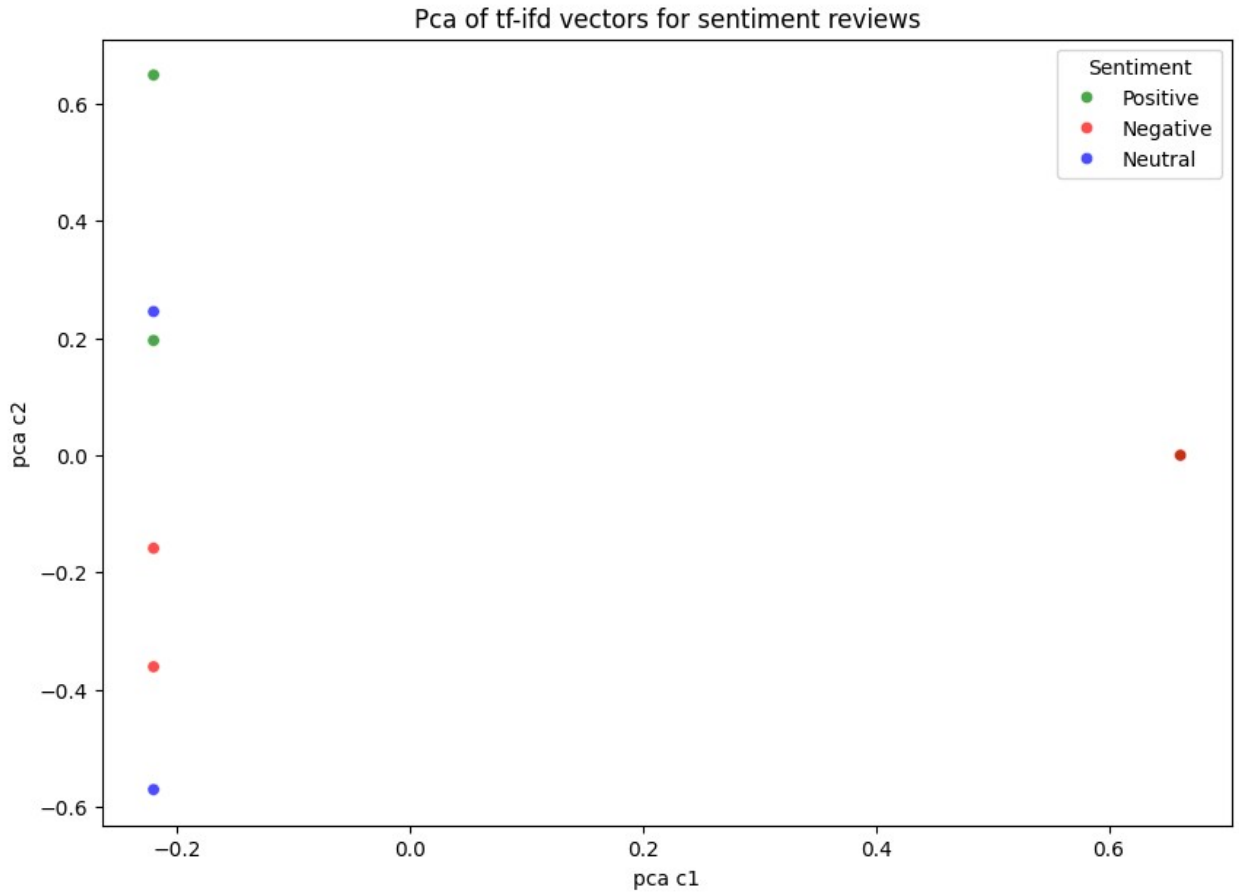
Q1

Q1. TF-IDF Embeddings + PCA Plot Use TF-IDF vectors on sentiment_reviews.csv. • Apply PCA to reduce vectors to 2 dimensions • Plot them with different colors for each sentiment label (Positive/Negative/Neutral)

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("NLP_LAB/1_TFIDF_PCA/review.csv")

tfidf = TfidfVectorizer(max_features=1000, stop_words='english')
x_tfidf = tfidf.fit_transform(df['review_text'])

pca = PCA(n_components=2)
x_pca = pca.fit_transform(x_tfidf.toarray())
feature_names = tfidf.get_feature_names_out()
print(feature_names)

pca_df = pd.DataFrame({
    'pca1': x_pca[:, 0],
    'pca2': x_pca[:, 1],
    'sentiment': df['sentiment_label']
})

plt.figure(figsize=(10, 7))
sns.scatterplot(data=pca_df, x='pca1', y='pca2', hue='sentiment',
palette={'Positive': 'Green', 'Negative': 'Red', 'Neutral': 'Blue'},
alpha=0.7)
plt.title("Pca of tf-ifd vectors for sentiment reviews")
plt.xlabel('pca c1')
plt.ylabel('pca c2')
plt.legend(title='Sentiment')
plt.show()
```

```
['ads' 'amazing' 'app' 'bad' 'best' 'better' 'buggy' 'clean' 'crashes'
 'customer' 'daily' 'excellent' 'experience' 'far' 'fine' 'frequently'
 'good' 'hate' 'just' 'okay' 'software' 'support' 'ui' 'unusable'
'use'
 'useful']
```



Q2

Q2. FastText Word Embeddings Train FastText on cleaned content from open_source.txt. • Compare similarities between the word "code" and "software", "collaboration", "freedom" • Comment on which pair is most semantically similar.

```
from gensim.models import FastText
from nltk.tokenize import word_tokenize, sent_tokenize


text = ''
with open('open_text.txt', 'r', encoding='utf-8') as file:
    text = file.read()

tokens = [word_tokenize(i) for i in text.split('\n') if
i.lower().strip()]
# tokens = [word_tokenize(text.lower())]

model = FastText(tokens, window=5, vector_size=100, min_count=2,
epochs=25)
```

```
sents = [["code", "software"],["code","collaboration"],
["code","freedom"]]

for i, j in sents:
    if i in model.wv and j in model.wv:
        print(f"{i} -> {j}: {model.wv.similarity(i ,j)}")
    else:
        print("OOV")
```

```
code -> software: 0.3622528314590454
code -> collaboration: 0.5744575262069702
code -> freedom: 0.5309708714485168
```

Q3

Q3. Contextual Similarity with spaCy Using spaCy's word vectors , compute similarity between "technology" and "innovation". • Discuss the meaning of the similarity score • Suggest another pair and compare

```
import spacy

nlp = spacy.load("en_core_web_lg")

w1 = nlp("technology")
w2 = nlp("innovation")

similarity = w1.similarity(w2)
print(f"Similarity between 'technology' and 'innovation':
{similarity}")

w3 = nlp("science")
w4 = nlp("research")

similarity2 = w3.similarity(w4)
print(f"Similarity between 'science' and 'research': {similarity2}")
```

