AI VIET NAM
@aivietnam.edu.vn

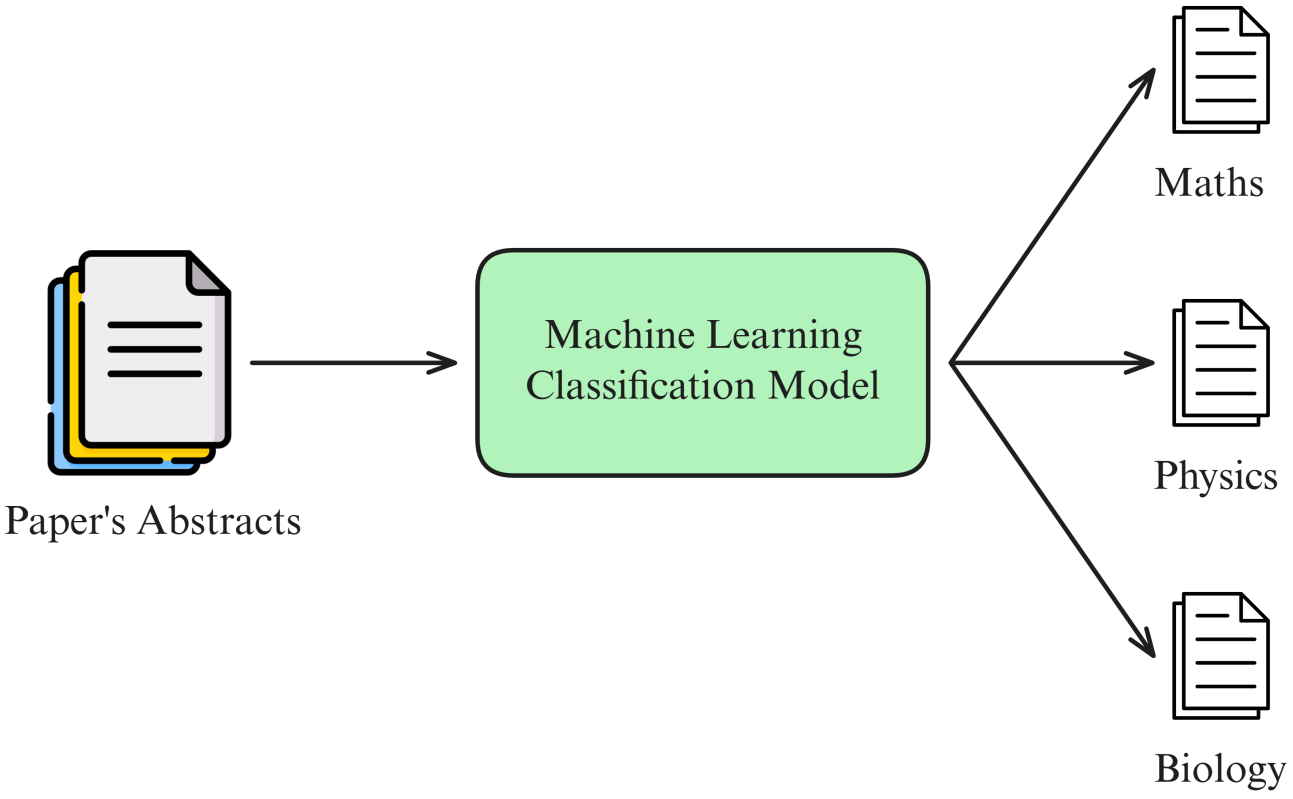
AI VIET NAM – AI COURSE 2025

Phân loại chủ đề bài báo dựa vào thuật toán KNN, K-Means và Decision Tree

Trần Minh Nam Nguyễn Quốc Thái Đinh Quang Vinh

I. Giới thiệu

Text Classification (Tạm dịch: Phân loại văn bản) là một trong những bài toán cơ bản và quan trọng trong lĩnh vực Xử lý ngôn ngữ tự nhiên (NLP). Mục tiêu của bài toán này là phân loại các đoạn văn bản vào các nhóm hoặc nhãn khác nhau dựa trên nội dung của chúng. Các ứng dụng phổ biến của Text Classification bao gồm phân loại email (spam vs. ham), phân loại tin tức, phân loại cảm xúc (sentiment analysis), và nhiều ứng dụng khác.



