Team 12 - Assignment 04: Cluster Analysis, ANN and Text Mining Project

Using a Python version 3.11.7 globally or in a virtual environment

1. Update pip: `python.exe -m pip install --upgrade pip`
2. run the following command in your terminal: `pip install -r requirements.txt`

This will install the required dependencies to run our Python code

- ipykernel: the Python execution backend for Jupyter
- matplotlib: a comprehensive library for creating static, animated, and interactive visualizations in Python
- tensorflow: open source software library for high performance numerical computation.
  - As of 2023/12/08 `tensorflow` does not work with Python version 3.12.0
- numpy: for scientific computing with Python
- pandas: provides fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive
- scikit-learn: for machine learning built on top of SciPy and is distributed under the 3-Clause BSD license

Team 12 Members:

- Alicia Luna
- Derek Dilger
- Gary Young
- Ian Schultz
- Jesus Cervantes
- Matthew Mendoza
- Thomas Jaramillo-Ochoa
- Yun-jeong Lee

1. Clustering

K-Means

Reading dataset to achieve K-means from cluster analysis

In [ ]:
```
import pandas as pd

# Load the dataset
imdb_data = pd.read_csv('./data/imdb_dataset.csv')
```

```
# Displaying the first few rows of the dataset for an overview
imdb_data.head()
```

Out[ ]:

	Unnamed: 0	title	title_type	genre	runtime	mpaa_rating	studio	thtr_r
0	1	Filly Brown	Feature Film	Drama	80.0	R	Indomina Media Inc.	
1	2	The Dish	Feature Film	Drama	101.0	PG-13	Warner Bros. Pictures	
2	3	Waiting for Guffman	Feature Film	Comedy	84.0	R	Sony Pictures Classics	
3	4	The Age of Innocence	Feature Film	Drama	139.0	PG	Columbia Pictures	
4	5	Malevolence	Feature Film	Horror	90.0	R	Anchor Bay Entertainment	

5 rows × 33 columns

Implementing the Elbow method by plotting SSE against possible cluster counts

In [ ]:

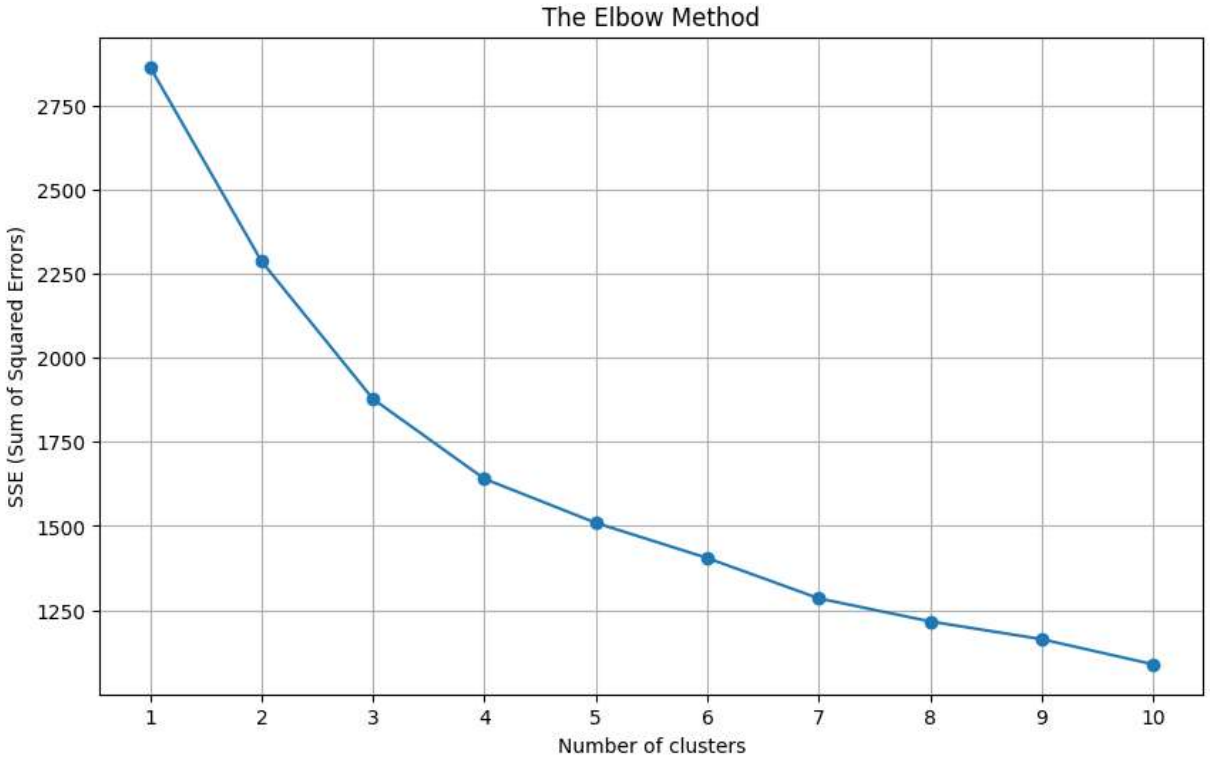
```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
import numpy as np

# Selecting relevant features for clustering
# Assuming 'genre', 'runtime', 'mpaa_rating', 'imdb_rating', 'imdb_num_votes' as re
features = ['genre', 'runtime', 'mpaa_rating', 'imdb_rating', 'imdb_num_votes']
imdb_data_selected = imdb_data[features]

# Preprocessing steps
# Handling missing values and encoding categorical data
# Numerical features will be standardized
numerical_features = imdb_data_selected.select_dtypes(
    include=['int64', 'float64']).columns
categorical_features = imdb_data_selected.select_dtypes(
    include=['object']).columns

numerical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('std_scaler', StandardScaler())
])

categorical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