```
Similarity between 'technology' and 'innovation': 0.7079066640586905
Similarity between 'science' and 'research': 0.6774647939129809
```

Q4

Q4. Word Graph with Cosine Similarity

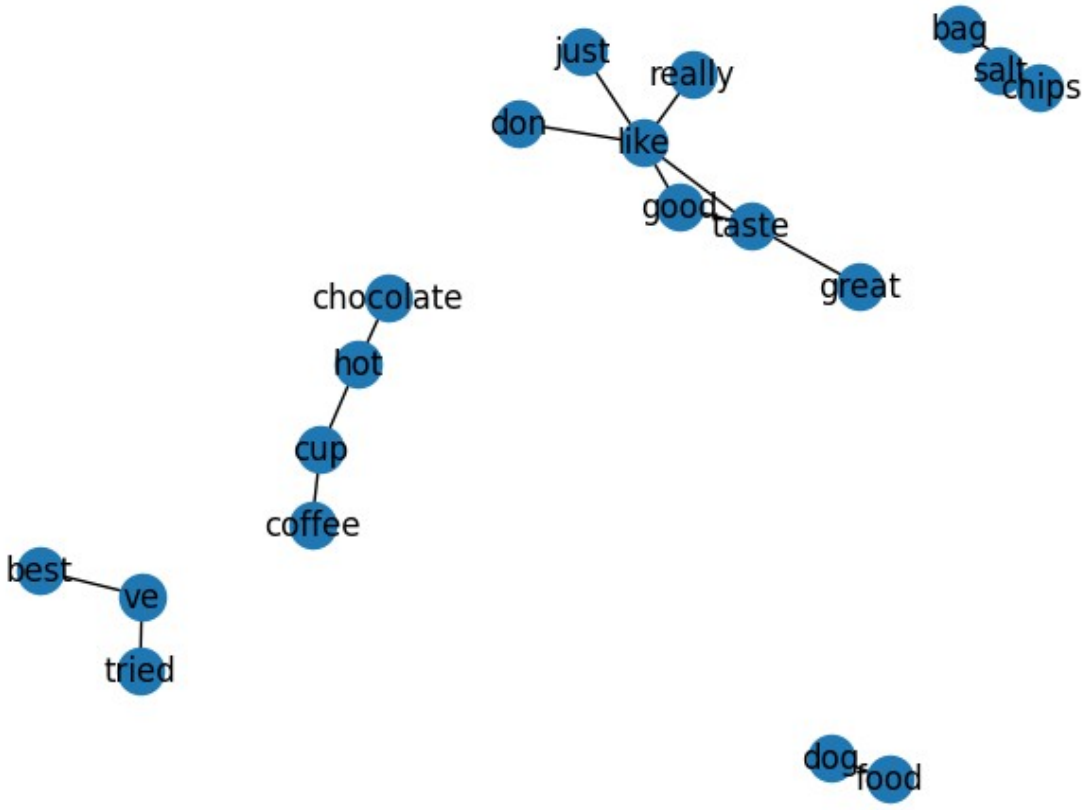
From doc1.txt:

• Create a cosine similarity matrix of TF-IDF vectors

• Use NetworkX to build a graph where words with similarity > 0.5 are connected

• Visualize the network with labels

```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

df = pd.read_csv("NLP_LAB/4_WordGraph_CosineSim/Reviews.csv")[:5000]
["Text"]

data = df.dropna().tolist()

tfidf = TfidfVectorizer(stop_words="english", max_features = 50)
tokens = tfidf.fit_transform(data).T

cos_sim = cosine_similarity(tokens)
words = tfidf.get_feature_names_out()

G = nx.Graph()

for i in range(len(words)):
    for j in range(i+1, len(words)):
        if cos_sim[i,j]>=0.2:
            G.add_edge(words[i], words[j], weight = cos_sim[i,j])

pos = nx.spring_layout(G, seed = 42)
nx.draw(G, pos, with_labels = True)
plt.show()
```



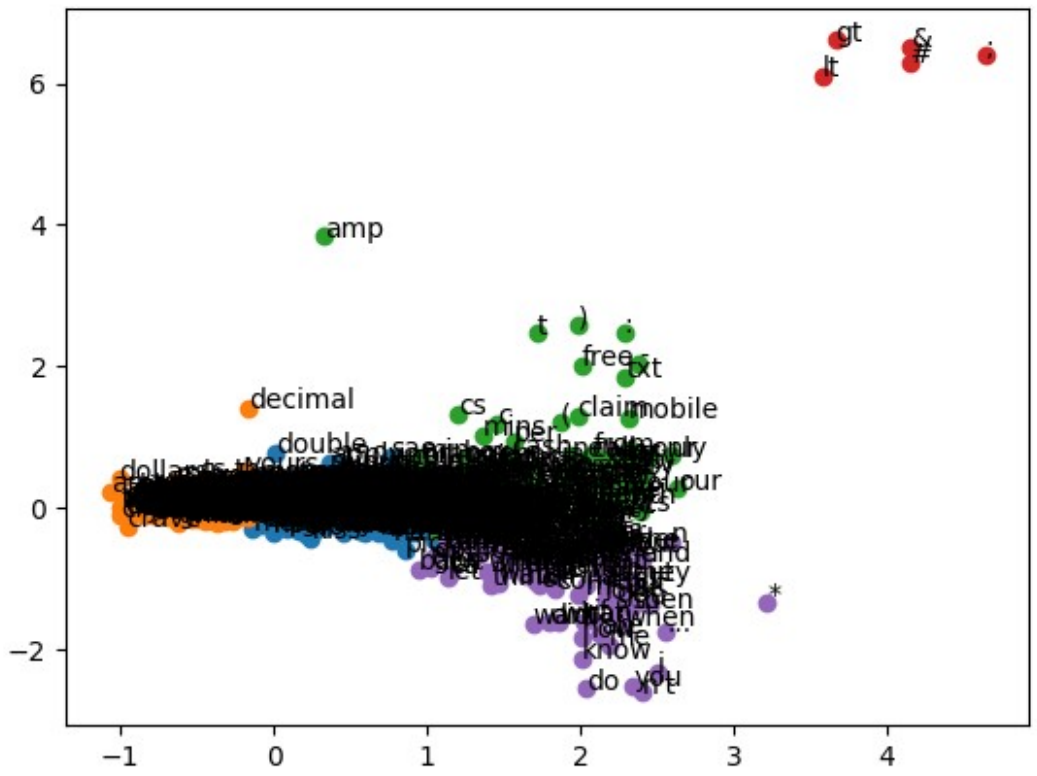
Q5

Q5. CBOW Word Clustering Train a CBOW Word2Vec model on email_dataset.csv. • Cluster word vectors using KMeans • Visualize clusters in 2D using PCA • Annotate each cluster with top representative words

```
from gensim.models import Word2Vec
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from nltk.tokenize import word_tokenize

df = pd.read_csv("NLP_LAB/5_CBOW_Word_Cluster/email.csv")

data = df["Message"].dropna().str.lower().tolist()

tokens = [word_tokenize(i) for i in data]

model = Word2Vec(tokens, vector_size=100, window=5, sg=0)
```

```
words = list(model.wv.index_to_key)
keys = model.wv[words]

k=5
kmeans = KMeans(n_clusters=k)
labels = kmeans.fit_predict(keys)

pca = PCA(n_components=2)
reduced_vecs = pca.fit_transform(keys)

for i in range(k):
    idx = [j for j,label in enumerate(labels) if i==label]
    plt.scatter(reduced_vecs[idx,0], reduced_vecs[idx,1],
label=f"Cluster{i}")

for i in range(len(words)):
    plt.text(reduced_vecs[i][0], reduced_vecs[i,1], words[i])
plt.show()
```

```
C:\Users\kushal bang\AppData\Local\Programs\Python\Python311\Lib\site-
packages\IPython\core\pylabtools.py:170: UserWarning: Glyph 146 (\x92)
missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```



