In [1]:
```
!pip install PyPDF2
```

```
Collecting PyPDF2
  Downloading pypdf2-3.0.1-py3-none-any.whl (232 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 232.6/232.6 kB 4.0 M
B/s eta 0:00:00 0:00:01
Installing collected packages: PyPDF2
Successfully installed PyPDF2-3.0.1
```

In [2]:
```
!pip install python-docx
```

```
Collecting python-docx
  Downloading python_docx-1.1.2-py3-none-any.whl (244 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 244.3/244.3 kB 5.2 M
B/s eta 0:00:00a 0:00:01
Requirement already satisfied: lxml>=3.1.0 in /usr/local/lib/pytho
n3.10/dist-packages (from python-docx) (4.9.4)
Requirement already satisfied: typing-extensions>=4.9.0 in /usr/lo
cal/lib/python3.10/dist-packages (from python-docx) (4.11.0)
Installing collected packages: python-docx
Successfully installed python-docx-1.1.2
```

In [3]:
```
!pip install contractions
```

```
Collecting contractions
  Downloading contractions-0.1.73-py2.py3-none-any.whl (8.7 kB)
Collecting textsearch>=0.0.21 (from contractions)
  Downloading textsearch-0.0.24-py2.py3-none-any.whl (7.6 kB)
Collecting anyascii (from textsearch>=0.0.21->contractions)
  Downloading anyascii-0.3.2-py3-none-any.whl (289 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 289.9/289.9 kB 7.6 M
B/s eta 0:00:00
Collecting pyahocorasick (from textsearch>=0.0.21->contractions)
  Downloading pyahocorasick-2.1.0-cp310-cp310-manylinux_2_5_x86_6
4.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl
(110 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 110.7/110.7 kB 13.1
MB/s eta 0:00:00
Installing collected packages: pyahocorasick, anyascii, textsearc
h, contractions
Successfully installed anyascii-0.3.2 contractions-0.1.73 pyahocor
asick-2.1.0 textsearch-0.0.24
```

In [4]:
```
!pip install unidecode
```

```
Collecting unidecode
  Downloading Unidecode-1.3.8-py3-none-any.whl (235 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 235.5/235.5 kB 4.4 M
B/s eta 0:00:00a 0:00:01
Installing collected packages: unidecode
Successfully installed unidecode-1.3.8
```

In [5]:
```
import os
import PyPDF2
import pandas as pd
import docx
import re
import nltk
import contractions
import unidecode
import numpy as np
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer, PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from gensim.models import Word2Vec
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_
from sklearn.cluster import DBSCAN
from collections import Counter
from sklearn.metrics import silhouette_score
from sklearn.svm import LinearSVC
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.decomposition import TruncatedSVD
import warnings
warnings.filterwarnings('ignore')
```

In [6]:
```
#function to store PDF data
def store_pdf_data(file_path):
  pdf_reader = PyPDF2.PdfReader(file)
  content = ""
  for page_number in range(len(pdf_reader.pages)):
    content += pdf_reader.pages[page_number].extract_text()
  return content
```

In [7]:
```
#function to store DOCX data
def store_doc_data(file_path):
  doc = docx.Document(file_path)
  content = ""
  for paragraph in doc.paragraphs:
    content += paragraph.text
  return content
```

In [8]:
```
# storing all the data extracted from files into a dataframe
path = "/content/Data"
whole_content=[]
for filename in os.listdir(path):
  row_data = {}
  file_path = os.path.join(path, filename)
  row_data["file_name"]=filename
  if filename.endswith(".pdf"):
    with open(file_path, "rb") as file:
      df_content = store_pdf_data(file_path)
    row_data["content"]=df_content
    row_data["file_type"]="PDF"
    row_data["label"]= filename.split('_')[0]
  elif filename.endswith(".txt"):
    with open(file_path, "r") as file:
      df_content = file.read()
    row_data["content"]=df_content
    row_data["file_type"]="TXT"
    row_data["label"]= filename.split('_')[0]
  elif filename.endswith(".docx"):
    df_content = store_doc_data(file_path)
    row_data["content"]=df_content
    row_data["file_type"]="DOCX"
    row_data["label"]= filename.split('_')[0]
  whole_content.append(row_data)
```