```
preprocessor = ColumnTransformer([
    ('num', numerical_pipeline, numerical_features),
    ('cat', categorical_pipeline, categorical_features)
])

# Preprocessing the data
imdb_data_preprocessed = preprocessor.fit_transform(imdb_data_selected)

# Checking the shape of the processed data
imdb_data_preprocessed.shape
```

Out[ ]:
```
(651, 20)
```

In [ ]:
```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Determining the optimal number of clusters using the Elbow method
sse = []
range_clusters = range(1, 11)  # Trying with 1 to 10 clusters

for k in range_clusters:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(imdb_data_preprocessed)
    sse.append(kmeans.inertia_)

# Plotting the SSE vs number of clusters
plt.figure(figsize=(10, 6))
plt.plot(range_clusters, sse, marker='o')
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('SSE (Sum of Squared Errors)')
plt.xticks(range_clusters)
plt.grid(True)
plt.show()
```

```
d:\repo\csc177-group-project\.venv\Lib\site-packages\sklearn\cluster\_kmeans.py:141
6: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



Applying K-means clustering

In [ ]:
```
# Applying K-means clustering with the chosen number of clusters
optimal_clusters = 4
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
kmeans.fit(imdb_data_preprocessed)

# Assigning the cluster labels to each movie
cluster_labels = kmeans.labels_

# Adding cluster labels to the original dataset for analysis
imdb_data['cluster'] = cluster_labels

# Displaying the first few rows with cluster labels
imdb_data[
    ['title',
     'genre',
     'runtime',
     'mpaa_rating',
     'imdb_rating',
     'imdb_num_votes',
     'cluster']
].head()
```

Out[ ]:

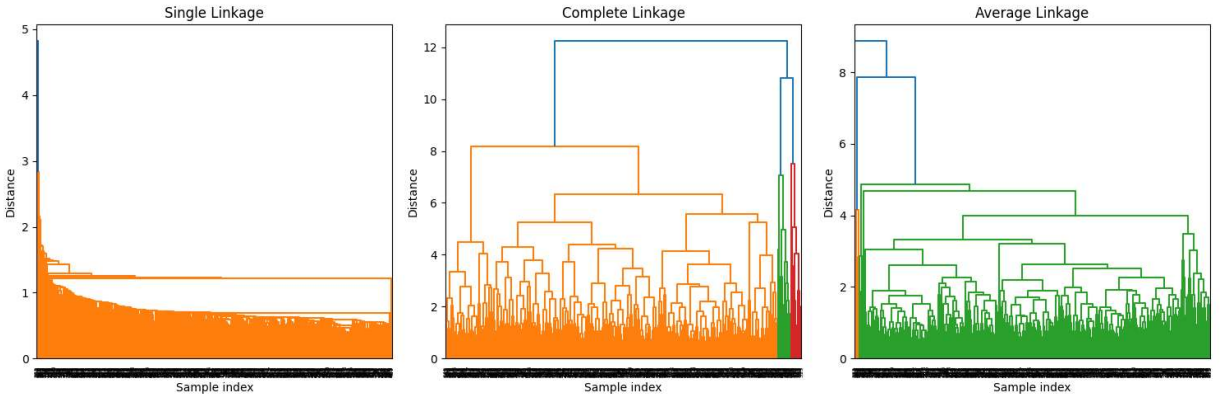
	title	genre	runtime	mpaa_rating	imdb_rating	imdb_num_votes	cluster
0	Filly Brown	Drama	80.0	R	5.5	899	0
1	The Dish	Drama	101.0	PG-13	7.3	12285	1
2	Waiting for Guffman	Comedy	84.0	R	7.6	22381	1
3	The Age of Innocence	Drama	139.0	PG	7.2	35096	2
4	Malevolence	Horror	90.0	R	5.1	2386	0