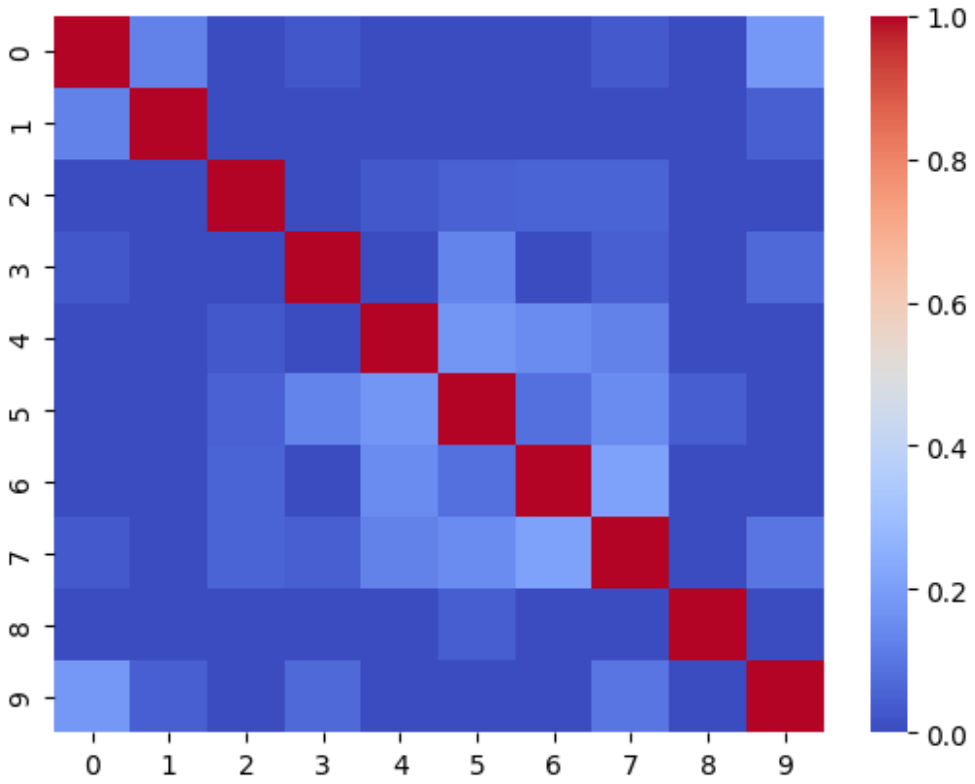
Q6

Q6. Review Similarity Heatmap Create a cosine similarity matrix for all reviews in sentiment_reviews.csv. • Use Seaborn to visualize this as a heatmap • Explain any visible clusters or patterns

```
import pandas as pd
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt

data = pd.read_csv("NLP_LAB/6_ReviewSIm_HeatMap/Reviews.csv")[:10]
["Text"].dropna().tolist()
print(data)

tfidf = TfidfVectorizer(stop_words="english")
tfidf_mat = tfidf.fit_transform(data)

cos_sim  = cosine_similarity(tfidf_mat)

sns.heatmap(cos_sim, cmap = "coolwarm")
plt.show()
```

```
['I have bought several of the Vitality canned dog food products and
have found them all to be of good quality. The product looks more like
a stew than a processed meat and it smells better. My Labrador is
finicky and she appreciates this product better than  most.', 'Product
arrived labeled as Jumbo Salted Peanuts...the peanuts were actually
small sized unsalted. Not sure if this was an error or if the vendor
intended to represent the product as "Jumbo".', 'This is a confection
that has been around a few centuries.  It is a light, pillowy citrus
gelatin with nuts - in this case Filberts. And it is cut into tiny
squares and then liberally coated with powdered sugar.  And it is a
tiny mouthful of heaven.  Not too chewy, and very flavorful.  I highly
recommend this yummy treat.  If you are familiar with the story of
C.S. Lewis\' "The Lion, The Witch, and The Wardrobe" - this is the
treat that seduces Edmund into selling out his Brother and Sisters to
the Witch.', 'If you are looking for the secret ingredient in
Robitussin I believe I have found it.  I got this in addition to the
Root Beer Extract I ordered (which was good) and made some cherry
soda.  The flavor is very medicinal.', 'Great taffy at a great price.
There was a wide assortment of yummy taffy.  Delivery was very quick.
If your a taffy lover, this is a deal.', 'I got a wild hair for taffy
and ordered this five pound bag. The taffy was all very enjoyable with
many flavors: watermelon, root beer, melon, peppermint, grape, etc. My
only complaint is there was a bit too much red/black licorice-flavored
pieces (just not my particular favorites). Between me, my kids, and my
husband, this lasted only two weeks! I would recommend this brand of
```

```
taffy -- it was a delightful treat.', "This saltwater taffy had great
flavors and was very soft and chewy.  Each candy was individually
wrapped well.  None of the candies were stuck together, which did
happen in the expensive version, Fralinger's.  Would highly recommend
this candy!  I served it at a beach-themed party and everyone loved
it!", 'This taffy is so good.  It is very soft and chewy.  The flavors
are amazing.  I would definitely recommend you buying it.  Very
satisfying!!', "Right now I'm mostly just sprouting this so my cats
can eat the grass. They love it. I rotate it around with Wheatgrass
and Rye too", 'This is a very healthy dog food. Good for their
digestion. Also good for small puppies. My dog eats her required
amount at every feeding.']
```



Q7

Q7. spaCy Context-Based Similarity From doc2.txt, identify words most contextually similar to "intelligence" using spaCy's similarity API. • List the top 5 most similar words • Reflect on the results and semantic meanings

```
import spacy

nlp = spacy.load("en_core_web_lg")
```

```
data = """Artificial intelligence is transforming the way we interact
with technology.
Smart assistants like Siri and Alexa use natural language processing
to understand speech.
Machine learning models improve as they process more data, enabling
better predictions.
Deep learning and neural networks are at the core of modern AI
systems.
The intelligence exhibited by machines today is narrow but evolving.
Cognitive computing aims to simulate human intelligence through
algorithms and learning systems.
"""

tokens  = [w for w in data.split(" ") if w.isalpha()]

vec = list(set([w for w in tokens if w not in
nlp.Defaults.stop_words]))

target = nlp("intelligence")
sims = []

for i in tokens:
    i_vec = nlp(i)
    sim = target.similarity(i_vec)
    sims.append((i,sim))

top = sorted(sims, key = lambda x:x[1], reverse = True)[:5]

for i,j, in top:
    print(f"{i}->{j:.4f}")
```