In [9]:
```
df = pd.DataFrame(whole_content)
```

Text cleaning

In [10]:
```
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

Out[10]: True

In [11]:
```
# function to perform text cleaning
def clean_text(text):

  #removing contractions
  text = contractions.fix(text)

  #making the text to lowercase
  text = text.lower()

  #removing non-alphabetical charecters
  text = re.sub(r'[^a-zA-Z\s]', '', text)

  #decoding the encoded data
  text = unidecode.unidecode(text)

  #performing tokenization
  tokens = nltk.word_tokenize(text)

  #removing stopwords
  stop_words = set(stopwords.words('english'))
  tokens = [word for word in tokens if word not in stop_words]

  #performing lemmatization
  lemmatizer = WordNetLemmatizer()
  tokens = [lemmatizer.lemmatize(word) for word in tokens]

  return tokens
```

In [13]:
```
df['cleaned_content'] = df['content'].apply(clean_text)
```

In [14]:
```
#encoding the labels to numerical classification
label_encoding = {
    'education': 0,
    'health': 1,
    'entertainment': 2,
}
df['encoding'] = df['label'].map(label_encoding)
X=df['cleaned_content']
y=df['encoding']
```

In [15]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,train_size = 0
```

Vectorization

In [20]:
```
vector_size = 100
window = 5
min_count = 3
workers = 4

#considering word2vec model to create word embeddings
model = Word2Vec(sentences=X_train, vector_size=vector_size, window

model.save("word2vec.model")
```

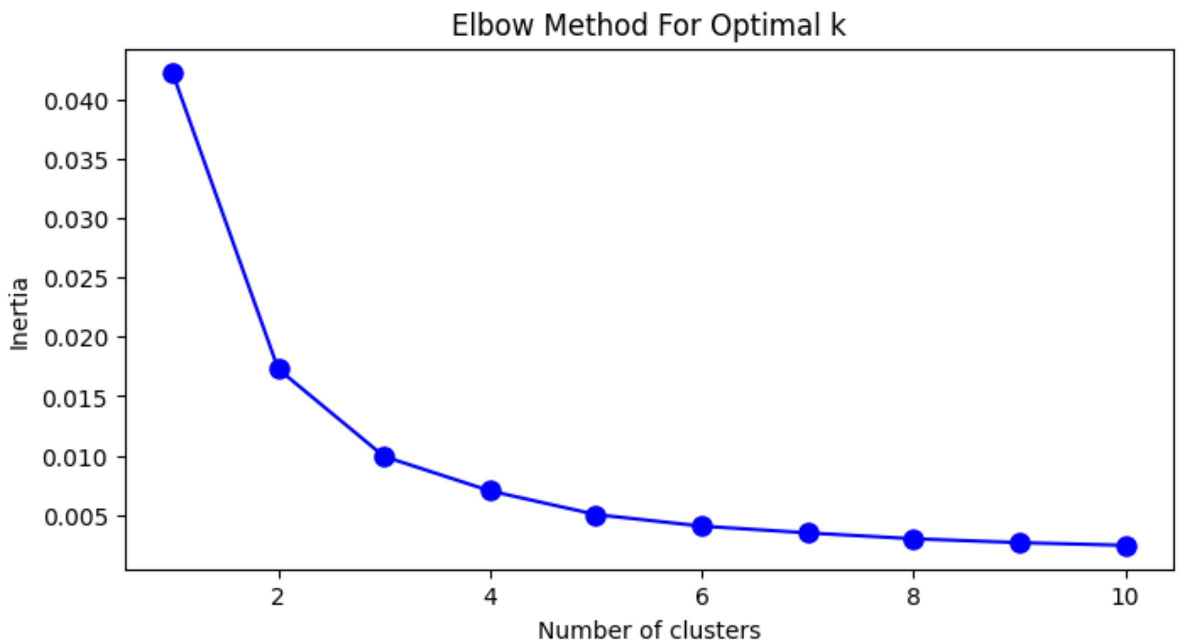
In [21]:
```
#function creating sentence vectors using the word embeddings
def sentence_vector(words, model):
    word_vectors = [model.wv[word] for word in words if word in mod
    if not word_vectors:
        return np.zeros(model.vector_size)
    return np.mean(word_vectors, axis=0)
```

