Natural Language Processing Tutorial 1 - Static Word Embeddings with Word2Vec

What you'll learn in this tutorial

- We'll use the gensim library to:
  - explore pretrained word embeddings
  - pretrain our own embeddings
- We will additionally:
  - visualize word embeddings
  - evaluate them intrisically and extrinsically

Our schedule for today

- Part 1: Using pretrained word embeddings with gensim
  - How to download already pretrained embeddings
  - Nearest neighbour similarity search
  - Word embedding visualization via PCA
  - Intrisic evaluation with word analogy and word similarity benchmarks
  - **Task 1**
- Part 2: Pretraining your **own** embeddings
  - Training choices
  - Saving and loading your embeddings
- Part 3: Extrinsic evaluation of word embeddings
  - Using word2vec embeddings for spam classification
  - **Task 2**

Part 1 : Using pretrained embeddings with gensim

What is gensim?

- Gensim is one of many core NLP libraries:
  - together with NLTK, spaCy and HuggingFace 🤗
  - you can find its documentation here
- It can be used to deal with corpora and perform:
  - Retrieval
  - Topic Modelling
  - Representation Learning (**word2vec** and **doc2vec**)

In [ ]:
```
# Run this cell now!
import gensim

import numpy as np
import pandas as pd

import gensim.downloader as api
from gensim import utils
from gensim.models import KeyedVectors
from gensim.test.utils import datapath
from gensim.models import Word2Vec

from sklearn.metrics.pairwise import cosine_similarity
from sklearn.decomposition import PCA
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegressionCV
from sklearn.preprocessing import LabelEncoder

from scipy.stats import pearsonr, spearmanr

import nltk
from nltk.corpus import stopwords

import torch
import torch.nn as nn

import plotly.express as px
```

Let's download some embeddings!

In [ ]:
```
# Run this cell now!
word_emb = api.load('word2vec-google-news-300')
```

- The object that we get is of type KeyedVectors
- This is simply a map $w \rightarrow \mathbf{e}_w \in \mathbb{R}^{300}$
- You can explore here all the possible models or simply run `api.info()`

How do these embeddings look like?

In [ ]:
```
# Access embeddings with word-lookup
print(word_emb["apple"].shape)
print(word_emb["apple"])
```

```
(300,)
[-0.06445312 -0.16015625 -0.01208496  0.13476562 -0.22949219  0.16210938
  0.3046875  -0.1796875  -0.12109375  0.25390625 -0.01428223 -0.06396484
 -0.08056641 -0.05688477 -0.19628906  0.2890625  -0.05151367  0.14257812
 -0.10498047 -0.04736328 -0.34765625  0.35742188  0.265625    0.00188446
 -0.01586914  0.00195312 -0.35546875  0.22167969  0.05761719  0.15917969
  0.08691406 -0.0267334  -0.04785156  0.23925781 -0.05981445  0.0378418
  0.17382812 -0.41796875  0.2890625   0.32617188  0.02429199 -0.01647949
 -0.06494141 -0.08886719  0.07666016 -0.15136719  0.05249023 -0.04199219
 -0.05419922  0.00108337 -0.20117188  0.12304688  0.09228516  0.10449219
 -0.00408936 -0.04199219  0.01409912 -0.02111816 -0.13476562 -0.24316406
  0.16015625 -0.06689453 -0.08984375 -0.07177734 -0.00595093 -0.00482178
 -0.00089264 -0.30664062 -0.0625      0.07958984 -0.00909424 -0.04492188
  0.09960938 -0.33398438 -0.3984375   0.05541992 -0.06689453 -0.04467773
  0.11767578 -0.13964844 -0.26367188  0.17480469 -0.17382812 -0.40625
 -0.06738281 -0.07617188  0.09423828  0.20996094 -0.16308594 -0.08691406
 -0.0534668  -0.10351562 -0.07617188 -0.11083984 -0.03515625 -0.14941406
  0.0378418   0.38671875  0.14160156 -0.2890625  -0.16894531 -0.140625
 -0.04174805  0.22753906  0.24023438 -0.01599121 -0.06787109  0.21875
 -0.42382812 -0.5625     -0.49414062 -0.3359375   0.13378906  0.01141357
  0.13671875  0.0324707   0.06835938 -0.27539062 -0.15917969  0.00121307
  0.01208496 -0.0039978   0.00442505 -0.04541016  0.08642578  0.09960938
 -0.04296875 -0.11328125  0.13867188  0.41796875 -0.28320312 -0.07373047
 -0.11425781  0.08691406 -0.02148438  0.328125   -0.07373047 -0.01348877
  0.17773438 -0.02624512  0.13378906 -0.11132812 -0.12792969 -0.12792969
  0.18945312 -0.13867188  0.29882812 -0.07714844 -0.37695312 -0.10351562
  0.16992188 -0.10742188 -0.29882812  0.00866699 -0.27734375 -0.20996094
 -0.1796875  -0.19628906 -0.22167969  0.08886719 -0.27734375 -0.13964844
  0.15917969  0.03637695  0.03320312 -0.08105469  0.25390625 -0.08691406
 -0.21289062 -0.18945312 -0.22363281  0.06542969 -0.16601562  0.08837891
 -0.359375   -0.09863281  0.35546875 -0.00741577  0.19042969  0.16992188
 -0.06005859 -0.20605469  0.08105469  0.12988281 -0.01135254  0.33203125
 -0.08691406  0.27539062 -0.03271484  0.12011719 -0.0625      0.1953125
 -0.10986328 -0.11767578  0.20996094  0.19921875  0.02954102 -0.16015625
  0.00276184 -0.01367188  0.03442383 -0.19335938  0.00352478 -0.06542969
 -0.05566406  0.09423828  0.29296875  0.04052734 -0.09326172 -0.10107422
 -0.27539062  0.04394531 -0.07275391  0.13867188  0.02380371  0.13085938
  0.00236511 -0.2265625   0.34765625  0.13574219  0.05224609  0.18164062
  0.0402832   0.23730469 -0.16992188  0.10058594  0.03833008  0.10839844
 -0.05615234 -0.00946045  0.14550781 -0.30078125 -0.32226562  0.18847656
 -0.40234375 -0.3125     -0.08007812 -0.26757812  0.16699219  0.07324219
  0.06347656  0.06591797  0.17285156 -0.17773438  0.00276184 -0.05761719
 -0.2265625  -0.19628906  0.09667969  0.13769531 -0.49414062 -0.27929688
  0.12304688 -0.30078125  0.01293945 -0.1875     -0.20898438 -0.1796875
 -0.16015625 -0.03295898  0.00976562  0.25390625 -0.25195312  0.00210571
  0.04296875  0.01184082 -0.20605469  0.24804688 -0.203125   -0.17773438
  0.07275391  0.04541016  0.21679688 -0.2109375   0.14550781 -0.16210938
  0.20410156 -0.19628906 -0.35742188  0.35742188 -0.11962891  0.35742188
  0.10351562  0.07080078 -0.24707031 -0.10449219 -0.19238281  0.1484375
  0.00057983  0.296875   -0.12695312 -0.03979492  0.13183594 -0.16601562
  0.125       0.05126953 -0.14941406  0.13671875 -0.02075195  0.34375   ]
```