Hierarchical

In [ ]:

```
import pandas as pd
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import numpy as np

# Load the dataset
data = pd.read_csv('./data/imdb_dataset.csv')

# Selecting relevant numerical features for clustering
features = data[["runtime", "thtr_rel_year", "imdb_rating",
                 "imdb_num_votes", "critics_score", "audience_score"]]

# Check for missing or infinite values and handle them
# Replace infinite values with NaN
features = features.replace([np.inf, -np.inf], np.nan)
features = features.dropna()  # Drop rows with NaN values

# Normalizing the data
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Ensure the scaled data does not contain non-finite values
if not np.all(np.isfinite(features_scaled)):
    raise ValueError("Non-finite values found in scaled features")

# Hierarchical clustering with different linkages
linkage_methods = ['single', 'complete', 'average']
clusterings = {method: linkage(features_scaled, method=method)
               for method in linkage_methods}

# Plotting dendrograms for each linkage method
plt.figure(figsize=(15, 5))
for i, (method, clustering) in enumerate(clusterings.items(), 1):
    plt.subplot(1, 3, i)
    dendrogram(clustering)
    plt.title(f'{method.capitalize()} Linkage')
    plt.xlabel('Sample index')
```

```
    plt.ylabel('Distance')
plt.tight_layout()
plt.show()
```



2. Text Mining

In [ ]:
```
import sklearn.feature_extraction.text as sk_text

corpus = [
    'The sudden rainstorm washed crocodiles into the ocean.',
    'It turns out you dont need all that stuff you insisted you did.',
    'Tomatoes make great weapons when water balloons arent available.',
    'Ive traveled all around Africa and still havent found the gnu who stole my sca
    'The group quickly understood that toxic waste was the most effective barrier t
    'Youve been eyeing me all day and waiting for your move like a lion stalking a
]

vectorizer = sk_text.CountVectorizer(min_df=1)

matrix = vectorizer.fit_transform(corpus)

print(type(matrix))  # Compressed Sparse Row matrix
print(matrix.toarray())  # convert it to numpy array

print(vectorizer.get_feature_names_out())
```

```
<class 'scipy.sparse._csr.csr_matrix'>
[[0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
  1 0 0 1 0 0 0 0 0 0 1 0 2 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1
  0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 3 0 0 0]
 [0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0]
 [1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0
  0 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0
  0 0 1 0 0 0 0 0 0 0 0 0 1 3 1 0 1 0 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 1]
 [0 0 1 1 0 0 0 0 0 0 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 1 0 1 0 0
  0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0]]
['africa' 'against' 'all' 'and' 'arent' 'around' 'available' 'balloons'
 'barrier' 'been' 'crocodiles' 'day' 'did' 'dont' 'effective' 'eyeing'
 'for' 'found' 'gazelle' 'gnu' 'great' 'group' 'havent' 'in' 'insisted'
 'into' 'it' 'ive' 'like' 'lion' 'make' 'me' 'most' 'move' 'my' 'need'
 'ocean' 'out' 'quickly' 'rainstorm' 'savannah' 'scarf' 'stalking' 'still'
 'stole' 'stuff' 'sudden' 'that' 'the' 'to' 'tomatoes' 'toxic' 'traveled'
 'turns' 'understood' 'use' 'waiting' 'was' 'washed' 'waste' 'water'
 'weapons' 'when' 'who' 'you' 'your' 'youve' 'zombies']
```

In [ ]:

```python
vectorizer = sk_text.TfidfVectorizer(
    # stop_words='english',
    # max_features = 1000,
    min_df=1)


# max_features:build a vocabulary that only consider the top max_features features

matrix = vectorizer.fit_transform(corpus)

print(type(matrix))          # Compressed Sparse Row matrix
print(matrix.toarray())  # convert it to numpy array

print(vectorizer.get_feature_names_out())
```

```
<class 'scipy.sparse._csr.csr_matrix'>
[[0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.35539767 0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.35539767 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.35539767 0.         0.         0.35539767 0.         0.
  0.         0.         0.         0.         0.35539767 0.
  0.49209244 0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.35539767 0.
  0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.16249621 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.23471514 0.23471514 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.23471514 0.         0.23471514 0.         0.         0.
  0.         0.         0.         0.         0.         0.23471514
  0.         0.23471514 0.         0.         0.         0.
  0.         0.         0.         0.23471514 0.         0.19246977
  0.         0.         0.         0.         0.         0.23471514
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.70414543 0.
  0.         0.        ]
 [0.         0.         0.         0.         0.33333333 0.
  0.33333333 0.33333333 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.33333333 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.33333333 0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.33333333 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.33333333 0.33333333 0.33333333 0.         0.         0.
  0.         0.        ]
 [0.27085439 0.         0.18751586 0.22210447 0.         0.27085439
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.27085439
  0.         0.27085439 0.         0.         0.27085439 0.
  0.         0.         0.         0.27085439 0.         0.
  0.         0.         0.         0.         0.27085439 0.
  0.         0.         0.         0.         0.         0.27085439
  0.         0.27085439 0.27085439 0.         0.         0.
  0.18751586 0.         0.         0.         0.27085439 0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.27085439 0.         0.
  0.         0.        ]
 [0.         0.23579337 0.         0.         0.         0.
  0.         0.         0.23579337 0.         0.         0.
  0.         0.         0.23579337 0.         0.         0.
  0.         0.         0.         0.23579337 0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.23579337 0.         0.         0.
  0.         0.         0.23579337 0.         0.         0.
```

```
 0.         0.         0.         0.         0.         0.19335394
 0.48972805 0.23579337 0.         0.23579337 0.         0.
 0.23579337 0.23579337 0.         0.23579337 0.         0.23579337
 0.         0.         0.         0.         0.         0.
 0.         0.23579337]
[0.         0.         0.17226329 0.20403846 0.         0.
 0.         0.         0.         0.24882305 0.         0.24882305
 0.         0.         0.         0.24882305 0.24882305 0.
 0.24882305 0.         0.         0.         0.         0.24882305
 0.         0.         0.         0.         0.24882305 0.24882305
 0.         0.24882305 0.         0.24882305 0.         0.
 0.         0.         0.         0.         0.24882305 0.
 0.24882305 0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.24882305 0.         0.         0.
 0.         0.         0.         0.         0.         0.24882305
 0.24882305 0.        ]]
['africa' 'against' 'all' 'and' 'arent' 'around' 'available' 'balloons'
 'barrier' 'been' 'crocodiles' 'day' 'did' 'dont' 'effective' 'eyeing'
 'for' 'found' 'gazelle' 'gnu' 'great' 'group' 'havent' 'in' 'insisted'
 'into' 'it' 'ive' 'like' 'lion' 'make' 'me' 'most' 'move' 'my' 'need'
 'ocean' 'out' 'quickly' 'rainstorm' 'savannah' 'scarf' 'stalking' 'still'
 'stole' 'stuff' 'sudden' 'that' 'the' 'to' 'tomatoes' 'toxic' 'traveled'
 'turns' 'understood' 'use' 'waiting' 'was' 'washed' 'waste' 'water'
 'weapons' 'when' 'who' 'you' 'your' 'youve' 'zombies']
```