In [22]:
```
train_sentence_vectors = [sentence_vector(words, model) for words i
```

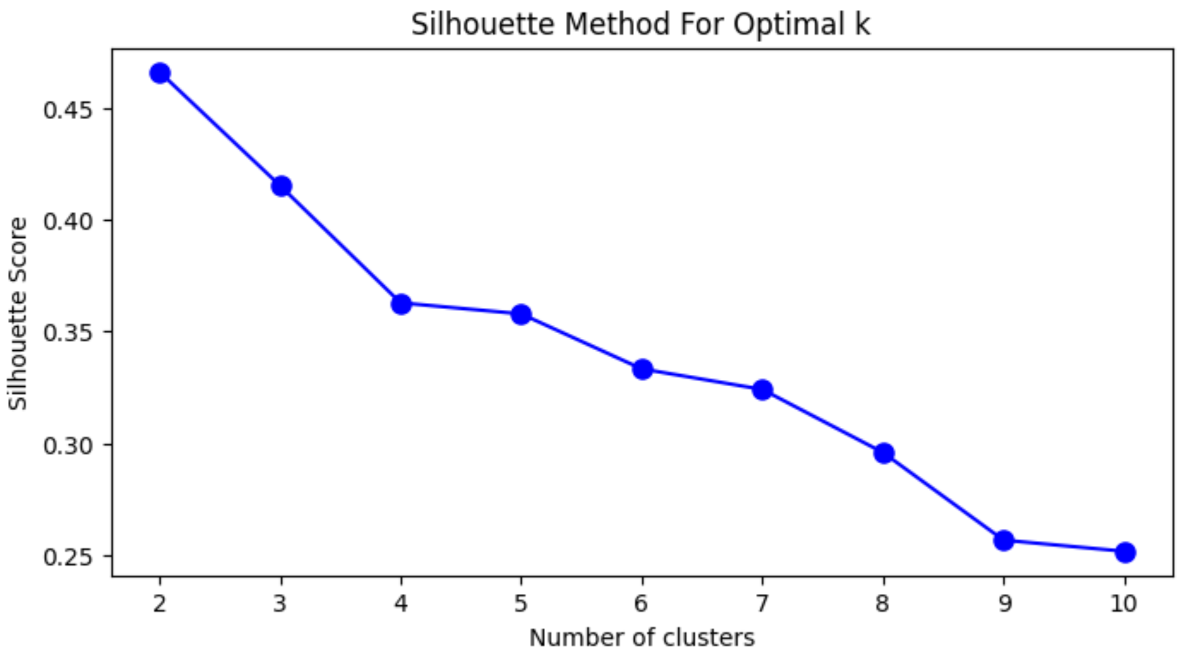
In [23]:
```
test_sentence_vectors = [sentence_vector(words, model) for words in
```

##Clustering In order to segregate the files based on the context and metadata, we should determine the optimal 'k'

In [24]:
```
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.metrics import silhouette_score

# Elbow Method
inertia = []
K = range(1, 11)

for k in K:
  kmeans = KMeans(n_clusters=k, random_state=42)
  kmeans.fit(train_sentence_vectors)
  inertia.append(kmeans.inertia_)

# Plot
plt.figure(figsize=(8, 4))
plt.plot(K, inertia, 'bo-', markersize=8)
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method For Optimal k')
plt.show()
```



In [25]:
```
# Silhouette Method
silhouette_scores = []
K = range(2, 11)

for k in K:
  kmeans = KMeans(n_clusters=k, random_state=42)
  cluster_labels = kmeans.fit_predict(train_sentence_vectors)
  silhouette_avg = silhouette_score(train_sentence_vectors, cluster
  silhouette_scores.append(silhouette_avg)

# Plot
plt.figure(figsize=(8, 4))
plt.plot(K, silhouette_scores, 'bo-', markersize=8)
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Method For Optimal k')
plt.show()
```



- Here, the best k based on Elbow method is 3
- Here, the best k based on Silhouette score is 2