In [ ]:
```
# Access embeddings with index-lookup
print(word_emb[10])
```

```
[ 2.60009766e-02 -1.89208984e-03  1.85546875e-01 -5.17578125e-02
  5.12695312e-03 -1.09863281e-01 -8.17871094e-03 -8.83789062e-02
  9.66796875e-02  4.83398438e-02  1.10473633e-02 -3.63281250e-01
  8.20312500e-02 -2.12402344e-02  1.58203125e-01  4.41894531e-02
 -1.17797852e-02  2.12890625e-01 -5.73730469e-02  5.66406250e-02
 -1.07421875e-01  1.85546875e-01  7.71484375e-02  1.44958496e-04
  1.52343750e-01 -6.54296875e-02 -1.52343750e-01  2.25585938e-01
  8.10546875e-02  8.88671875e-02  7.32421875e-02 -1.03515625e-01
 -6.68945312e-02  1.76757812e-01  2.12890625e-01  1.40625000e-01
 -3.41796875e-02  1.78222656e-02  5.95703125e-02  2.86102295e-04
  5.88378906e-02  9.27734375e-03  1.66992188e-01 -2.70080566e-03
  1.15722656e-01  1.04492188e-01  5.37109375e-02  1.85546875e-02
  1.06445312e-01  5.05371094e-02 -1.64794922e-02 -1.27929688e-01
  2.16796875e-01  5.15136719e-02  4.78515625e-02  1.52343750e-01
  1.71875000e-01  7.86132812e-02 -5.88378906e-02 -4.29687500e-02
 -7.27539062e-02  1.81640625e-01 -8.05664062e-02 -1.54296875e-01
 -1.16699219e-01  8.44726562e-02 -6.17675781e-02 -4.51660156e-02
  9.21630859e-03  1.33789062e-01  1.92871094e-02  6.44531250e-02
  1.08886719e-01  1.58203125e-01 -2.35595703e-02  1.23535156e-01
  1.69921875e-01  3.49121094e-02  1.29882812e-01  2.65625000e-01
  1.93359375e-01 -8.83789062e-02  8.49609375e-02 -2.96630859e-02
  5.76171875e-02  2.51464844e-02 -1.01562500e-01  1.99218750e-01
  1.04492188e-01 -2.42919922e-02  2.01416016e-02 -3.51562500e-02
  6.64062500e-02 -6.20117188e-02  2.90527344e-02 -9.81445312e-02
 -1.81640625e-01  2.14843750e-01 -5.76171875e-02 -4.51660156e-02
  4.49218750e-02 -1.95312500e-02 -2.08984375e-01  1.19628906e-01
 -9.03320312e-02  5.07812500e-02  9.03320312e-03 -9.76562500e-02
 -7.86132812e-02 -1.36718750e-01 -1.13769531e-01 -5.64575195e-03
 -4.07714844e-02 -2.05993652e-03 -5.66406250e-02  3.64685059e-03
  8.30078125e-02 -7.08007812e-02  2.63671875e-01  1.24511719e-01
 -1.61132812e-02  9.13085938e-02 -2.39257812e-01 -1.04980469e-02
 -6.78710938e-02  1.40625000e-01  2.34375000e-01 -6.39648438e-02
  1.95312500e-01  5.02929688e-02 -1.25000000e-01  2.06298828e-02
 -1.19140625e-01 -1.17187500e-01 -9.01222229e-05  3.68652344e-02
  1.46484375e-01  2.47802734e-02 -1.49414062e-01  3.03649902e-03
 -3.10058594e-02  1.06933594e-01  2.55859375e-01 -6.00585938e-02
 -2.07031250e-01  1.58203125e-01 -2.15820312e-01 -1.84570312e-01
 -1.72851562e-01  7.99560547e-03 -3.03955078e-02  9.81445312e-02
  4.66918945e-03  2.57812500e-01  1.06933594e-01  1.26953125e-01
  6.34765625e-02 -1.30859375e-01  6.54296875e-02 -9.91210938e-02
  5.90820312e-02 -3.71093750e-02  1.01074219e-01  1.53320312e-01
 -1.53320312e-01 -7.56835938e-02  5.85937500e-02 -5.05371094e-02
  2.08007812e-01  4.85839844e-02 -9.42382812e-02 -9.71679688e-02
 -1.23046875e-01 -1.97265625e-01 -1.76757812e-01 -1.11328125e-01
  1.11328125e-01 -5.88378906e-02  2.27539062e-01  4.00390625e-02
  1.24511719e-01  1.47460938e-01  1.81884766e-02  4.05273438e-02
  1.69921875e-01  1.13769531e-01 -2.24609375e-02  6.73828125e-02
  8.59375000e-02  6.73828125e-02  2.06298828e-02  4.78515625e-02
  1.84326172e-02  2.05078125e-01 -4.68750000e-02  2.00195312e-01
 -1.56250000e-02 -1.40625000e-01  1.09863281e-02 -1.73828125e-01
  4.85839844e-02 -1.58203125e-01 -1.04492188e-01  3.63769531e-02
  3.01513672e-02  1.27929688e-01 -1.14257812e-01  1.41601562e-01
  2.34375000e-01 -8.98437500e-02 -1.02996826e-03 -1.50390625e-01
  1.79687500e-01  1.35742188e-01 -2.08007812e-01 -1.27563477e-02
  1.75781250e-01 -1.39648438e-01 -2.03125000e-01 -3.00292969e-02
 -2.78320312e-02 -6.50024414e-03  1.26953125e-01 -1.49414062e-01
  1.46484375e-01 -8.42285156e-03  1.12304688e-01  1.66015625e-01
 -1.57470703e-02  1.23046875e-01  7.22656250e-02 -4.37011719e-02
 -7.56835938e-02 -9.03320312e-02  1.01562500e-01 -1.44531250e-01
 -4.00390625e-02 -1.26953125e-02  2.66113281e-02 -7.81250000e-02
  3.56445312e-02  3.49121094e-02  1.79687500e-01 -1.38671875e-01
  2.80761719e-02 -2.86865234e-02  6.78710938e-02  7.03125000e-02
  9.57031250e-02  5.00488281e-02 -2.20947266e-02 -3.00781250e-01
  1.14257812e-01  7.51953125e-02  1.26342773e-02  1.32812500e-01
```

```
 2.52685547e-02  3.63769531e-02 -2.81982422e-02 -1.36718750e-01
 1.79687500e-01 -9.27734375e-02  8.49609375e-02  1.32812500e-01
 3.97949219e-02  4.29687500e-01 -1.87988281e-02 -1.47460938e-01
 6.10351562e-02  9.03320312e-02  8.69140625e-02 -6.88476562e-02
 1.10839844e-01  9.81445312e-02  1.50390625e-01  1.61132812e-01
-8.05664062e-02 -1.74804688e-01 -3.32031250e-02 -1.28906250e-01
 1.22558594e-01 -1.44653320e-02 -1.63085938e-01 -3.58886719e-02
 2.78320312e-02 -6.34765625e-02 -7.91015625e-02 -1.14746094e-01
 1.84326172e-02  2.91748047e-02 -3.00781250e-01 -4.58984375e-02
-1.74804688e-01  2.33398438e-01  2.25830078e-02  1.10351562e-01
-1.03515625e-01 -1.21582031e-01  2.21679688e-01 -2.19726562e-02]
```

Let's check the vocabulary

- Two important attributes:
  - `key_to_index` : maps a word to its vocabulary index
  - `index_to_key` : maps a vocabulary index to corresponding word

In [ ]:
```
print(f"Vocabulary length {len(word_emb.key_to_index)}")
print(f"Index of cat {word_emb.key_to_index['cat']}") # from word to index
print(f"Word at position 5947 {word_emb.index_to_key[5947]}") # from index to word
```

```
Vocabulary length 3000000
Index of cat 5947
Word at position 5947 cat
```

Compute similarity and distance

In [ ]:
```
pairs = [
    ('car', 'minivan'),
    ('car', 'bicycle'),
    ('car', 'airplane'),
    ('car', 'cereal'),
    ('car', 'communism'),
]
print("w1     w2    cos_sim   cos_dist")
for w1, w2 in pairs:
    print(f"{w1}  {w2}  {word_emb.similarity(w1, w2):.3f}  {word_emb.distance(w1, w
```

```
w1     w2    cos_sim   cos_dist
car  minivan  0.691  0.309
car  bicycle  0.536  0.464
car  airplane  0.424  0.576
car  cereal  0.139  0.861
car  communism  0.058  0.942
```

Nearest Neighbour (NN) Retrieval // Similarity Search

- gensim has a `most_similar` function:
  - however, it does not perform exhaustive nearest-neighbour search
  - given a query word w_q we want to find a ranked list L_q of words in vocabulary V in decreasing order of cosine similarity
  - e.g. w_q = "joy" then L_q = ["joy", "happiness", "smile", ...]
  - Sometimes we don't want to look for nearest neighbours in the entire vocabulary V:
    - we can restrict it to the top-r frequent words to have a much faster search
- We can write our own function!