In [ ]:

```
# min_df: ignore terms that have a document frequency strictly lower than the given
vectorizer = sk_text.CountVectorizer(min_df=1)

matrix = vectorizer.fit_transform(corpus)

print(type(matrix))  # Compressed Sparse Row matrix
print(matrix.toarray())  # convert it to numpy array

print(vectorizer.get_feature_names_out())
```

```
<class 'scipy.sparse._csr.csr_matrix'>
[[0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
  1 0 0 1 0 0 0 0 0 1 0 2 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1
  0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 3 0 0 0]
 [0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0]
 [1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0
  0 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0
  0 0 1 0 0 0 0 0 0 0 0 1 3 1 0 1 0 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1]
 [0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 1 1 0 1 0 0 0 0 1 0 0 0 0 1 1 0 1 0 1 0 0
  0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0]]
['africa' 'against' 'all' 'and' 'arent' 'around' 'available' 'balloons'
 'barrier' 'been' 'crocodiles' 'day' 'did' 'dont' 'effective' 'eyeing'
 'for' 'found' 'gazelle' 'gnu' 'great' 'group' 'havent' 'in' 'insisted'
 'into' 'it' 'ive' 'like' 'lion' 'make' 'me' 'most' 'move' 'my' 'need'
 'ocean' 'out' 'quickly' 'rainstorm' 'savannah' 'scarf' 'stalking' 'still'
 'stole' 'stuff' 'sudden' 'that' 'the' 'to' 'tomatoes' 'toxic' 'traveled'
 'turns' 'understood' 'use' 'waiting' 'was' 'washed' 'waste' 'water'
 'weapons' 'when' 'who' 'you' 'your' 'youve' 'zombies']
```

In [ ]:

```python
vectorizer = sk_text.TfidfVectorizer(
    stop_words='english',  # remove stop words like 'the', 'a' etc.
    max_features=1000,  # only consider top 1000 words ordered by term frequency
    min_df=1  # ignore terms that have a document frequency < min_df.
)

matrix = vectorizer.fit_transform(corpus)

print(type(matrix))  # Compressed Sparse Row matrix
print(matrix.toarray())  # convert it to numpy array

print(vectorizer.get_feature_names_out())
```

```
<class 'scipy.sparse._csr.csr_matrix'>
[[0.         0.         0.         0.         0.         0.4472136
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.4472136  0.
  0.4472136  0.         0.         0.         0.         0.
  0.4472136  0.         0.         0.         0.         0.
  0.         0.         0.4472136  0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.40824829 0.40824829 0.         0.         0.
  0.         0.         0.         0.         0.40824829 0.
  0.         0.         0.         0.40824829 0.         0.
  0.         0.         0.         0.         0.         0.40824829
  0.         0.         0.         0.         0.40824829 0.
  0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.35355339 0.35355339 0.35355339 0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.35355339 0.         0.         0.         0.
  0.         0.         0.35355339 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.35355339 0.         0.         0.         0.
  0.         0.         0.         0.         0.35355339 0.35355339
  0.         0.        ]
 [0.37796447 0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.37796447 0.         0.         0.37796447 0.         0.37796447
  0.         0.         0.         0.         0.         0.
  0.         0.         0.37796447 0.         0.37796447 0.
  0.         0.         0.         0.37796447 0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.33333333 0.
  0.         0.         0.         0.33333333 0.         0.
  0.         0.         0.33333333 0.         0.         0.
  0.         0.         0.         0.         0.         0.33333333
  0.         0.         0.         0.         0.         0.
  0.         0.         0.33333333 0.         0.         0.33333333
  0.33333333 0.         0.         0.33333333 0.         0.
  0.         0.33333333]
 [0.         0.         0.         0.         0.         0.
  0.33333333 0.         0.         0.         0.33333333 0.33333333
  0.         0.         0.         0.         0.         0.
  0.33333333 0.33333333 0.         0.         0.         0.
  0.         0.33333333 0.         0.33333333 0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.33333333 0.         0.         0.         0.
  0.33333333 0.        ]]
['africa' 'arent' 'available' 'balloons' 'barrier' 'crocodiles' 'day'
 'did' 'dont' 'effective' 'eyeing' 'gazelle' 'gnu' 'great' 'group'
 'havent' 'insisted' 'ive' 'like' 'lion' 'make' 'need' 'ocean' 'quickly'
 'rainstorm' 'savannah' 'scarf' 'stalking' 'stole' 'stuff' 'sudden'
 'tomatoes' 'toxic' 'traveled' 'turns' 'understood' 'use' 'waiting'
 'washed' 'waste' 'water' 'weapons' 'youve' 'zombies']
```