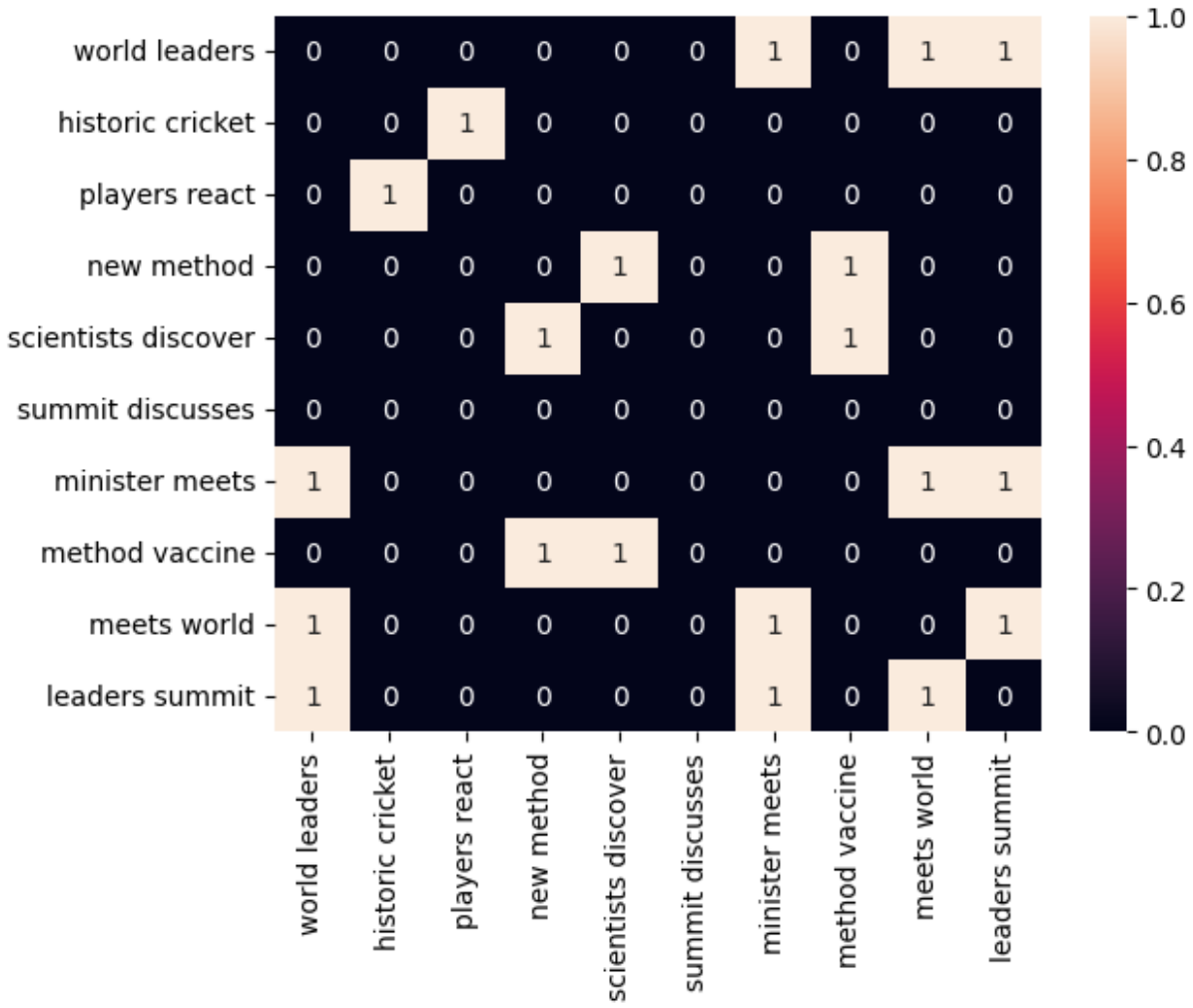
```
intelligence->1.0000
intelligence->1.0000
intelligence->1.0000
human->0.5103
computing->0.4209
```

Q8

Q8. Co-occurrence Heatmap Create a word co-occurrence matrix from news_headlines.csv. • Use NLTK or sklearn's CountVectorizer with bigrams • Plot the co-occurrence matrix as a heatmap • Explain how this aids topic modeling or news classification

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv("NLP_LAB/8_CoOccurence_Heatmap/news.csv")
texts = df["headline"].dropna().tolist()

model = CountVectorizer(ngram_range=(2,2), stop_words="english")
x = model.fit_transform(texts)

co_oc = (x.T*x)
co_oc.setdiag(0)

co_df = pd.DataFrame(co_oc.toarray(), index =
model.get_feature_names_out(), columns =
model.get_feature_names_out())


top_n = 10
top_bigrams = co_df.sum(axis =
1).sort_values(ascending=False).head(top_n).index
heatmap_df = co_df.loc[top_bigrams, top_bigrams]

sns.heatmap(heatmap_df, annot = True )
plt.show()
```

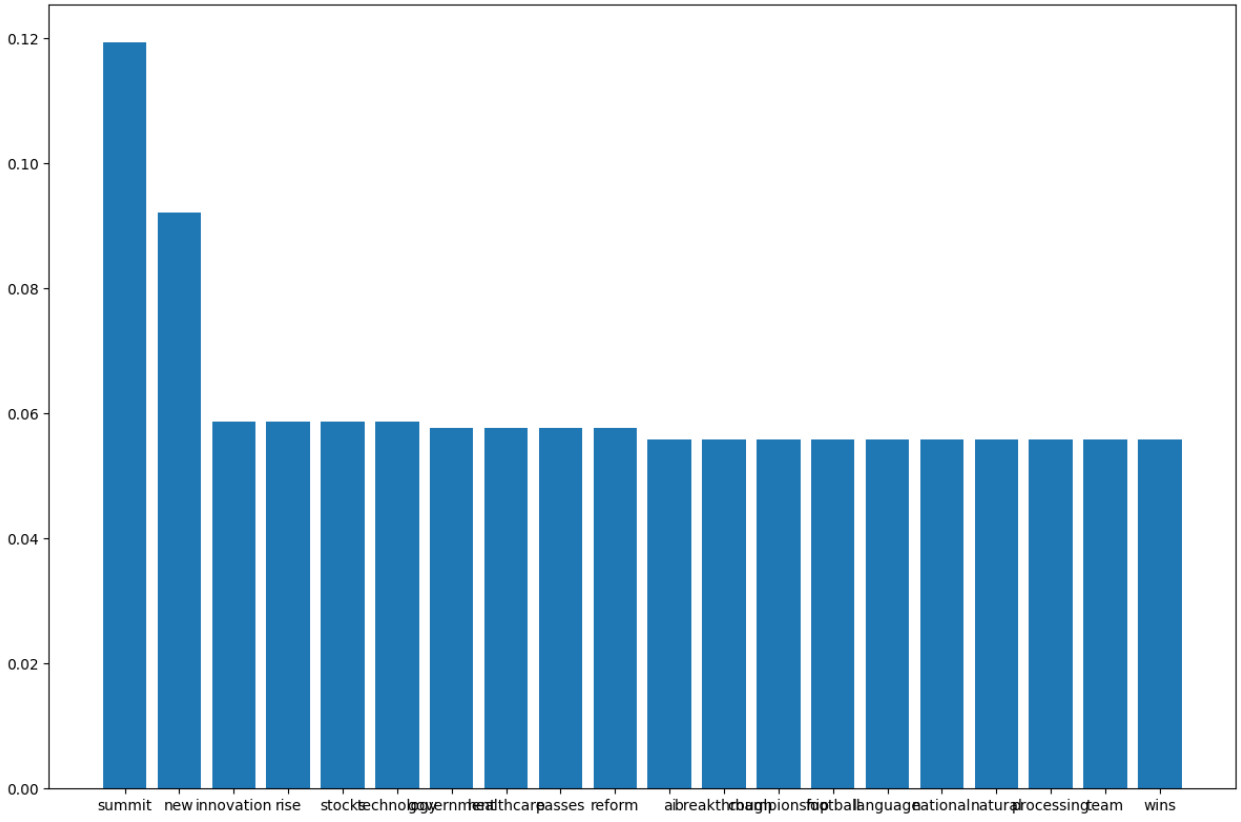


Q9

Q9. TF-IDF Feature Visualization Using news_headlines.csv, compute TF-IDF scores. • Identify the top 20 words with highest average TF-IDF weight • Plot a horizontal bar chart of these features with labels

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt

df = pd.read_csv("NLP_LAB/9_Tfidf_Feature_Vizualisation/news.csv")
text = df["headline"].dropna().tolist()

tfidf = TfidfVectorizer(stop_words="english")
vec = tfidf.fit_transform(text)

features = tfidf.get_feature_names_out()
```

```
scores = vec.mean(axis=0).A1

word_scores = list(zip(features, scores))
top = sorted(word_scores, key = lambda x: x[1], reverse = True)[:20]

words, scores = zip(*top)
plt.figure(figsize=(12, 8))
plt.bar(words, scores)
plt.tight_layout()
plt.show()
```