In [ ]:
```python
def retrieve_most_similar(query_words, all_word_emb, restrict_vocab=10000):

    # Step 1: Get full or restricted vocabulary embeddings
    # If restrict_vocab=None then we have exhaustive search, otherwise we restrict
    vocab_emb = all_word_emb.vectors[:restrict_vocab+1,:] if restrict_vocab is not

    # Step 2: get the word embeddings for the query words
    query_emb = all_word_emb[query_words] # shape: |Q| x word_emb_size

    # Step 3: get cosine similarity between queries and embeddings
    cos_sim = cosine_similarity(query_emb, vocab_emb) # shape: |Q| x |V_r|

    # Step 4: sort similarities in desceding orders and get indices of nearest neig
    nn = np.argsort(-cos_sim) # shape: |Q| x |V_r|

    # Step 5: delete self-similarity, i.e. cos_sim(w,w)=1.0
    nn_filtered = nn[:, 1:] # remove self_similarity

    # Step 6: use the indices to get the words
    nn_words = np.array(word_emb.index_to_key)[nn_filtered]

    return nn_words
```

In [ ]:
```python
queries = ["king", "queen", "italy", "Italy", "nurse"]
res = retrieve_most_similar(queries, word_emb, restrict_vocab=100000)
top_k = 10
res_k = res[:, :top_k]
del res
print(res_k)
```

```
[['kings' 'queen' 'monarch' 'crown_prince' 'prince' 'sultan' 'ruler'
  'princes' 'throne' 'royal']
 ['queens' 'princess' 'king' 'monarch' 'Queen' 'princesses' 'royal'
  'prince' 'duchess' 'Queen_Elizabeth_II']
 ['french' 'Italy' 'i' 'haha' 'Cagliari' 'india' 'dont' 'thats' 'mr'
  'lol']
 ['Italian' 'Sicily' 'Italians' 'ITALY' 'Spain' 'Bologna' 'Italia'
  'France' 'Milan' 'Romania']
 ['registered_nurse' 'nurses' 'nurse_practitioner' 'midwife' 'Nurse'
  'nursing' 'doctor' 'medic' 'pharmacist' 'paramedic']]
```

Dimensionality Reduction and Plotting

- We want to plot our word embeddings
- But they "live" in $\mathbb{R}^{300}$
- Let's use dimensionality reduction techniques, like PCA

In [ ]:
```
all_res_words = res_k.flatten()
res_word_emb = word_emb[all_res_words]
print("(|Q| x k) x word_emb_size")
print(res_word_emb.shape)
```

```
(|Q| x k) x word_emb_size
(50, 300)
```

In [ ]:
```
pca = PCA(n_components=3) #Perform 3d-PCA
word_emb_pca = pca.fit_transform(res_word_emb)
```

In [ ]:
```
pca_df = pd.DataFrame(word_emb_pca, columns=["pca_x", "pca_y", "pca_z"])

pca_df["word"] = res_k.flatten()

labels = np.array([queries]).repeat(top_k)
pca_df["query"] = labels

print(pca_df.head())
```

```
      pca_x     pca_y     pca_z          word query
0 -0.951780 -0.588461  0.546893         kings  king
1 -1.366599 -0.059902  0.103550         queen  king
2 -2.038808 -0.398816 -0.404128       monarch  king
3 -1.730922 -0.289503 -0.157777  crown_prince  king
4 -1.596841 -0.419770 -0.252166        prince  king
```

In [ ]:
```
px.scatter_3d(pca_df, x='pca_x', y='pca_y', z='pca_z', color="query", text="word",
```

3d-PCA representation of word embeddings

Word embedding evaluation

- There are two main types of evaluation:
  - intrisic evaluation: evaluate word embeddings without a downstream task
    - word similarity benchmarks
    - word analogy benchmarks
  - extrinsic evaluation: evaluate word embeddings on a downstream task

Word Similarity Benchmarks

- Word similarity benchmarks, such as WS353, contain word pairs and a human-given similarity score

In [ ]:
```
ws353_df = pd.read_csv(datapath('wordsim353.tsv'), sep="\t", skiprows=1).rename(col
ws353_df.sample(5)
```

Out[ ]:

	Word 1	Word 2	Human (mean)
5	computer	internet	7.58
286	seafood	food	8.34
127	planet	star	8.45
263	profit	warning	3.88
304	environment	ecology	8.81

- To evaluate word embeddings, we need to do three steps:

1. For each pair (w_{i_1}, w_{i_2}) we get the embeddings $(\mathbf{e}_{w_{i_1}}, \mathbf{e}_{w_{i_2}})$
2. For each pair we compute the cosine similarity between its word embeddings $s_i = \cos(\mathbf{e}_{w_{i_1}}, \mathbf{e}_{w_{i_2}})$
3. We compute a correlation score (Pearson's r or Spearman's ρ) between the human given scores h_i and the cosine similarities s_i
   - the higher the score, the better!

Evaluating word similarity with gensim

- gensim allows us to do everything with the `evaluate_word_pairs` function

In [ ]:
```
word_emb.evaluate_word_pairs(datapath('wordsim353.tsv'), case_insensitive=False)
```

Out[ ]:
```
(PearsonRResult(statistic=0.6525349640723466, pvalue=3.3734155032900286e-44),
 SpearmanrResult(correlation=0.7000166486272194, pvalue=2.86866666051422e-53),
 0.0)
```

Word analogy benchmarks

- When doing word analogy resolution with word embeddings, we want to solve equations such as

  man : king = woman : x

- word2vec paper shows that word2vec embeddings can solve (some) of these equations by algebric operations:

1. Get $\mathbf{e}_x = \mathbf{e}_{king} - \mathbf{e}_{man} + \mathbf{e}_{woman}$
2. Check if $NN_V(\mathbf{e}_x) = \text{queen}$

Evaluating word analogies with gensim

- gensim provides us with a `most_similar` function
- It has several arguments, the most important are:
  - `positive` : list of words that should be summed together
  - `negative` : list of words that should be subtracted
- In formulas, this function computes:

$$\mathbf{e}_x = \sum_{i \in \text{pos}} \mathbf{e}_i - \sum_{i \in \text{neg}} \mathbf{e}\_i$$

- And then returns nearest neighbours of $\mathbf{e}_x$

In [ ]:
```
print(word_emb.most_similar(positive=["king", "woman"], negative=["man"], restrict_
print(word_emb.most_similar(positive=["iPod", "Sony"], negative=["Apple"], restrict
```

```
[('queen', 0.7118192911148071), ('monarch', 0.6189674735069275), ('princess', 0.5902
431011199951), ('crown_prince', 0.549946129322052), ('prince', 0.5377321243286133),
('kings', 0.5236843824386597), ('queens', 0.5181134343147278), ('sultan', 0.50985932
35015869), ('monarchy', 0.5087411403656006), ('royal_palace', 0.5087166428565979)]
[('Walkman', 0.581480860710144), ('MP3_player', 0.5763883590698242), ('MP3', 0.55208
24193954468), ('Panasonic', 0.5468560457229614), ('Blu_ray_disc', 0.543582856655120
8), ('JVC', 0.525976836681366), ('camcorder', 0.5257487297058105), ('Sony_PSP', 0.52
26278305053711), ('PlayStation_Portable', 0.5171500444412231), ('Blu_ray', 0.5171388
983726501)]
```