In [ ]:

```
vectorizer = sk_text.TfidfVectorizer(
    stop_words='english',  # remove stop words like 'the', 'a' etc.
    max_features=1000,  # only consider top 1000 words ordered by term frequency
    min_df=1,  # ignore terms that have a document frequency < min_df.
    max_df=0.5  # ignore terms that have a document frequency > max_df.
)

matrix = vectorizer.fit_transform(corpus)
print(type(matrix))  # Compressed Sparse Row matrix

tfidf_data = matrix.toarray()  # convert it to numpy array

print(tfidf_data)  # print the tfidf data
print(vectorizer.get_feature_names_out())  # get the feature names
```

```
<class 'scipy.sparse._csr.csr_matrix'>
[[0.         0.         0.         0.         0.         0.4472136
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.4472136  0.
  0.4472136  0.         0.         0.         0.         0.
  0.4472136  0.         0.         0.         0.         0.
  0.         0.         0.4472136  0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.40824829 0.40824829 0.         0.         0.
  0.         0.         0.         0.         0.40824829 0.
  0.         0.         0.         0.40824829 0.         0.
  0.         0.         0.         0.         0.         0.40824829
  0.         0.         0.         0.         0.40824829 0.
  0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.35355339 0.35355339 0.35355339 0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.35355339 0.         0.         0.         0.
  0.         0.         0.35355339 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.35355339 0.         0.         0.         0.
  0.         0.         0.         0.         0.35355339 0.35355339
  0.         0.        ]
 [0.37796447 0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.37796447 0.         0.         0.37796447 0.         0.37796447
  0.         0.         0.         0.         0.         0.
  0.         0.         0.37796447 0.         0.37796447 0.
  0.         0.         0.         0.37796447 0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.33333333 0.
  0.         0.         0.         0.33333333 0.         0.
  0.         0.         0.33333333 0.         0.         0.
  0.         0.         0.         0.         0.         0.33333333
  0.         0.         0.         0.         0.         0.
  0.         0.         0.33333333 0.         0.         0.33333333
  0.33333333 0.         0.         0.33333333 0.         0.
  0.         0.33333333]
 [0.         0.         0.         0.         0.         0.
  0.33333333 0.         0.         0.         0.33333333 0.33333333
  0.         0.         0.         0.         0.         0.
  0.33333333 0.33333333 0.         0.         0.         0.
  0.         0.33333333 0.         0.33333333 0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.33333333 0.         0.         0.         0.
  0.33333333 0.        ]]
['africa' 'arent' 'available' 'balloons' 'barrier' 'crocodiles' 'day'
 'did' 'dont' 'effective' 'eyeing' 'gazelle' 'gnu' 'great' 'group'
 'havent' 'insisted' 'ive' 'like' 'lion' 'make' 'need' 'ocean' 'quickly'
 'rainstorm' 'savannah' 'scarf' 'stalking' 'stole' 'stuff' 'sudden'
 'tomatoes' 'toxic' 'traveled' 'turns' 'understood' 'use' 'waiting'
 'washed' 'waste' 'water' 'weapons' 'youve' 'zombies']
```

