = embeddings[cap1]

    # Obtener el vector del pais 1
    pas1_emb = embeddings[pas1]

    # Obtener el vector de la capital 2
    cap2_emb = embeddings[cap2]

    # obtener el vector del país 2 (es una combinación de los vectores del país 1, la capital 1 y la capita
    # Recuerda: King - Man + Woman = Queen
    vec = pas1_emb - cap1_emb + cap2_emb

    # Inicializar la similitud a -1 (será reemplazada por similitudes más cercanas a +1)
    similarity = -1

    # inicializar país una cadena vacía
    country = ''

    # recorrer todas las palabras del word_embeddings
    for word in embeddings.keys():

        # primero verifique que la palabra no esté ya en el 'grupo'
        if word not in group:

            # obtener el vector de la palabra del word_embeddings
            word_emb = embeddings[word]

            #  calcule la similitud coseno entre el vector del país 2 y el vector de las palabras del word_em
            cur_similarity = similitud_cos(vec, word_emb)

            # si la similitud coseno es más similar que la mejor similitud anterior
            if cur_similarity > similarity:

                # actualizar la similitud con la nueva y mejor similitud
                similarity = cur_similarity
```

```
            # almacenar el país como una tupla, que contiene la palabra y la similitud
            country = (word, similarity)

    return country
```

Prueba del modelo

Entrada: 1: Athens 2: Greece 3: Baghdad,
Salida: Su tarea es predecir el país 4: Iraq.

In [27]:
```
modelo_pas('Athens', 'Greece', 'Baghdad', word_embeddings)
```

Out[27]:
```
('Iraq', 0.63551915)
```

Acurracy y Prueba

Función para obtener el Acurracy

$$\text{Accuracy} = \frac{\text{Correct \# of predictions}}{\text{Total \# of predictions}}$$

In [28]:
```
def get_accuracy(word_embeddings, data):

    num_correct = 0

    # recorrer las filas del dataframe
    for i, row in data.iterrows():

        # Obteniendo la capital 1
        cap1 = row[0]

        # Obteniendo el pais de la capital 1
        pas1 = row[1]

        # Obteniendo la capital 2
        cap2 =  row[2]

        # Obteniendo el pais de la capital 2
        pas2 = row[3]

        # obtener el pais correspondiente (predicho)
        predicted_country2, _ = modelo_pas(cap1,pas1,cap2,word_embeddings)

        # verificando si el pais predicho es igual al pais verdadero
        if predicted_country2 == pas2:
            # incrementando en 1 el contador
            num_correct += 1

    # obtener el tamño del dataframe
    m = len(data)

    # calculando el accuracy de acuerdo a la funcion establecida
    accuracy = num_correct/m

    return accuracy
```

Prueba de la funcion

In [36]:
```
model_acc = get_accuracy(word_embeddings, data)
print("El Accuracy del modelo es: {:.2f}".format(model_acc))
```

El Accuracy del modelo es: 0.92

PCA y Prueba

La función PCA para reducir cada vector de palabra en 2 dimensiones

	1	2		299	300
Word 1	12245	2	...	0	625
Word 2	1345	2	53	5	4251
⋮	3654	13	352	1324	245
Word m	1029	22	24	2345	6254

=>

	1	2
Word 1	2134	4315
Word 2	756453	4253
⋮	43	2
Word m	2452	6541

In [30]:
```
def compute_pca(X, n_components=2):

    # media de toda la matriz
    X_demeaned = X - np.mean(X,axis=0)
    print('X_demeaned.shape: ',X_demeaned.shape)

    # calcular la matriz de covarianza
    covariance_matrix = np.cov(X_demeaned, rowvar=False)

    # calcular los vectores propios y los valores propios de la matriz de covarianza
    eigen_vals, eigen_vecs = np.linalg.eigh(covariance_matrix, UPLO='L')

    # ordenar el valor propio en orden creciente (obtener los índices de la ordenación)
    idx_sorted = np.argsort(eigen_vals)

    # invertir el orden para que sea de mayor a menor.
    idx_sorted_decreasing = idx_sorted[::-1]

    # ordenar los valores propios por idx_sorted_decreasing
    eigen_vals_sorted = eigen_vals[idx_sorted_decreasing]

    # ordenar vectores propios usando los índices idx_sorted_decreasing
    eigen_vecs_sorted = eigen_vecs[:,idx_sorted_decreasing]

    # seleccione los primeros n vectores propios (n es la dimensión deseada
    # de matriz de datos reescalados, o dims_rescaled_data)
    eigen_vecs_subset = eigen_vecs_sorted[:,0:n_components]

    # transformar los datos multiplicando la transposición de los vectores propios
    # con la transposición de los datos degradados
    # Tomando la transposición de ese producto.
    X_reduced = np.dot(eigen_vecs_subset.transpose(),X_demeaned.transpose()).transpose()

    return X_reduced
```

Probando la función

In [31]:
```
np.random.seed(1)
X = np.random.rand(3, 10)
```

```
X_reduced = compute_pca(X, n_components=2)
print("Your original matrix was " + str(X.shape) + " and it became:")
print(X_reduced)
```

```
X_demeaned.shape:  (3, 10)
Your original matrix was (3, 10) and it became:
[[ 0.43437323  0.49820384]
 [ 0.42077249 -0.50351448]
 [-0.85514571  0.00531064]]
```

Bloque de codigo para obtener el vector correspondiente de cada palabra

In [47]:
```
words = ['oil', 'gas', 'happy', 'sad', 'Austria', 'China',
         'petroleum', 'joyful', 'Peru', 'France']