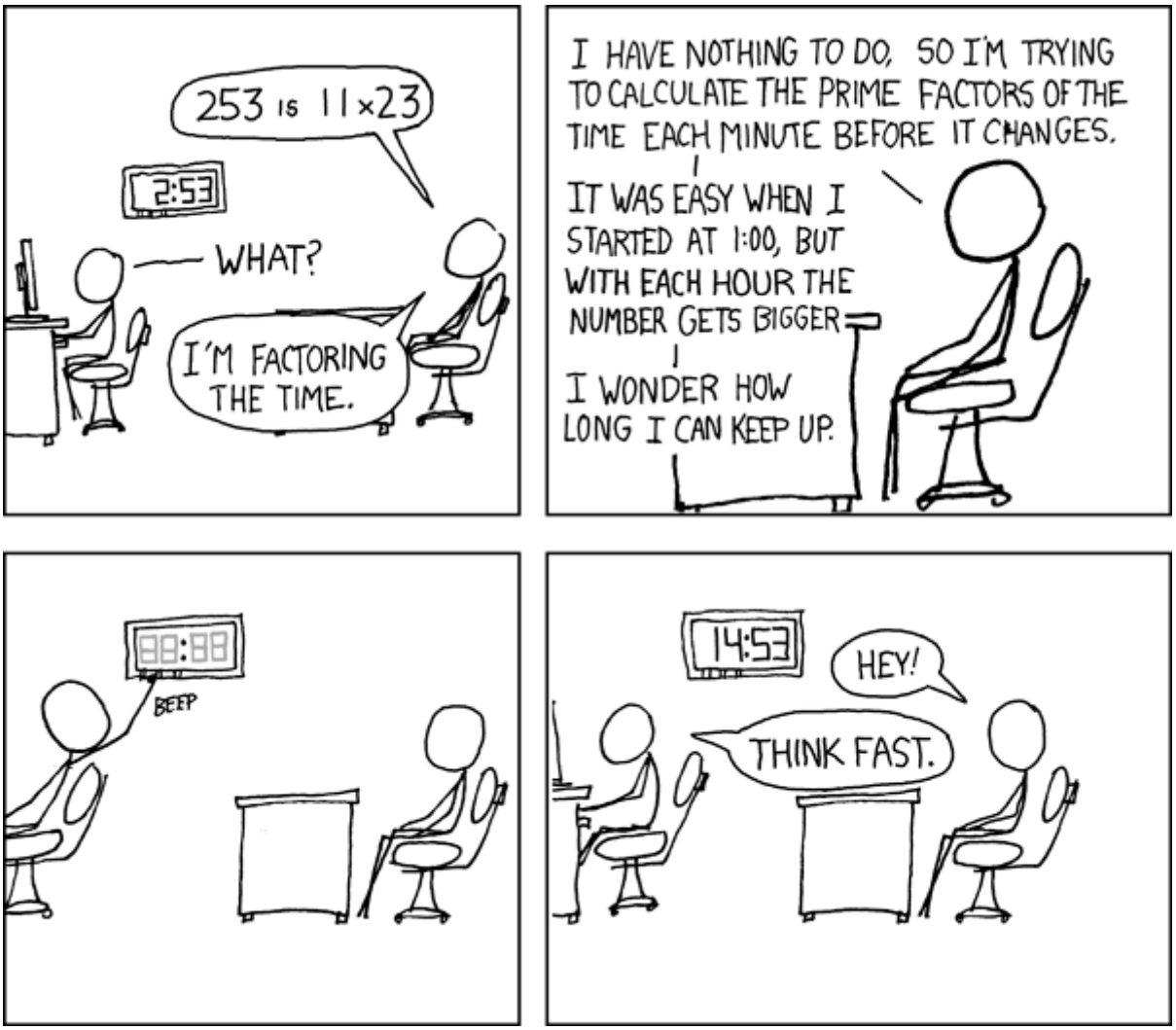
In [ ]:
```
f = open(datapath('questions-words.txt'))
print("".join(f.readlines()[:15]))
f.close()
```

```
: capital-common-countries
Athens Greece Baghdad Iraq
Athens Greece Bangkok Thailand
Athens Greece Beijing China
Athens Greece Berlin Germany
Athens Greece Bern Switzerland
Athens Greece Cairo Egypt
Athens Greece Canberra Australia
Athens Greece Hanoi Vietnam
Athens Greece Havana Cuba
Athens Greece Helsinki Finland
Athens Greece Islamabad Pakistan
Athens Greece Kabul Afghanistan
Athens Greece London England
Athens Greece Madrid Spain
```

In [ ]:
```
accuracy, results = word_emb.evaluate_word_analogies(datapath('questions-words.txt'
print(f"Accuracy {accuracy}")
print(results[0].keys())
print(f"Correct {results[0]['correct'][:5]}")
print(f"Incorrect {results[0]['incorrect'][:5]}")
```

```
Accuracy 0.7401448525607863
dict_keys(['section', 'correct', 'incorrect'])
Correct [('ATHENS', 'GREECE', 'BANGKOK', 'THAILAND'), ('ATHENS', 'GREECE', 'BEIJING
', 'CHINA'), ('ATHENS', 'GREECE', 'BERLIN', 'GERMANY'), ('ATHENS', 'GREECE', 'BERN',
'SWITZERLAND'), ('ATHENS', 'GREECE', 'CAIRO', 'EGYPT')]
Incorrect [('ATHENS', 'GREECE', 'BAGHDAD', 'IRAQ'), ('ATHENS', 'GREECE', 'HANOI', 'V
IETNAM'), ('ATHENS', 'GREECE', 'KABUL', 'AFGHANISTAN'), ('ATHENS', 'GREECE', 'LONDON
', 'ENGLAND'), ('BAGHDAD', 'IRAQ', 'BERN', 'SWITZERLAND')]
```

It's your turn! Go ahead with *Task 1.*



Task 1

Implement instrisic evaluation using wordsim353 benchamark. For computing correlations (step 3) use `spearmanr` and `pearsonr` from `scipy.stats`

In [ ]:
```
#TODO: implement WS353 evaluation benchmark in the three steps below.

# Step 0: (re)load the data

ws353_df = pd.read_csv(datapath('wordsim353.tsv'), sep="\t", skiprows=1).rename(col

# Step 1: Get embeddings (use ws353_df defined above)
w1 = word_emb[ws353_df["Word 1"]] # shape: |D| x word_emb_size
w2 = word_emb[ws353_df["Word 2"]] # shape: |D| x word_emb_size

# Step 2: Compute Cosine similarities
cos_mat = cosine_similarity(w1, w2) # shape: |D| x |D|
cos_pairs = np.diag(cos_mat) # shape |D|

# Step 3: Compute correlations
print(pearsonr(cos_pairs, ws353_df["Human (mean)"]))
print(spearmanr(cos_pairs, ws353_df["Human (mean)"]))
```

```
PearsonRResult(statistic=0.6525349483670781, pvalue=3.3734367166422075e-44)
SpearmanrResult(correlation=0.7000166486272194, pvalue=2.86866666051422e-53)
```

Part 2 : Pretraining your own embeddings

- Up to know we have used embeddings that someone else trained for us
- What if you want to pretrain your own embeddings for you domain or task of interest?
- The first thing we need is data!