3. Artificial Neural Networks (ANN)

In [ ]:
```
from keras import Sequential
from keras.layers import Dense
from sklearn import preprocessing
from sklearn.metrics import accuracy_score, classification_report
import collections
import numpy as np
import pandas as pd

# Encode text values to dummy variables(i.e. [1,0,0],[0,1,0],[0,0,1] for red,green,

def encode_text_dummy(df, name):
    dummies = pd.get_dummies(df[name])
    for x in dummies.columns:
        dummy_name = "{}-{}".format(name, x)
        df[dummy_name] = dummies[x]
    df.drop(name, axis=1, inplace=True)

# Encode text values to indexes(i.e. [1],[2],[3] for red,green,blue).

def encode_text_index(df, name):
    le = preprocessing.LabelEncoder()
    df[name] = le.fit_transform(df[name])
    return le.classes_

# Convert a Pandas dataframe to the x,y inputs that TensorFlow needs

def to_xy(df, target):
    result = []
    for x in df.columns:
        if x != target:
            result.append(x)
    # find out the type of the target column.
    target_type = df[target].dtypes
    target_type = target_type[0] if isinstance(
        target_type, collections.abc.Sequence) else target_type
    # Encode to int for classification, float otherwise. TensorFlow likes 32 bits.
    if target_type in (np.int64, np.int32):
        # Classification
        dummies = pd.get_dummies(df[target])
        return df[result].values.astype(np.float32), dummies.values.astype(np.float
    else:
        # Regression
        return df[result].values.astype(np.float32), df[target].values.astype(np.fl
```

```
WARNING:tensorflow:From d:\repo\csc177-group-project\.venv\Lib\site-packages\keras\s
rc\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Pl
ease use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [ ]:
```
# Loading the dataset
data_df = pd.read_csv('data/admission_data_set.csv', header='infer')
```

In [ ]:
```
# Dropping unnecessary Columns
data_df = data_df.drop('SOP', axis=1)
data_df = data_df.drop('LOR', axis=1)
data_df = data_df.drop('CGPA', axis=1)
data_df
```

Out[ ]:

	Serial No.	GRE Score	TOEFL Score	University Rating	Research	Chance of Admit
0	1	337	118	4	1	0.92
1	2	324	107	4	1	0.76
2	3	316	104	3	1	0.72
3	4	322	110	3	1	0.80
4	5	314	103	2	0	0.65
...	...	...	...	...	...	...
495	496	332	108	5	1	0.87
496	497	337	117	5	1	0.96
497	498	330	120	5	1	0.93
498	499	312	103	4	0	0.73
499	500	327	113	4	0	0.84

500 rows × 6 columns

In [ ]:
```
# Loading the test data
testdata = data_df[0:100]
testdata
```

Out[ ]:

	Serial No.	GRE Score	TOEFL Score	University Rating	Research	Chance of Admit
0	1	337	118	4	1	0.92
1	2	324	107	4	1	0.76
2	3	316	104	3	1	0.72
3	4	322	110	3	1	0.80
4	5	314	103	2	0	0.65
...	...	...	...	...	...	...
95	96	304	100	4	0	0.42
96	97	306	100	2	0	0.48
97	98	331	120	3	1	0.86
98	99	332	119	4	1	0.90
99	100	323	113	3	1	0.79

100 rows × 6 columns

In [ ]:
```
# Encoding the data based on the Research Column
encode_text_index(testdata, 'Research')
```

```
C:\Users\matrn\AppData\Local\Temp\ipykernel_17184\369207395.py:22: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df[name] = le.fit_transform(df[name])
```

Out[ ]:
```
array([0, 1], dtype=int64)
```

In [ ]:
```
# Splitting the data by the Research Column
X, Y = to_xy(data_df, 'Research')
testX, testY = to_xy(data_df, 'Research')
```

In [ ]:
```
# Generating the ANN model
model = Sequential()
model.add(Dense(12, input_dim=X.shape[1], activation='relu'))
model.add(Dense(6, activation='relu'))
model.add(Dense(2, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
model.fit(X, Y, verbose=2, epochs=100)
```

```
WARNING:tensorflow:From d:\repo\csc177-group-project\.venv\Lib\site-packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From d:\repo\csc177-group-project\.venv\Lib\site-packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

Epoch 1/100
WARNING:tensorflow:From d:\repo\csc177-group-project\.venv\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

16/16 - 1s - loss: 27.6681 - 675ms/epoch - 42ms/step
Epoch 2/100
16/16 - 0s - loss: 17.0982 - 20ms/epoch - 1ms/step
Epoch 3/100
16/16 - 0s - loss: 8.4376 - 16ms/epoch - 1ms/step
Epoch 4/100
16/16 - 0s - loss: 4.1714 - 16ms/epoch - 1ms/step
Epoch 5/100
16/16 - 0s - loss: 1.3584 - 15ms/epoch - 937us/step
Epoch 6/100
16/16 - 0s - loss: 0.8666 - 15ms/epoch - 937us/step
Epoch 7/100
16/16 - 0s - loss: 0.8040 - 15ms/epoch - 937us/step
Epoch 8/100
16/16 - 0s - loss: 0.7475 - 14ms/epoch - 875us/step
Epoch 9/100
16/16 - 0s - loss: 0.7309 - 14ms/epoch - 875us/step
Epoch 10/100
16/16 - 0s - loss: 0.7214 - 14ms/epoch - 882us/step
Epoch 11/100
16/16 - 0s - loss: 0.7155 - 14ms/epoch - 875us/step
Epoch 12/100
16/16 - 0s - loss: 0.7150 - 16ms/epoch - 1000us/step
Epoch 13/100
16/16 - 0s - loss: 0.7150 - 16ms/epoch - 1ms/step
Epoch 14/100
16/16 - 0s - loss: 0.7146 - 20ms/epoch - 1ms/step
Epoch 15/100
16/16 - 0s - loss: 0.7285 - 19ms/epoch - 1ms/step
Epoch 16/100
16/16 - 0s - loss: 0.7232 - 20ms/epoch - 1ms/step
Epoch 17/100
16/16 - 0s - loss: 0.7124 - 17ms/epoch - 1ms/step
Epoch 18/100
16/16 - 0s - loss: 0.7120 - 17ms/epoch - 1ms/step
Epoch 19/100
16/16 - 0s - loss: 0.7115 - 17ms/epoch - 1ms/step
Epoch 20/100
16/16 - 0s - loss: 0.7131 - 16ms/epoch - 1ms/step
Epoch 21/100
16/16 - 0s - loss: 0.7098 - 16ms/epoch - 969us/step
Epoch 22/100
16/16 - 0s - loss: 0.7080 - 17ms/epoch - 1ms/step
```