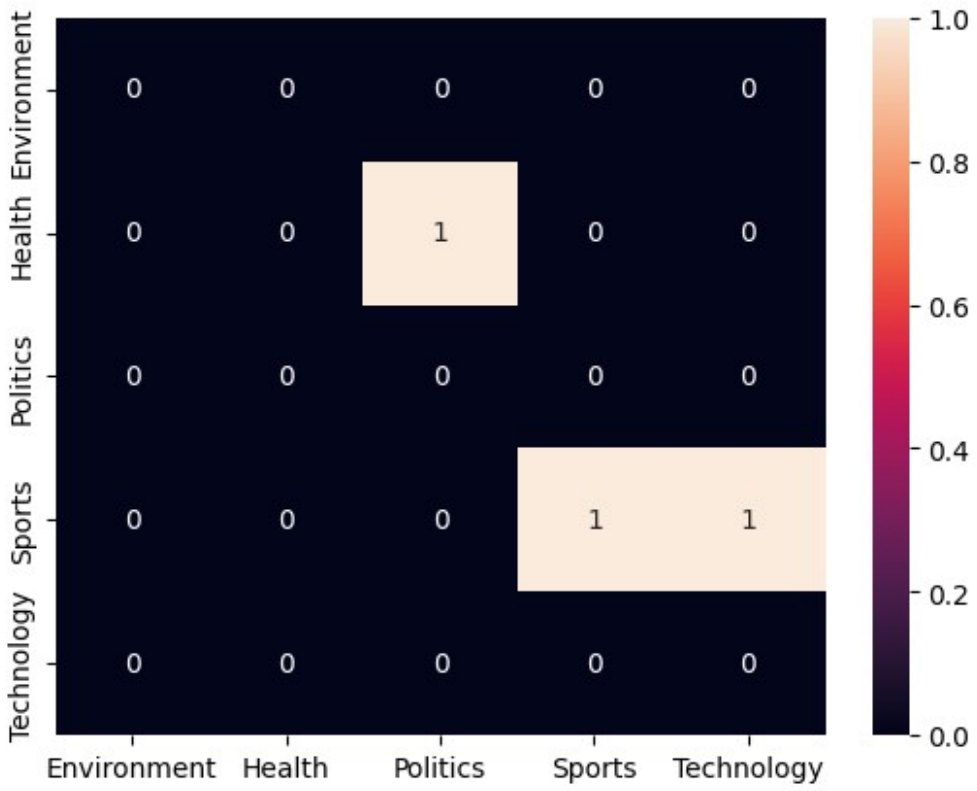
Q10

Q10. News Headline Classification Model Dataset: news_headlines.csv Objective: Classify news headlines into categories such as Politics, Sports, and Technology.

1. Load the dataset and preprocess the headlines (lowercasing, punctuation removal, etc.).
2. Vectorize the text using TF-IDF.
3. Train a classification model (e.g., Logistic Regression or Multinomial Naive Bayes).
4. Split data into training and test sets, and evaluate the model using: o Accuracy o Confusion matrix o Classification report

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("NLP_LAB/10_News_Classification_model/news.csv")

x = df["headline"].astype("str")
y = df["category"]

tfidf = TfidfVectorizer(stop_words="english")
x_vec = tfidf.fit_transform(x)

x_train,x_test,y_train,y_test = train_test_split(x_vec, y,
test_size=0.3, random_state=42)

model = LogisticRegression(max_iter=1000)
model.fit(x_train,y_train)

y_pred = model.predict(x_test)


accuracy = accuracy_score(y_pred, y_test)
print("Accuracy is: ",accuracy)

print("Classification report: \n",classification_report(y_pred,
y_test, zero_division=0))

cm = confusion_matrix(y_pred, y_test, labels=model.classes_)
sns.heatmap(cm,annot=True, xticklabels= model.classes_,
yticklabels=model.classes_)
plt.show()
```

```
Accuracy is:  0.3333333333333333
Classification report:
               precision    recall  f1-score   support

      Health       0.00      0.00      0.00         1
    Politics       0.00      0.00      0.00         0
      Sports       1.00      0.50      0.67         2
  Technology       0.00      0.00      0.00         0

    accuracy                           0.33         3
   macro avg       0.25      0.12      0.17         3
weighted avg       0.67      0.33      0.44         3
```



Q11

Q11. Sentiment Analysis Classifier Dataset: sentiment_reviews.csv Objective: Build a classifier to detect whether a review is Positive, Negative, or Neutral.

1. Clean the review text (remove stop words, lowercase, tokenize).
2. Convert text into TF-IDF features.
3. Build and train a classifier using any supervised ML model.
4. Predict sentiment on unseen data such as: "This app is just okay, not too good, not too bad."
5. Display and explain model metrics (accuracy, precision, recall).

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("NLP_LAB/11_SentimentAnalysis_Classifier/review.csv")

x = df["review_text"].astype("str")
```

```
y = df["sentiment_label"]

tfidf = TfidfVectorizer(stop_words="english")
x_vec = tfidf.fit_transform(x)

x_train,x_test, y_train, y_test = train_test_split(x_vec, y,
test_size=0.3, random_state = 42)

model = LogisticRegression(max_iter = 1000)
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

print("Accuracy: ", accuracy_score(y_pred, y_test))
print("Classification report:\n ", classification_report(y_pred,
y_test))

new = ["This was an absolute shitty product. Don't buy it."]
new_vec = tfidf.transform(new)
new_pred = model.predict(new_vec)[0]

print(new,new_pred)
```

```
Accuracy:  0.0
Classification report:
                precision    recall  f1-score   support

    Negative       0.00      0.00      0.00       1.0
     Neutral       0.00      0.00      0.00       0.0
    Positive       0.00      0.00      0.00       2.0

    accuracy                           0.00       3.0
   macro avg       0.00      0.00      0.00       3.0
weighted avg       0.00      0.00      0.00       3.0

["This was an absolute shitty product. Don't buy it."] Positive

C:\Users\kushal bang\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\metrics\_classification.py:1531:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
C:\Users\kushal bang\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\metrics\_classification.py:1531:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

```
C:\Users\kushal bang\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\metrics\_classification.py:1531:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

Q12

Q12. Spam-Ham Classification for Emails Dataset: email_dataset.csv Objective: Classify emails as Spam or Ham using text classification.

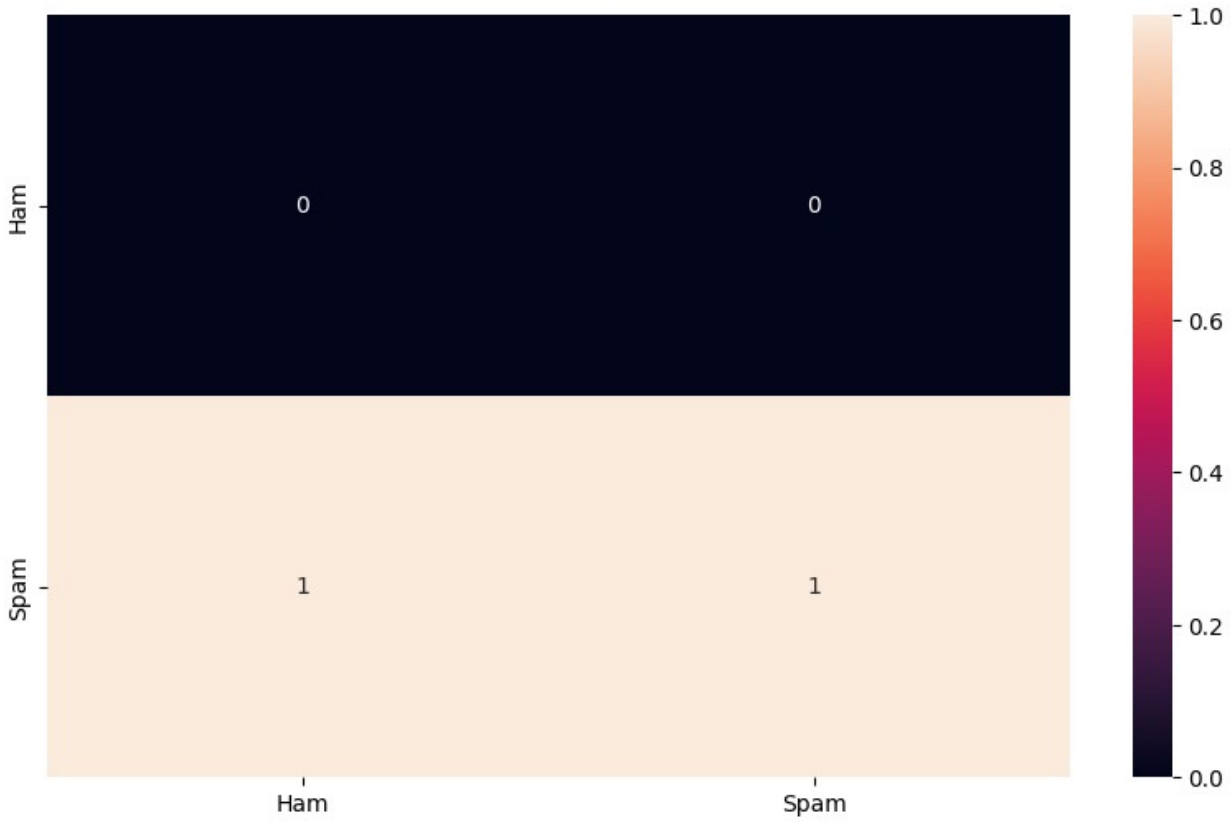
1. Clean the email messages (punctuation, numbers, stopwords).
2. Use CountVectorizer or TF-IDF Vectorizer to extract features.
3. Train a classification model (e.g., MultinomialNB or SVM).
4. Evaluate your model: o Confusion matrix o Accuracy
5. Identify and display top features/words contributing to spam prediction.