In [ ]:
```
corpus = open(datapath('lee_background.cor'))
sample = corpus.readline()
print(sample, utils.simple_preprocess(sample))
```

```
Hundreds of people have been forced to vacate their homes in the Southern Highlands
of New South Wales as strong winds today pushed a huge bushfire towards the town of
Hill Top. A new blaze near Goulburn, south-west of Sydney, has forced the closure of
the Hume Highway. At about 4:00pm AEDT, a marked deterioration in the weather as a s
torm cell moved east across the Blue Mountains forced authorities to make a decision
to evacuate people from homes in outlying streets at Hill Top in the New South Wales
southern highlands. An estimated 500 residents have left their homes for nearby Mitt
agong. The New South Wales Rural Fire Service says the weather conditions which caus
ed the fire to burn in a finger formation have now eased and about 60 fire units in
and around Hill Top are optimistic of defending all properties. As more than 100 bla
zes burn on New Year's Eve in New South Wales, fire crews have been called to new fi
re at Gunning, south of Goulburn. While few details are available at this stage, fir
e authorities says it has closed the Hume Highway in both directions. Meanwhile, a n
ew fire in Sydney's west is no longer threatening properties in the Cranebrook area.
Rain has fallen in some parts of the Illawarra, Sydney, the Hunter Valley and the no
rth coast. But the Bureau of Meteorology's Claire Richards says the rain has done li
ttle to ease any of the hundred fires still burning across the state. "The falls hav
e been quite isolated in those areas and generally the falls have been less than abo
ut five millimetres," she said. "In some places really not significant at all, less
than a millimetre, so there hasn't been much relief as far as rain is concerned. "In
fact, they've probably hampered the efforts of the firefighters more because of the
wind gusts that are associated with those thunderstorms."
 ['hundreds', 'of', 'people', 'have', 'been', 'forced', 'to', 'vacate', 'their', 'ho
mes', 'in', 'the', 'southern', 'highlands', 'of', 'new', 'south', 'wales', 'as', 'st
rong', 'winds', 'today', 'pushed', 'huge', 'bushfire', 'towards', 'the', 'town', 'of
', 'hill', 'top', 'new', 'blaze', 'near', 'goulburn', 'south', 'west', 'of', 'sydney
', 'has', 'forced', 'the', 'closure', 'of', 'the', 'hume', 'highway', 'at', 'about',
'pm', 'aedt', 'marked', 'deterioration', 'in', 'the', 'weather', 'as', 'storm', 'cel
l', 'moved', 'east', 'across', 'the', 'blue', 'mountains', 'forced', 'authorities',
'to', 'make', 'decision', 'to', 'evacuate', 'people', 'from', 'homes', 'in', 'outlyi
ng', 'streets', 'at', 'hill', 'top', 'in', 'the', 'new', 'south', 'wales', 'southern
', 'highlands', 'an', 'estimated', 'residents', 'have', 'left', 'their', 'homes', 'f
or', 'nearby', 'mittagong', 'the', 'new', 'south', 'wales', 'rural', 'fire', 'servic
e', 'says', 'the', 'weather', 'conditions', 'which', 'caused', 'the', 'fire', 'to',
'burn', 'in', 'finger', 'formation', 'have', 'now', 'eased', 'and', 'about', 'fire',
'units', 'in', 'and', 'around', 'hill', 'top', 'are', 'optimistic', 'of', 'defending
', 'all', 'properties', 'as', 'more', 'than', 'blazes', 'burn', 'on', 'new', 'year',
'eve', 'in', 'new', 'south', 'wales', 'fire', 'crews', 'have', 'been', 'called', 'to
', 'new', 'fire', 'at', 'gunning', 'south', 'of', 'goulburn', 'while', 'few', 'detai
ls', 'are', 'available', 'at', 'this', 'stage', 'fire', 'authorities', 'says', 'it',
'has', 'closed', 'the', 'hume', 'highway', 'in', 'both', 'directions', 'meanwhile',
'new', 'fire', 'in', 'sydney', 'west', 'is', 'no', 'longer', 'threatening', 'propert
ies', 'in', 'the', 'cranebrook', 'area', 'rain', 'has', 'fallen', 'in', 'some', 'par
ts', 'of', 'the', 'illawarra', 'sydney', 'the', 'hunter', 'valley', 'and', 'the', 'n
orth', 'coast', 'but', 'the', 'bureau', 'of', 'meteorology', 'claire', 'richards', '
says', 'the', 'rain', 'has', 'done', 'little', 'to', 'ease', 'any', 'of', 'the', 'hu
ndred', 'fires', 'still', 'burning', 'across', 'the', 'state', 'the', 'falls', 'have
', 'been', 'quite', 'isolated', 'in', 'those', 'areas', 'and', 'generally', 'the', '
falls', 'have', 'been', 'less', 'than', 'about', 'five', 'millimetres', 'she', 'said
', 'in', 'some', 'places', 'really', 'not', 'significant', 'at', 'all', 'less', 'tha
n', 'millimetre', 'so', 'there', 'hasn', 'been', 'much', 'relief', 'as', 'far', 'as
', 'rain', 'is', 'concerned', 'in', 'fact', 'they', 've', 'probably', 'hampered', 't
he', 'efforts', 'of', 'the', 'firefighters', 'more', 'because', 'of', 'the', 'wind',
'gusts', 'that', 'are', 'associated', 'with', 'those', 'thunderstorms']
```

In [ ]:

```python
class MyCorpus:
    """An iterator that yields sentences (lists of str)."""

    def __iter__(self):
        corpus_path = datapath('lee_background.cor')
        for line in open(corpus_path):
            # assume there's one document per line, tokens separated by whitespace
            yield utils.simple_preprocess(line)
```

Let's pretrain our own embeddings

- We will use the `Word2Vec` class from `gensim.models`
- Let's look at the most important parameters

In [ ]:
```
# Vocabulary building + training
model = Word2Vec(sentences=MyCorpus(),
                                 min_count=3, # ignore all words with freq < min_coun
                                 vector_size=200, # dimensionality of the vectors
                                 sg=1, # 1 for skip-gram, 0 for CBOW
                                 epochs=10, # num_epochs
                                 alpha=0.025, # initial learning rate
                                 batch_words=10000, # batch size
                                 window=5, # window size for context words
                                 negative=10, # number of negatives for negative samp
                                 ns_exponent=0.75 # exponent of the sampling distribu
                                )
print(model)
word_emb_lee = model.wv # wv attribute contains word embeddings
```

```
Word2Vec<vocab=2747, vector_size=200, alpha=0.025>
```

Saving and loading your embeddings

- Saving or loading the full model (i.e. embeddings + hyperparameters)
  - This allows to resume training

In [ ]:
```
save_path = "word2vee_lee.model"
model.save(save_path)
model_reloaded = Word2Vec.load(save_path)
```

- Saving or loading **only** word embeddings
  - This does **NOT** allow to resume training

In [ ]:
```
save_path = "word2vee_lee.emb"
model.wv.save(save_path)
emb_reloaded = KeyedVectors.load(save_path)
```

Part 3 : Extrinsic evaluation of word embeddings

- Up to know we have evaluated words embeddings intrisically
- Let's try to see how they fare in a real world task
- We will use them to solve a spam classification task

- Please now run all cells below!
  - You will need them for **Task 2**
- Remember to put `SMSSpamCollection.tsv` in the same folder as this notebook
  - Or upload it if you're using Colab

In [ ]:
```
spam_df = pd.read_csv("SMSSpamCollection.tsv", sep="\t", header=None, names=["label
spam_df
```

Out[ ]:

	label	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ü b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

In [ ]:
```
# Let's do one-hot encoding of the labels
label_encoder = LabelEncoder()
spam_df["label"] = label_encoder.fit_transform(spam_df["label"])
spam_df
```

Out[ ]:

	label	text
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	1	This is the 2nd time we have tried 2 contact u...
5568	0	Will ü b going to esplanade fr home?
5569	0	Pity, * was in mood for that. So...any other s...
5570	0	The guy did some bitching but I acted like i'd...
5571	0	Rofl. Its true to its name

5572 rows × 2 columns

Building a classification model

- We want to use a standard ML approach
- We will first preprocess the text:
  - lowercasing
  - tokenization
  - stopword removal
- After this, we will create a sentence embedding of each SMS as the average of word embeddings in that sentence

In [ ]:
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Tommaso\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[ ]: True

In [ ]:
```
# lowercase, tokenize and stopword removal
stop_words = set(stopwords.words('english'))
spam_df["preprocessed_text"] = spam_df["text"].apply(lambda sentence: [word for wor
spam_df
```

Out[ ]:

	label	text	preprocessed_text
0	0	Go until jurong point, crazy.. Available only ...	[go, jurong, point, crazy, available, bugis, g...
1	0	Ok lar... Joking wif u oni...	[ok, lar, joking, wif, oni]
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	[free, entry, wkly, comp, win, fa, cup, final,...
3	0	U dun say so early hor... U c already then say...	[dun, say, early, hor, already, say]
4	0	Nah I don't think he goes to usf, he lives aro...	[nah, think, goes, usf, lives, around, though]
...	...	...	...
5567	1	This is the 2nd time we have tried 2 contact u...	[nd, time, tried, contact, pound, prize, claim...
5568	0	Will ü b going to esplanade fr home?	[going, esplanade, fr, home]
5569	0	Pity, * was in mood for that. So...any other s...	[pity, mood, suggestions]
5570	0	The guy did some bitching but I acted like i'd...	[guy, bitching, acted, like, interested, buyin...
5571	0	Rofl. Its true to its name	[rofl, true, name]

5572 rows × 3 columns

In [ ]:
```
# Create sentence embeddings
spam_df["sent_emb"] = spam_df["preprocessed_text"].apply(lambda tok_sentence: np.me
```

```
C:\Users\Tommaso\anaconda3\envs\pd_nlp\lib\site-packages\numpy\core\fromnumeric.py:3
432: RuntimeWarning:

Mean of empty slice.
```

In [ ]:
```
spam_df = spam_df.dropna()
```

In [ ]:
```
all_features = spam_df.drop(columns="label")
features_train, features_test, y_train, y_test = train_test_split(all_features , sp
```

In [ ]:
```
print(features_train.shape, features_test.shape)
```

```
(4444, 3) (1111, 3)
```