```
Epoch 23/100
16/16 - 0s - loss: 0.7098 - 18ms/epoch - 1ms/step
Epoch 24/100
16/16 - 0s - loss: 0.7118 - 16ms/epoch - 1ms/step
Epoch 25/100
16/16 - 0s - loss: 0.7100 - 15ms/epoch - 938us/step
Epoch 26/100
16/16 - 0s - loss: 0.7094 - 21ms/epoch - 1ms/step
Epoch 27/100
16/16 - 0s - loss: 0.7129 - 21ms/epoch - 1ms/step
Epoch 28/100
16/16 - 0s - loss: 0.7030 - 19ms/epoch - 1ms/step
Epoch 29/100
16/16 - 0s - loss: 0.7044 - 19ms/epoch - 1ms/step
Epoch 30/100
16/16 - 0s - loss: 0.7050 - 16ms/epoch - 998us/step
Epoch 31/100
16/16 - 0s - loss: 0.7218 - 16ms/epoch - 969us/step
Epoch 32/100
16/16 - 0s - loss: 0.6979 - 15ms/epoch - 937us/step
Epoch 33/100
16/16 - 0s - loss: 0.7035 - 15ms/epoch - 938us/step
Epoch 34/100
16/16 - 0s - loss: 0.7044 - 15ms/epoch - 938us/step
Epoch 35/100
16/16 - 0s - loss: 0.7178 - 16ms/epoch - 1000us/step
Epoch 36/100
16/16 - 0s - loss: 0.7023 - 18ms/epoch - 1ms/step
Epoch 37/100
16/16 - 0s - loss: 0.7003 - 23ms/epoch - 1ms/step
Epoch 38/100
16/16 - 0s - loss: 0.6996 - 20ms/epoch - 1ms/step
Epoch 39/100
16/16 - 0s - loss: 0.7037 - 21ms/epoch - 1ms/step
Epoch 40/100
16/16 - 0s - loss: 0.7005 - 19ms/epoch - 1ms/step
Epoch 41/100
16/16 - 0s - loss: 0.6998 - 18ms/epoch - 1ms/step
Epoch 42/100
16/16 - 0s - loss: 0.7021 - 20ms/epoch - 1ms/step
Epoch 43/100
16/16 - 0s - loss: 0.7015 - 23ms/epoch - 1ms/step
Epoch 44/100
16/16 - 0s - loss: 0.7126 - 18ms/epoch - 1ms/step
Epoch 45/100
16/16 - 0s - loss: 0.7121 - 20ms/epoch - 1ms/step
Epoch 46/100
16/16 - 0s - loss: 0.7042 - 18ms/epoch - 1ms/step
Epoch 47/100
16/16 - 0s - loss: 0.7132 - 46ms/epoch - 3ms/step
Epoch 48/100
16/16 - 0s - loss: 0.6948 - 39ms/epoch - 2ms/step
Epoch 49/100
16/16 - 0s - loss: 0.7014 - 21ms/epoch - 1ms/step
Epoch 50/100
16/16 - 0s - loss: 0.7051 - 17ms/epoch - 1ms/step
```