In [26]:
```
#performing kmeans clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(train_sentence_vectors)
labels = kmeans.labels_
```

In [27]:
```
test_cluster_labels = kmeans.predict(test_sentence_vectors)
```

In [28]:
```
test_cluster_labels
```

Out[28]:
```
array([1, 2, 1, 0, 1, 0, 2, 1, 2, 1, 2, 2], dtype=int32)
```

In [ ]:
```
all_cluster_labels = np.concatenate([labels, test_cluster_labels])
cluster_points = {label: [] for label in set(all_cluster_labels)}
for i, label in enumerate(all_cluster_labels):
    cluster_points[label].append(i)

for cluster_label, points in cluster_points.items():
    print(f'Cluster {cluster_label}:')
    for point in points:
        print(f'- {os.listdir(path)[point]}')
```

In [30]:
```
all_cluster_labels = np.concatenate([labels, test_cluster_labels])
cluster_counts = {label: 0 for label in set(all_cluster_labels)}
for label in all_cluster_labels:
    cluster_counts[label] += 1

for cluster_label, count in cluster_counts.items():
    print(f'Cluster {cluster_label}: {count} points')
```

```
Cluster 0: 13 points
Cluster 1: 28 points
Cluster 2: 19 points
```

In [31]:
```
#Based on the Silhouette score, we can say that the clustering is d
silhouette = silhouette_score(train_sentence_vectors, labels)
print("Silhouette Score:", silhouette)
```

```
Silhouette Score: 0.41539642
```

Classification

If the labels of the datapoints are given, we perfrom classification as it is supervised machine learning algorithm.

In [32]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_

clf_log = LogisticRegression()
clf_dt = DecisionTreeClassifier()
clf_rf = RandomForestClassifier()

# A family of models are considered to perform the classification i
models = {
    'Logistic Regression': clf_log,
    'Decision Tree' : clf_dt,
    'Random Forest': clf_rf,
}

# model evaluations are perfromed respectively and the results are
def evaluate_model(model, X_train, X_test, y_train, y_test):
  model.fit(train_sentence_vectors, y_train)
  y_pred_train = model.predict(train_sentence_vectors)
  y_pred_test = model.predict(test_sentence_vectors)

  metrics = {
      'Test Accuracy': accuracy_score(y_test, y_pred_test),
      'Test Precision': precision_score(y_test, y_pred_test, averag
      'Test Recall': recall_score(y_test, y_pred_test, average='wei
      'Test F1 Score': f1_score(y_test, y_pred_test, average='weigh
    }
  return metrics
```

In [58]:
```
results = {}

for model_name, model in models.items():
  metrics = evaluate_model(model, train_sentence_vectors, test_sent
  results[model_name] = metrics

results_df = pd.DataFrame(results).T
results_df.columns = ['Accuracy', 'Precision', 'Recall', 'F1 Score'
results_df['Algorithm'] = results_df.index
results_df = results_df.reset_index(drop=True)

print(results_df)
```

```
   Accuracy  Precision    Recall  F1 Score            Algorithm
0  0.500000   0.257143  0.500000  0.337500  Logistic Regression
1  0.666667   0.857143  0.666667  0.650000        Decision Tree
2  0.916667   0.937500  0.916667  0.918831        Random Forest
```

In [59]:
```
results_df
```

Out[59]:

	Accuracy	Precision	Recall	F1 Score	Algorithm
0	0.500000	0.257143	0.500000	0.337500	Logistic Regression
1	0.666667	0.857143	0.666667	0.650000	Decision Tree
2	0.916667	0.937500	0.916667	0.918831	Random Forest

In [60]:
```
results_df.set_index('Algorithm', inplace=True)

ax = results_df.plot(kind='bar', figsize=(12, 6))

for p in ax.patches:
    ax.annotate(f'{p.get_height():.2f}', (p.get_x() * 1.005, p.get_

plt.title('Comparison of Classification Algorithms')
plt.ylabel('Score')
plt.xlabel('Metric')
plt.ylim(0, 1)
plt.legend(title='Algorithm')
plt.show()

#Therefore, we can say that Random Forest classifier works best amo
```