Logistic regression classifier on top of these sentence embeddings

In [ ]:
```
logreg_model = LogisticRegressionCV(Cs=10, cv=5, penalty='l2', max_iter=1000)
sent_emb_train = np.stack(features_train["sent_emb"]) # shape: train_size x 300
logreg_model.fit(sent_emb_train, y_train) # 5-fold GridSearchCV followed by trainin
```

Out[ ]:
```
▾          LogisticRegressionCV

LogisticRegressionCV(cv=5, max_iter=1000)
```

In [ ]:
```
sent_emb_test = np.stack(features_test["sent_emb"])
print(f"Accuracy of the model {logreg_model.score(sent_emb_test, y_test)}")
```

```
Accuracy of the model 0.9567956795679567
```

In [ ]:
```
print(classification_report(y_test, logreg_model.predict(sent_emb_test), target_nam
```

```
              precision    recall  f1-score   support

         ham       0.97      0.98      0.98       962
        spam       0.87      0.79      0.83       149

    accuracy                           0.96      1111
   macro avg       0.92      0.89      0.90      1111
weighted avg       0.96      0.96      0.96      1111
```

Sneak-peek: using gensim embeddings in PyTorch

In [ ]:
```
embs = nn.Embedding( len(word_emb.key_to_index), 300).from_pretrained(torch.from_nu
idx = torch.LongTensor([word_emb.key_to_index[word] for word in ["soccer", "tennis"
embs(idx)
```