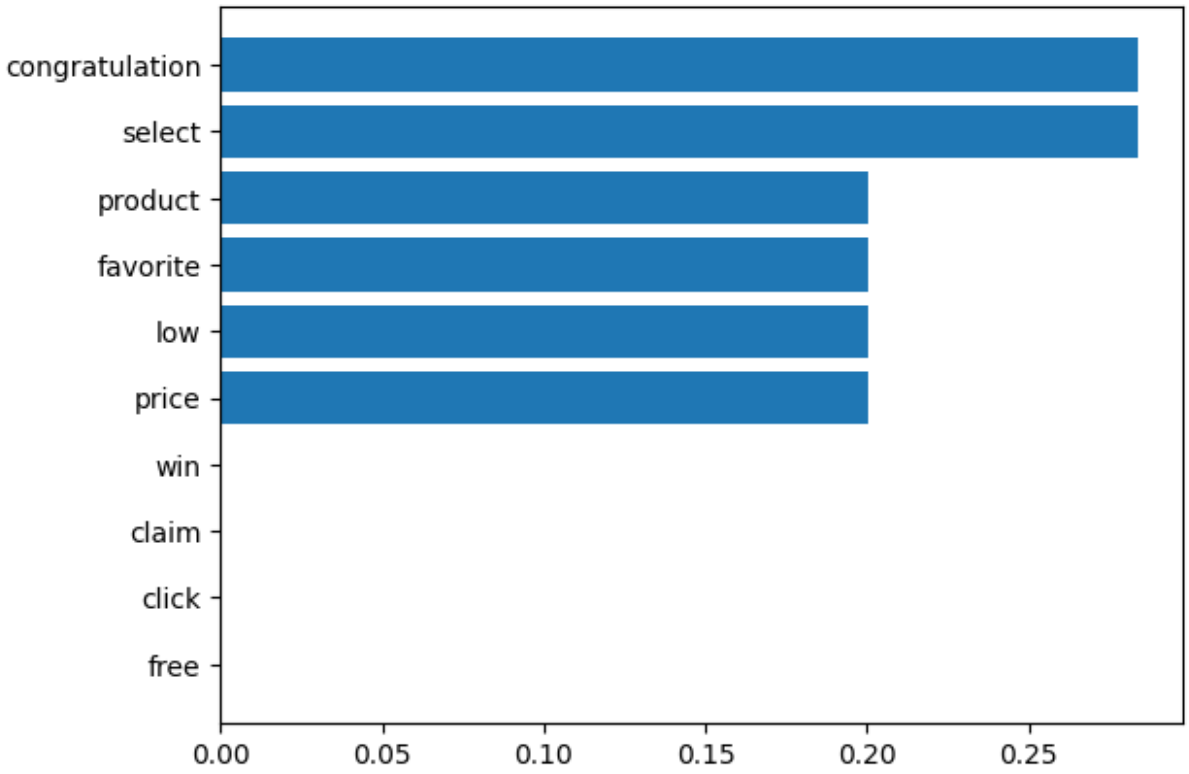
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
import spacy
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
import seaborn as sns
import matplotlib.pyplot as plt

nlp = spacy.load("en_core_web_sm")

def preprocess(x):
    doc = nlp(x.lower())
    tokens = [
        token.lemma_ for token in doc if token.is_alpha and not
token.is_stop
    ]
    return " ".join(tokens)

df = pd.read_csv("NLP_LAB/12_Spam_Ham_Email_Classify/email.csv")
df["message"] = df["message"].apply(preprocess)

x = df["message"]
y = df["label"]

tfidf = TfidfVectorizer(stop_words="english")
x_vec = tfidf.fit_transform(x)
```

```
x_train, x_test, y_train, y_test = train_test_split(x_vec, y,
test_size=0.3, random_state=42)

model = LogisticRegression()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

print("Accuracy: ",accuracy_score(y_pred, y_test))
print("Classification report: ", classification_report(y_pred,
y_test))

cm = confusion_matrix(y_pred, y_test)
plt.figure(figsize=(10,6))

sns.heatmap(cm, xticklabels=model.classes_,
yticklabels=model.classes_, annot = True)

feature_names = tfidf.get_feature_names_out()
coefs = model.coef_[0]

coef_df = pd.DataFrame({
    "word": feature_names,
    "coef": coefs
})

top_spam_words = coef_df.sort_values(by="coef",
ascending=False).head(10)
plt.figure()
plt.barh(top_spam_words["word"][::-1], top_spam_words["coef"][::-1])
plt.show()
```

```
Accuracy:  0.5
Classification report:                precision    recall  f1-score
support

         Ham       0.00      0.00      0.00         0
        Spam       1.00      0.50      0.67         2

    accuracy                           0.50         2
   macro avg       0.50      0.25      0.33         2
weighted avg       1.00      0.50      0.67         2


C:\Users\kushal bang\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\metrics\_classification.py:1531:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

```
C:\Users\kushal bang\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\kushal bang\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```





Q13

Q13. Spam-Ham Detection with Feature Importance Dataset: email_dataset.csv Objective: Detect spam emails and identify the most predictive words.

1. Train a logistic regression classifier using TF-IDF features.
2. Use model coefficients to identify the top 10 indicative words of spam.
3. Visualize them using a horizontal bar chart with importance scores.
4. Compare results with Naive Bayes classifier.

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("NLP_LAB/13_SpamHam_FeatureImportance/emails.csv")

x = df["message"].astype("str")
y = df["label"]
```

```
tfidf = TfidfVectorizer()
x_vec = tfidf.fit_transform(x)

x_train, x_test, y_train, y_test = train_test_split(x_vec, y,
test_size=0.3, random_state=42)

model = MultinomialNB()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

print("Accuracy: ", accuracy_score(y_pred, y_test))
print("Classification report: ", classification_report(y_pred,
y_test))

feature_names = tfidf.get_feature_names_out()
log_scores = model.feature_log_prob_[1]
idxs = log_scores.argsort()[-10:][::-1]
top_words = [feature_names[i] for i in idxs]

print("Top Words were: ",top_words )
```

```
Accuracy:  0.5
Classification report:                precision    recall  f1-score
support

         Ham       0.00      0.00      0.00         0
        Spam       1.00      0.50      0.67         2

    accuracy                           0.50         2
   macro avg       0.50      0.25      0.33         2
weighted avg       1.00      0.50      0.67         2

Top Words were:  ['congratulations', 've', 'selected', 'been',
'products', 'lowest', 'on', 'prices', 'favorite', 'your']

C:\Users\kushal bang\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\metrics\_classification.py:1531:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
C:\Users\kushal bang\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\metrics\_classification.py:1531:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
C:\Users\kushal bang\AppData\Local\Programs\Python\Python311\Lib\site-
```