# Obteniendo los vectores de las plabras dadas
X = get_vectors(word_embeddings, words)

print('You have 11 words each of 300 dimensions thus X.shape is:', X.shape)
```

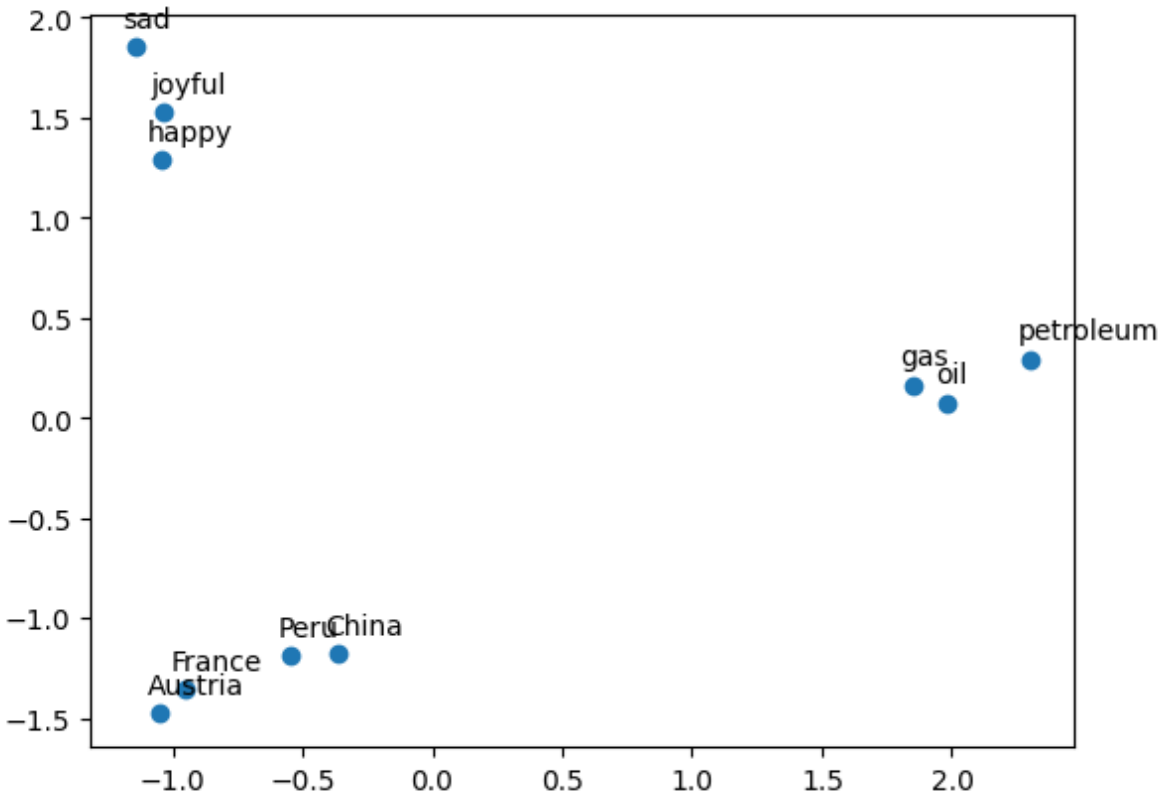
```
You have 11 words each of 300 dimensions thus X.shape is: (10, 300)
```

Visualizando los resultados (Graficando los wordembbeding usando PCA)

In [48]:
```
result = compute_pca(X, 2)
plt.scatter(result[:, 0], result[:, 1])
for i, word in enumerate(words):
    plt.annotate(word, xy=(result[i, 0] - 0.05, result[i, 1] + 0.1))

plt.show()
```

```
X_demeaned.shape:  (10, 300)
```



TRIVIA

In [34]:
```
import random
```

In [52]:
```
num = 0
puntaje = 0

while num < 5:

  #numero aleatorio para obtener los datos del capital.txt segun el numero de fila
  num_aleatorio = random.randint(0, len(data))
  fila = data.loc[num_aleatorio]

  #obteniendo los datos con los que se hara la pregunta
  cap1 = fila['capital1']
  pas1 = fila['pais1']
  cap2 = fila['capital2']
  pas2 = fila['pais2']
  pas_corr = ''

  #para hacer la pregunta
  pais_ingresado = input(f'\nSi {cap1} es capital de {pas1} entonces {cap2} es a: ')

  """
  pas_pre = modelo_pas(cap1, pas1, cap2, word_embeddings)

  if pas_pre == pas2:
    pas_corr = pas_pre
  else:
    pas_corr = pas2
  """

  #verificar si el pais ingresado es igual al pais que se verdadero
  if pais_ingresado == pas2:
    print('Correcto')
    puntaje += 20
  else:
    print('Incorrecto')

    print(f'El pais es: {pas2}')

  num += 1

print(f'\n\033[94mTU PUNTAJE ES {puntaje}  \033[0m')
```

```
Si Beirut es capital de Lebanon entonces Bujumbura es a: Burundi
Correcto

Si Apia es capital de Samoa entonces Berlin es a: Germany
Correcto

Si Paramaribo es capital de Suriname entonces Accra es a: Peru
Incorrecto
El pais es: Ghana

Si Banjul es capital de Gambia entonces Harare es a: Zimbabwe
Correcto

Si Minsk es capital de Belarus entonces Paramaribo es a: Asia
Incorrecto
El pais es: Suriname

TU PUNTAJE ES 60
```