Out[ ]:
```
tensor([[-7.6660e-02,  1.1035e-01,  3.5352e-01, -7.9102e-02, -5.0049e-02,
         -2.9688e-01,  1.0938e-01, -3.5938e-01, -8.7402e-02, -7.0312e-02,
          2.0801e-01, -2.4512e-01, -5.5664e-02,  2.4219e-01,  2.3560e-02,
         -8.6670e-03,  2.6855e-02,  4.0234e-01,  1.7480e-01, -1.6602e-02,
         -2.0410e-01,  5.0000e-01, -8.9844e-02, -1.4355e-01,  5.5420e-02,
          9.0820e-02,  1.1426e-01,  1.5430e-01,  1.3477e-01,  2.2656e-01,
          4.4189e-02,  3.7109e-02, -1.1621e-01, -1.1328e-01, -1.6479e-02,
         -1.2695e-01,  2.9883e-01, -1.2598e-01,  1.0303e-01,  3.1641e-01,
          1.2665e-03, -8.8379e-02,  1.2695e-01,  1.5820e-01, -7.1777e-02,
         -2.1094e-01,  3.1641e-01, -2.0801e-01,  9.6893e-04,  3.2422e-01,
          7.1289e-02,  7.1289e-02, -9.2285e-02, -2.2705e-02,  9.5703e-02,
         -2.9883e-01, -6.5918e-02, -7.0801e-02, -7.4219e-02, -2.3535e-01,
         -2.8320e-01, -2.0215e-01, -1.6211e-01,  2.2070e-01, -2.3682e-02,
         -1.0645e-01,  1.9653e-02, -5.9082e-02,  2.3730e-01,  2.9785e-02,
          2.8711e-01,  1.0889e-01, -1.9141e-01, -5.2246e-02,  1.4746e-01,
          2.3730e-01,  4.1016e-02, -5.9326e-02,  2.0508e-01, -1.5430e-01,
         -2.2461e-01,  3.8330e-02, -8.1055e-02,  2.1387e-01, -1.7944e-02,
         -1.9043e-01, -1.2402e-01,  2.7539e-01,  1.0681e-02, -6.2500e-02,
          8.5449e-02, -3.0859e-01,  3.6621e-02,  6.9336e-02,  1.3184e-01,
          7.4219e-02,  1.6016e-01, -5.9326e-02,  1.2634e-02,  2.6758e-01,
         -3.5706e-03,  1.0303e-01,  2.9297e-01,  1.1816e-01,  2.0117e-01,
          4.8047e-01,  2.1680e-01, -3.2422e-01, -1.7188e-01, -1.6016e-01,
          1.4746e-01, -6.4392e-03,  2.0264e-02, -1.0791e-01,  9.9121e-02,
          2.7734e-01, -4.3945e-02,  2.3438e-01, -2.9175e-02, -1.3574e-01,
         -5.1514e-02, -3.0664e-01,  2.4048e-02, -5.4688e-02, -1.9727e-01,
         -1.2695e-01, -3.4375e-01, -1.0742e-02, -1.7285e-01,  4.0771e-02,
          2.7588e-02, -3.2227e-01, -1.4258e-01, -1.6504e-01, -3.8452e-03,
         -1.0889e-01, -2.4707e-01,  1.5430e-01, -1.1475e-02, -3.2501e-03,
         -2.2339e-02, -1.1035e-01,  3.4961e-01, -5.8105e-02,  1.7773e-01,
          3.1055e-01,  1.8066e-01, -6.1340e-03,  9.1553e-03, -3.2812e-01,
          1.7480e-01,  2.2266e-01, -2.7344e-01, -1.6602e-01, -2.8711e-01,
          2.5977e-01, -5.8350e-02, -3.7354e-02, -9.0332e-02, -8.3008e-02,
         -6.2500e-02,  4.1260e-02,  8.4839e-03, -5.8105e-02,  1.0681e-02,
         -4.8218e-03,  1.1182e-01, -3.0078e-01, -9.2773e-02, -6.4453e-02,
         -9.5215e-02,  2.8320e-01, -1.4453e-01, -1.6406e-01,  1.3574e-01,
         -4.3945e-02,  1.7188e-01, -2.1606e-02, -8.1055e-02,  8.8867e-02,
          1.3281e-01, -1.5137e-02, -2.3242e-01,  1.8555e-02,  4.3945e-02,
          3.7500e-01,  7.6172e-02, -9.7656e-02,  1.0681e-03,  1.3672e-01,
          2.5000e-01, -6.1768e-02,  6.2012e-02,  1.6309e-01, -1.9434e-01,
          7.9590e-02,  2.2461e-01,  1.8555e-01, -2.5195e-01, -9.5215e-02,
         -2.0508e-01, -1.3184e-01, -5.0293e-02, -3.0664e-01,  5.5176e-02,
         -5.7812e-01,  4.9561e-02, -6.7383e-02, -2.1777e-01, -2.1851e-02,
         -1.3574e-01,  1.1182e-01,  6.7383e-02,  1.7871e-01, -2.3828e-01,
          1.3184e-01,  9.0942e-03, -2.5024e-03, -3.2812e-01, -7.0312e-02,
         -1.1719e-01, -2.7734e-01, -4.1809e-03, -1.0352e-01, -1.6968e-02,
          1.4648e-01, -2.2705e-02,  2.4292e-02,  1.2500e-01, -2.3438e-02,
          9.7656e-02,  2.2827e-02, -1.1572e-01,  5.8350e-02,  1.7969e-01,
         -7.9346e-03,  1.2817e-02, -2.2363e-01,  1.0254e-01, -2.2363e-01,
          5.2979e-02, -2.3926e-01,  1.5137e-01,  7.9102e-02, -4.7363e-02,
          7.1411e-03, -2.7734e-01, -1.0938e-01, -4.1211e-01,  5.7373e-02,
          2.0117e-01,  8.4961e-02,  1.5918e-01,  2.2949e-01,  2.5391e-01,
          1.3867e-01,  2.7100e-02, -2.1875e-01, -2.3828e-01,  2.3535e-01,
          5.8594e-02, -1.3770e-01,  1.1670e-01,  1.7188e-01, -5.0049e-03,
          2.7344e-01, -2.9492e-01,  1.6406e-01,  1.0071e-02,  1.5039e-01,
         -3.3398e-01,  1.3965e-01, -1.5234e-01, -1.3672e-01,  1.8652e-01,
         -3.1836e-01, -4.3213e-02, -1.2207e-01,  1.9824e-01, -1.1328e-01,
         -9.2285e-02,  5.9814e-02,  9.0332e-02,  9.8267e-03, -1.2793e-01,
         -2.3828e-01, -1.7188e-01,  1.3281e-01,  1.2158e-01,  1.4160e-01,
         -1.3281e-01, -1.1621e-01, -2.2949e-02, -1.1670e-01,  2.1289e-01,
          3.7891e-01,  2.5391e-01,  6.2500e-02, -1.8359e-01,  2.6562e-01],
        [-2.8906e-01, -8.6426e-02,  2.3145e-01,  3.6377e-02,  4.7363e-02,
         -5.6885e-02,  9.4727e-02, -9.2773e-02, -1.7944e-02,  1.8848e-01,
          4.0039e-02, -2.6245e-02,  2.4902e-01,  7.2754e-02,  2.5513e-02,
          3.8818e-02,  1.3770e-01,  4.2383e-01,  2.6953e-01,  1.0437e-02,
```

```
-1.3379e-01,  4.7461e-01, -2.5000e-01, -1.0547e-01, -1.0156e-01,
-1.9238e-01, -8.6914e-02,  2.5586e-01,  1.0840e-01,  4.8096e-02,
-6.1035e-03,  2.8801e-04, -1.6406e-01, -2.9907e-02, -2.3071e-02,
-2.3926e-01,  1.5918e-01, -1.0986e-01,  1.3184e-01,  9.9609e-02,
 1.4551e-01, -9.4727e-02,  1.1035e-01,  1.2305e-01,  5.1025e-02,
-3.4766e-01,  2.8320e-01, -1.8750e-01,  7.5195e-02,  4.5703e-01,
 2.2754e-01,  2.1387e-01, -2.8125e-01, -1.3965e-01,  8.4473e-02,
-2.5391e-01, -1.6602e-01, -2.5781e-01, -2.4414e-02, -2.2070e-01,
-1.6016e-01,  5.3955e-02, -2.4414e-01, -1.8188e-02,  1.6724e-02,
 2.9297e-01,  3.4570e-01,  2.0996e-01,  2.2266e-01, -2.6367e-02,
 3.5547e-01,  1.8677e-02,  7.7148e-02,  2.7710e-02,  1.2878e-02,
 1.7090e-01, -1.3086e-01, -2.5391e-01, -3.0078e-01, -1.0693e-01,
-3.3984e-01, -3.5400e-02,  1.6309e-01,  5.0049e-02, -1.7578e-01,
 1.3867e-01, -2.9297e-01,  2.1289e-01,  9.1309e-02, -1.7578e-01,
-1.1572e-01, -6.1035e-02, -4.5654e-02,  1.6699e-01, -3.5400e-02,
 7.8125e-02,  3.4570e-01, -1.2109e-01, -3.4766e-01, -5.2490e-02,
-2.7148e-01,  3.8281e-01,  2.3828e-01,  2.3438e-02,  1.4551e-01,
 4.5508e-01,  2.9297e-01, -4.0771e-02, -2.3535e-01,  2.1240e-02,
-3.6133e-02, -1.0645e-01, -1.6724e-02,  1.3574e-01,  9.7656e-02,
 3.8867e-01,  1.2793e-01, -4.9805e-02, -5.3955e-02, -2.1094e-01,
-2.2949e-01, -2.1118e-02,  1.9141e-01, -2.5391e-02,  4.6875e-02,
-1.6211e-01, -1.1865e-01,  7.4158e-03,  1.9287e-02,  2.2095e-02,
 9.3262e-02, -1.7969e-01, -3.0664e-01,  2.0312e-01, -1.8555e-02,
-2.1289e-01,  5.3406e-03,  1.7969e-01, -3.7109e-01,  7.1289e-02,
-4.2480e-02, -2.7148e-01,  2.0605e-01, -2.9492e-01, -1.1230e-02,
 5.3906e-01,  2.5195e-01, -1.7773e-01, -9.4727e-02, -3.5352e-01,
 1.3477e-01,  2.1484e-01, -4.5117e-01, -1.1572e-01, -2.2168e-01,
 9.1309e-02,  6.2500e-02, -7.6172e-02, -6.4453e-02, -3.3398e-01,
-2.1777e-01,  2.2949e-02,  8.9844e-02, -2.6245e-02,  1.9653e-02,
-2.5586e-01,  1.9727e-01, -1.3281e-01, -4.3457e-02, -8.4961e-02,
 5.2490e-03,  2.1777e-01, -4.1260e-02,  5.4443e-02,  9.9609e-02,
-1.8652e-01,  3.5938e-01,  1.9727e-01,  8.9111e-03, -1.6602e-01,
 7.3242e-02,  2.3926e-01, -4.3359e-01, -4.3701e-02,  2.1191e-01,
 1.7773e-01,  1.0596e-01,  1.7188e-01,  1.8945e-01,  3.6377e-02,
 1.3867e-01,  9.3994e-03,  1.2988e-01,  1.8359e-01, -1.3672e-01,
 2.4316e-01,  3.1250e-02, -5.5420e-02,  1.4746e-01, -1.4160e-01,
-3.9258e-01, -1.8066e-02, -2.0898e-01, -3.0469e-01, -2.6953e-01,
-4.1016e-01,  3.6316e-03,  5.0781e-02, -2.1191e-01, -3.5889e-02,
 1.8555e-02,  2.5000e-01,  5.5908e-02,  1.1780e-02,  1.3281e-01,
-3.3875e-03, -1.0303e-01,  9.7656e-02,  2.8534e-03, -1.5430e-01,
-2.2461e-01, -3.2422e-01, -2.8198e-02, -5.1758e-02, -1.2256e-01,
-2.4170e-02,  1.4453e-01, -1.2354e-01,  6.5918e-02,  2.2339e-02,
 1.1182e-01, -4.9805e-02,  2.0996e-02, -1.4648e-01,  6.8848e-02,
-2.7832e-02,  1.3574e-01, -1.5820e-01,  1.9727e-01, -7.9956e-03,
 5.8594e-02, -2.3145e-01,  4.9805e-01,  1.9897e-02,  8.2520e-02,
-3.0151e-02, -2.8320e-01, -1.6797e-01, -1.2402e-01,  6.3965e-02,
 2.4902e-01,  1.5234e-01,  5.4688e-02,  2.0020e-01,  1.5918e-01,
 3.0078e-01,  2.2559e-01, -3.5645e-02, -2.6758e-01,  2.8320e-01,
 3.3203e-01,  1.8799e-02,  4.2236e-02, -1.4160e-01, -8.5449e-02,
 4.1992e-01, -1.1475e-01,  4.6143e-02,  8.3984e-02,  1.4453e-01,
-7.3730e-02,  3.7891e-01, -1.8555e-01,  3.0151e-02,  1.7090e-01,
-5.3223e-02,  1.2793e-01, -2.4414e-01,  2.1680e-01, -8.8501e-03,
 5.8594e-02,  1.8945e-01,  2.2754e-01, -1.6699e-01, -3.7354e-02,
-3.3447e-02, -3.6523e-01,  1.2891e-01,  5.6458e-04,  4.1016e-01,
-2.3242e-01,  1.1816e-01, -6.7871e-02, -2.2656e-01,  2.6562e-01,
 1.4941e-01,  6.5918e-02,  1.3965e-01, -1.8066e-01,  8.1543e-02],
[-9.7656e-02,  3.1982e-02,  2.5781e-01, -4.1504e-02,  1.0156e-01,
-1.0059e-01,  1.4648e-01, -1.9922e-01,  1.5332e-01,  6.3477e-02,
 8.3984e-02, -3.0078e-01,  6.3477e-02,  2.0898e-01, -2.1191e-01,
 1.8848e-01, -8.3496e-02,  3.2812e-01,  2.7930e-01, -1.4062e-01,
-1.6895e-01,  2.0410e-01,  4.9072e-02, -6.9885e-03,  9.4238e-02,
 9.8419e-04,  3.1250e-02,  2.4805e-01,  3.3594e-01,  2.6367e-01,
 5.6885e-02,  3.0469e-01,  1.2158e-01, -1.9727e-01,  1.7212e-02,
 9.9609e-02,  2.2754e-01, -1.2061e-01,  1.2354e-01,  3.7891e-01,
```

```
 2.3682e-02, -1.8652e-01,  6.2988e-02,  1.5234e-01,  3.7354e-02,
-1.6992e-01,  1.0645e-01, -4.9805e-02, -6.2012e-02,  1.6895e-01,
 4.4189e-02,  2.7832e-02, -1.1084e-01,  4.4922e-02,  2.7832e-02,
-4.4531e-01,  3.4912e-02, -6.2256e-02, -3.9307e-02, -2.0117e-01,
-3.0469e-01, -1.0059e-01, -1.6406e-01,  1.5234e-01,  1.1035e-01,
-1.5332e-01, -7.1289e-02,  7.9590e-02,  1.8750e-01,  6.8848e-02,
 2.4414e-01, -6.5613e-04, -1.9141e-01,  3.4912e-02,  1.9775e-02,
-5.4199e-02,  3.3203e-02, -2.0801e-01,  9.8633e-02, -1.9043e-01,
-6.0791e-02, -2.0703e-01, -2.1851e-02,  8.7891e-02,  2.0898e-01,
-2.3633e-01, -9.1797e-02,  2.2656e-01, -3.9307e-02,  9.1309e-02,
 1.1353e-02, -1.5527e-01,  6.7871e-02, -4.9072e-02,  2.5177e-03,
-4.6631e-02,  9.1797e-02,  1.0596e-01,  2.1094e-01,  4.2480e-02,
-4.9561e-02,  1.7676e-01,  3.3203e-01, -4.2236e-02,  2.0312e-01,
 2.9883e-01,  1.2109e-01, -4.8584e-02, -1.4160e-01, -2.5195e-01,
-2.2070e-01,  2.2363e-01,  2.2217e-02,  1.0938e-01,  3.1445e-01,
 3.7109e-01, -4.8340e-02,  2.7734e-01,  1.2756e-02, -9.6191e-02,
-2.0312e-01, -1.5527e-01,  1.1035e-01, -6.5430e-02, -2.7539e-01,
-2.3438e-01, -3.7891e-01, -8.3008e-02, -1.2500e-01, -1.4062e-01,
-9.2773e-03, -4.2188e-01, -1.1719e-01, -2.4121e-01,  3.4424e-02,
 5.3406e-03, -2.9883e-01,  3.4570e-01, -6.5430e-02,  5.5420e-02,
-5.1758e-02, -1.1279e-01,  2.0117e-01,  2.9785e-02,  1.0547e-01,
 2.7539e-01,  1.0205e-01, -1.6699e-01, -7.0801e-02, -2.9688e-01,
 1.5039e-01,  2.9492e-01, -1.9727e-01,  4.8584e-02, -4.2773e-01,
 1.5564e-02,  1.2061e-01, -1.7090e-01,  6.6895e-02, -1.5039e-01,
-5.2734e-02,  6.4941e-02,  3.7842e-02,  1.7456e-02,  1.4453e-01,
-1.4746e-01,  1.0742e-01, -1.9629e-01, -5.2002e-02, -6.2988e-02,
-1.7188e-01,  2.2583e-02, -5.2795e-03, -1.2256e-01, -6.2500e-02,
 8.8501e-04,  1.8433e-02,  1.1279e-01, -1.1133e-01,  1.0073e-05,
 2.4121e-01,  6.6406e-02, -2.4536e-02,  1.8921e-02, -1.8311e-02,
 3.0664e-01,  1.8311e-02, -3.8574e-02,  1.8164e-01,  2.2736e-03,
 1.4551e-01,  6.7383e-02, -1.6846e-02,  1.8750e-01, -1.2109e-01,
 2.1729e-02,  1.2988e-01,  1.5527e-01, -7.8125e-02,  5.0537e-02,
-7.6660e-02, -1.1816e-01,  4.5166e-02, -3.7109e-02,  2.4805e-01,
-3.1055e-01,  1.9824e-01, -2.3730e-01, -2.1851e-02, -9.7656e-02,
-1.0803e-02,  1.6992e-01,  1.1377e-01, -5.1758e-02, -1.9434e-01,
 9.4727e-02,  1.2891e-01,  2.0508e-02, -2.7734e-01, -8.6914e-02,
 1.2634e-02, -4.5703e-01, -1.3379e-01, -8.1543e-02,  2.8906e-01,
 1.8945e-01,  1.6211e-01, -1.6479e-02,  3.7537e-03, -1.3867e-01,
-1.0498e-01, -1.2012e-01, -1.1353e-02,  5.1514e-02,  4.6875e-02,
-8.9355e-02,  2.5635e-02, -2.7734e-01,  1.0547e-01, -1.0303e-01,
 2.0703e-01, -1.6797e-01,  1.9922e-01,  1.6724e-02, -1.8066e-01,
 1.0645e-01, -3.3984e-01, -5.8105e-02, -3.3594e-01,  4.3945e-02,
 8.5938e-02,  4.4434e-02,  2.7466e-03,  1.7090e-01,  7.4219e-02,
-3.0640e-02, -7.8613e-02,  5.0354e-03, -1.1670e-01,  3.1836e-01,
 2.6367e-01, -9.4727e-02,  1.2158e-01,  1.5234e-01, -2.3340e-01,
 3.8477e-01, -1.1328e-01,  4.1504e-02, -2.2070e-01,  1.0559e-02,
-3.4180e-02,  7.5684e-02, -2.3633e-01, -6.9824e-02,  3.0396e-02,
-2.0020e-01,  1.7480e-01, -1.5723e-01, -3.9551e-02, -2.6953e-01,
-1.1182e-01,  1.0498e-01,  1.1475e-01, -2.4796e-04, -1.4258e-01,
-1.6309e-01, -2.6367e-01,  1.4453e-01,  1.6309e-01,  2.1973e-02,
-2.0605e-01, -3.1738e-02, -1.5625e-01, -1.0938e-01,  4.5703e-01,
 2.7148e-01,  6.1279e-02,  2.0142e-03, -1.2158e-01,  1.5820e-01]])
```