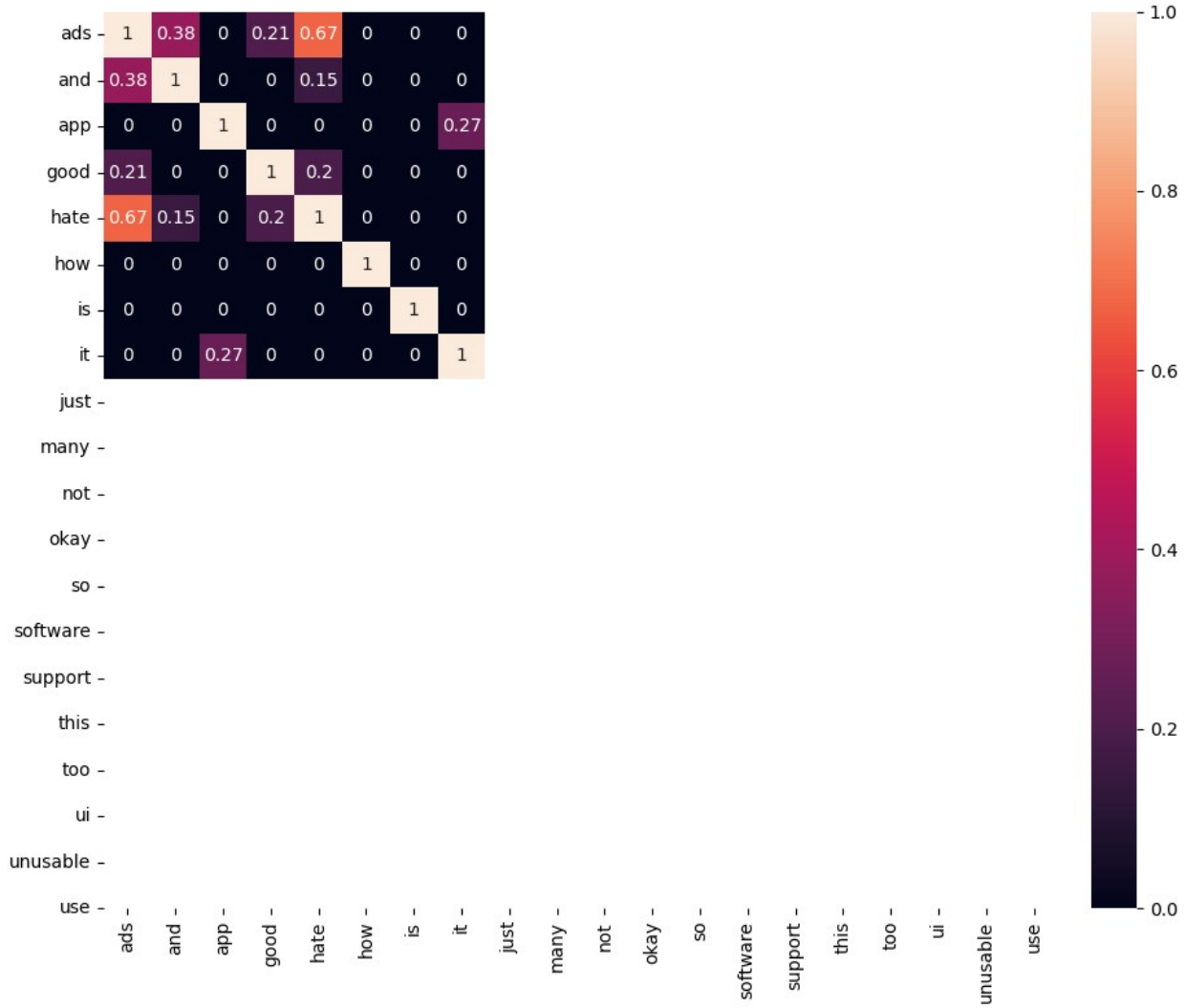
```
packages\sklearn\metrics\_classification.py:1531:
UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in
labels with no true samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

Q14

Q14. Multi-Class Sentiment Heatmap Dataset: sentiment_reviews.csv Objective: Visualize inter-sentiment similarities.

1. Convert reviews to TF-IDF.
2. Compute cosine similarity between all pairs.
3. Plot a similarity matrix heatmap with sentiment labels.

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt
import seaborn as sns


df = pd.read_csv("NLP_LAB/14_MultiClass_Sentiment/review.csv")

text = df["review_text"].dropna()

tfidf = TfidfVectorizer(max_features = 20)
vec = tfidf.fit_transform(text)

feature_names = tfidf.get_feature_names_out()
cos_sim = cosine_similarity(vec)

plt.figure(figsize=(10,8))
sns.heatmap(cos_sim, xticklabels=feature_names,
yticklabels=feature_names, annot = True)
plt.tight_layout()
plt.show()
```



Q15

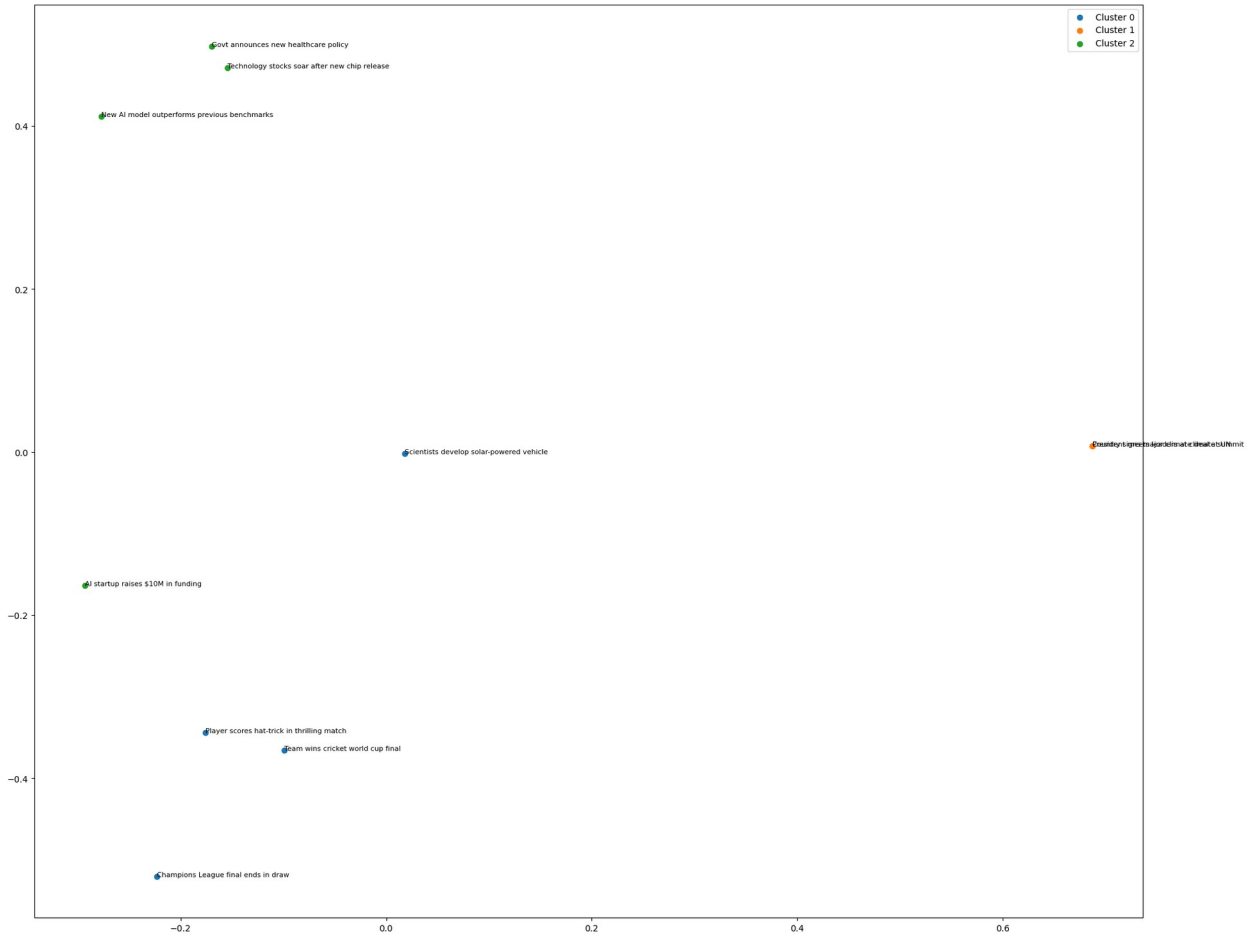
Q15. News Headline Clustering Dataset: news_headlines.csv Objective: Group headlines into clusters based on textual similarity.