```
Epoch 51/100
16/16 - 0s - loss: 0.6967 - 16ms/epoch - 1000us/step
Epoch 52/100
16/16 - 0s - loss: 0.6982 - 15ms/epoch - 937us/step
Epoch 53/100
16/16 - 0s - loss: 0.6907 - 15ms/epoch - 938us/step
Epoch 54/100
16/16 - 0s - loss: 0.7083 - 14ms/epoch - 882us/step
Epoch 55/100
16/16 - 0s - loss: 0.6971 - 15ms/epoch - 938us/step
Epoch 56/100
16/16 - 0s - loss: 0.6973 - 15ms/epoch - 938us/step
Epoch 57/100
16/16 - 0s - loss: 0.7060 - 15ms/epoch - 938us/step
Epoch 58/100
16/16 - 0s - loss: 0.6967 - 17ms/epoch - 1ms/step
Epoch 59/100
16/16 - 0s - loss: 0.6950 - 17ms/epoch - 1ms/step
Epoch 60/100
16/16 - 0s - loss: 0.6982 - 18ms/epoch - 1ms/step
Epoch 61/100
16/16 - 0s - loss: 0.6969 - 24ms/epoch - 1ms/step
Epoch 62/100
16/16 - 0s - loss: 0.6983 - 20ms/epoch - 1ms/step
Epoch 63/100
16/16 - 0s - loss: 0.6942 - 17ms/epoch - 1ms/step
Epoch 64/100
16/16 - 0s - loss: 0.6920 - 14ms/epoch - 875us/step
Epoch 65/100
16/16 - 0s - loss: 0.6939 - 16ms/epoch - 969us/step
Epoch 66/100
16/16 - 0s - loss: 0.6904 - 15ms/epoch - 937us/step
Epoch 67/100
16/16 - 0s - loss: 0.6971 - 15ms/epoch - 937us/step
Epoch 68/100
16/16 - 0s - loss: 0.6958 - 15ms/epoch - 938us/step
Epoch 69/100
16/16 - 0s - loss: 0.7023 - 15ms/epoch - 938us/step
Epoch 70/100
16/16 - 0s - loss: 0.6895 - 16ms/epoch - 1ms/step
Epoch 71/100
16/16 - 0s - loss: 0.6898 - 16ms/epoch - 970us/step
Epoch 72/100
16/16 - 0s - loss: 0.6947 - 15ms/epoch - 938us/step
Epoch 73/100
16/16 - 0s - loss: 0.6895 - 15ms/epoch - 938us/step
Epoch 74/100
16/16 - 0s - loss: 0.6869 - 15ms/epoch - 938us/step
Epoch 75/100
16/16 - 0s - loss: 0.6864 - 16ms/epoch - 1ms/step
Epoch 76/100
16/16 - 0s - loss: 0.6860 - 15ms/epoch - 937us/step
Epoch 77/100
16/16 - 0s - loss: 0.6917 - 20ms/epoch - 1ms/step
Epoch 78/100
16/16 - 0s - loss: 0.6929 - 17ms/epoch - 1ms/step
```

```
Epoch 79/100
16/16 - 0s - loss: 0.6939 - 15ms/epoch - 938us/step
Epoch 80/100
16/16 - 0s - loss: 0.7075 - 15ms/epoch - 938us/step
Epoch 81/100
16/16 - 0s - loss: 0.7122 - 14ms/epoch - 875us/step
Epoch 82/100
16/16 - 0s - loss: 0.7126 - 15ms/epoch - 938us/step
Epoch 83/100
16/16 - 0s - loss: 0.6860 - 15ms/epoch - 964us/step
Epoch 84/100
16/16 - 0s - loss: 0.6921 - 18ms/epoch - 1ms/step
Epoch 85/100
16/16 - 0s - loss: 0.6885 - 19ms/epoch - 1ms/step
Epoch 86/100
16/16 - 0s - loss: 0.6932 - 17ms/epoch - 1ms/step
Epoch 87/100
16/16 - 0s - loss: 0.6901 - 16ms/epoch - 1ms/step
Epoch 88/100
16/16 - 0s - loss: 0.6912 - 15ms/epoch - 907us/step
Epoch 89/100
16/16 - 0s - loss: 0.7079 - 17ms/epoch - 1ms/step
Epoch 90/100
16/16 - 0s - loss: 0.7098 - 15ms/epoch - 937us/step
Epoch 91/100
16/16 - 0s - loss: 0.7041 - 14ms/epoch - 875us/step
Epoch 92/100
16/16 - 0s - loss: 0.6933 - 14ms/epoch - 875us/step
Epoch 93/100
16/16 - 0s - loss: 0.6931 - 15ms/epoch - 938us/step
Epoch 94/100
16/16 - 0s - loss: 0.6905 - 16ms/epoch - 970us/step
Epoch 95/100
16/16 - 0s - loss: 0.6950 - 15ms/epoch - 937us/step
Epoch 96/100
16/16 - 0s - loss: 0.6906 - 16ms/epoch - 1000us/step
Epoch 97/100
16/16 - 0s - loss: 0.7084 - 15ms/epoch - 938us/step
Epoch 98/100
16/16 - 0s - loss: 0.7022 - 15ms/epoch - 938us/step
Epoch 99/100
16/16 - 0s - loss: 0.6965 - 16ms/epoch - 1ms/step
Epoch 100/100
16/16 - 0s - loss: 0.6970 - 15ms/epoch - 938us/step
```

Out[ ]: <keras.src.callbacks.History at 0x1f11508f790>

In [ ]:
```
pred = model.predict(testX)
print(pred[0])
```

```
16/16 [==============================] - 0s 1ms/step
[0.36167884 0.6383211 ]
```

In [ ]:
```
pred = np.argmax(pred, axis=1)
true = np.argmax(testY, axis=1)
print('Accuracy on test data is %.2f' % (accuracy_score(true, pred)))
```

Accuracy on test data is 0.59

In [ ]:
```
print(classification_report(true, pred))
```

```
              precision    recall  f1-score   support

           0       0.65      0.14      0.23       220
           1       0.58      0.94      0.72       280

    accuracy                           0.59       500
   macro avg       0.62      0.54      0.47       500
weighted avg       0.61      0.59      0.50       500
```