It's your turn! Go ahead with *Task 2.*

Task 2.1

- Plot the 3D PCA representation of the sentence embeddings, using a different color for each label (ham and spam)
  - Are the two classes neatly separated?

In [ ]:
```
pca_sent_emb =  PCA(n_components=3).fit_transform(np.stack(spam_df["sent_emb"]))
pca_df = pd.DataFrame(pca_sent_emb, columns=["pca_x", "pca_y", "pca_z"])
pca_df["label"] = label_encoder.inverse_transform(spam_df["label"])
px.scatter_3d(pca_df, x='pca_x', y='pca_y', z='pca_z', color="label", opacity=0.7)
```

pca_

Task 2.2

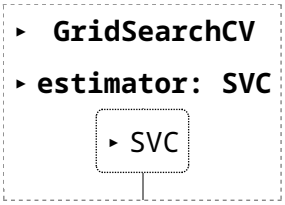
1. Implement a new classifier to solve this task.
   - You can use:
     - a `scikit-learn` estimator, for example: KNeighborsClassifier, SVC, DecisionTreeClassifier
     - **only** if you are confident with your pytorch skills, you can implement either:
       - a feed-forward network that processes the sentence embeddings
       - a RNN of your choice (LSTM, GRU, BiLSTM) that processes one word per time step
   - **Requirements**:
     - use the same train_test split done above (`sent_emb_train` / `sent_emb_test`)
2. Evaluate your model on the test set using classification_report

In [ ]:
```
## SOLUTION
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

tuned_parameters = [
    {"kernel": ["rbf"], "gamma": [1e-3, 1e-4], "C": [1, 10, 100, 1000]},
    {"kernel": ["linear"], "C": [1, 10, 100, 1000]},
]

# Find best hyperparameter combination according to F1 score and then refit the bes
grid_search = GridSearchCV(SVC(), tuned_parameters, cv=5, scoring="f1", refit="f1")
grid_search.fit(sent_emb_train, y_train) # GridSearch with 5-fold cross validation
```

Out[ ]:



In [ ]:

```
from sklearn.metrics import accuracy_score

print(f"SVC accuracy on test set {accuracy_score(y_test, grid_search.predict(sent_e
print(classification_report(y_test, grid_search.predict(sent_emb_test), target_name
```

```
SVC accuracy on test set 0.963996399639964
              precision    recall  f1-score   support

         ham       0.97      0.99      0.98       962
        spam       0.90      0.82      0.86       149

    accuracy                           0.96      1111
   macro avg       0.94      0.90      0.92      1111
weighted avg       0.96      0.96      0.96      1111
```