1. Use TF-IDF on headline text.
2. Apply KMeans clustering (k=3).
3. Visualize the cluster distribution using PCA or t-SNE.
4. Manually interpret whether clusters align with original labels.

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv("NLP_LAB/15_NewsHeadline_Cluster/headline.csv")

x = df["headline"].dropna().tolist()

tfidf = TfidfVectorizer()
x_vec = tfidf.fit_transform(x)

kmeans = KMeans(n_clusters = 3)
cluster = kmeans.fit_predict(x_vec)
print(cluster)

pca = PCA(n_components = 2)
x_pca = pca.fit_transform(x_vec.toarray())

plt.figure(figsize=(20,15))
for i in range(3):
    plt.scatter(x_pca[cluster == i,0], x_pca[cluster == i,1], label =
f"Cluster {i}")

for i,txt in enumerate(df["headline"]):
    plt.annotate(txt, (x_pca[i,0], x_pca[i,1]), fontsize = 8)

plt.legend()
plt.tight_layout()
plt.show()
```

```
[2 2 0 1 2 0 2 1 0 0]
```



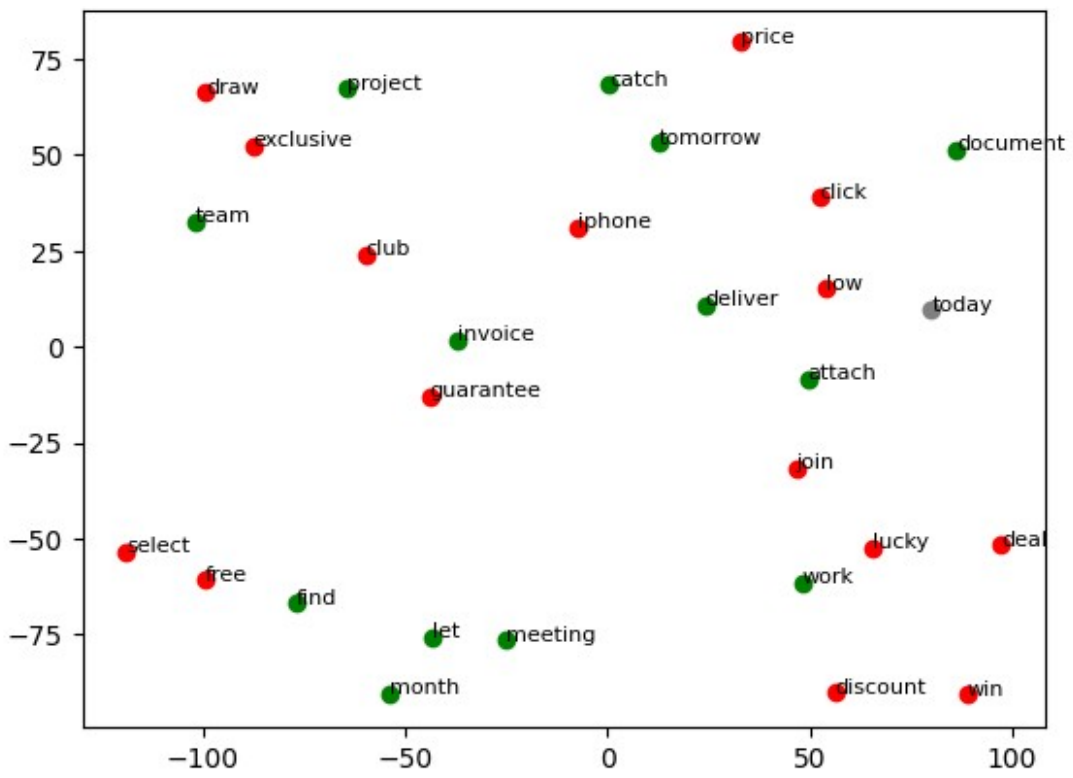
Q16

Q16. Email Word2Vec Visual Cluster Dataset: email_dataset.csv Objective: Visualize email vocabulary with Word2Vec.

1. Train a Word2Vec model on cleaned email text.
2. Reduce dimensions using t-SNE.
3. Plot word vectors in 2D.
4. Highlight spam-related and ham-related clusters with color coding.

```
import pandas as pd
from gensim.models import Word2Vec
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import spacy
from collections import Counter

nlp = spacy.load("en_core_web_sm")

def preprocess(x):
```

```
    doc = nlp(x.lower())
    token = [i.lemma_ for i in doc if i.is_alpha and not i.is_stop and
len(i.lemma_)>2]
    return token

df = pd.read_csv("NLP_LAB/16_Email_Word2Vec_Classify/email.csv")
df["tokens"] = df["message"].dropna().apply(preprocess)

model = Word2Vec(sentences = df["tokens"], window = 5, vector_size=50,
min_count = 1,sg = 0)

words = list(model.wv.index_to_key)
word_vec = model.wv[words]

tsne = TSNE(n_components = 2, random_state = 42, perplexity= 5)
x_tsne = tsne.fit_transform(word_vec)

spam_words = df[df["label"] == "Spam"]["tokens"].explode()
ham_words = df[df["label"] == "Ham"]["tokens"].explode()

spam_count = Counter(spam_words)
ham_count = Counter(ham_words)
colors = []

for word in words:
    spam_freq = spam_count[word]
    ham_freq = ham_count[word]
    if spam_freq>ham_freq:
        colors.append("Red")
    elif ham_freq>spam_freq:
        colors.append("Green")
    else:
        colors.append("Grey")

for i,word in enumerate(words):
    plt.scatter(x_tsne[i,0], x_tsne[i,1], color = colors[i])
    plt.annotate(word, (x_tsne[i,0], x_tsne[i,1]), fontsize = 8)
plt.show()
```


price
draw
project
catch
exclusive
tomorrow
document
click
team
iphone
club
low
deliver
today
invoice
attach
guarantee
join
select
lucky
deal
free
work
find
let
meeting
month
discount
win
75
50
25
0
−25
−50
−75
−100
−50
0
50
100