Hình 1: Minh họa ứng dụng phân loại topic của paper dựa trên abstract.

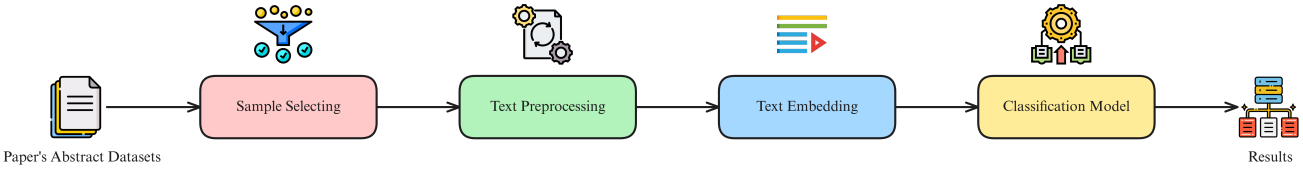
Trong project này, chúng ta sẽ xây dựng một chương trình Text Classification liên quan đến việc phân loại một abstract của publication (bài báo khoa học) thành các topic khác nhau. Chương trình sẽ được xây dựng trên nền tảng Python và sử dụng các thư viện học máy phổ biến như scikit-learn, numpy, v.v.
Theo đó, Input/Output chung của chương trình sẽ bao gồm:

- **Input:** Một abstract của publication (bài báo khoa học).
- **Output:** Topic của abstract đó (ví dụ: vật lý, toán học, khoa học máy tính, v.v.).

Mục lục

II. Xây dựng chương trình phân loại topic của publication abstract

Trong phần này, ta sẽ tiến hành cài đặt chương trình phân loại topic của publication abstract. Chương trình sẽ được xây dựng dựa trên ba phương pháp mã hóa văn bản khác nhau, bao gồm Bag-of-Words (BoW), TF-IDF và Sentence Embeddings. Mỗi phương pháp sẽ được áp dụng với một mô hình phân loại khác nhau, bao gồm Naive Bayes, KNN và Decision Tree.



Hình 2: Pipeline

II.1. Cài đặt và Import thư viện

Chúng ta import các thư viện cần thiết để thực hiện các bước tiền xử lý, trích xuất đặc trưng, huấn luyện mô hình và đánh giá mô hình. Các thư viện này bao gồm:

```
from collections import Counter, defaultdict
from typing import List, Dict, Literal, Union

import re
import math
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from datasets import load_dataset
from sentence_transformers import SentenceTransformer

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
```

Trước tiên, chúng ta cùng điểm qua một số thư viện chính được sử dụng trong bài:

- **re:** Thư viện để làm việc với biểu thức chính quy (regular expressions), giúp xử lý và phân tích chuỗi văn bản.
- **datasets:** Thư viện để tải và làm việc với các bộ dữ liệu từ Hugging Face.

- **sentence-transformers:** Thư viện để tạo và sử dụng các mô hình sentence embeddings, giúp chuyển đổi văn bản thành các vector ngữ nghĩa.
- **matplotlib.pyplot:** Thư viện để vẽ đồ thị và biểu đồ, hỗ trợ trực quan hóa dữ liệu.
- **seaborn:** Thư viện để vẽ đồ thị thống kê, cung cấp các hàm tiện ích để trực quan hóa dữ liệu một cách đẹp mắt.
- **numpy:** Cung cấp các đối tượng mảng đa chiều và các hàm toán học để làm việc với các mảng này.
- **scikit-learn:** Thư viện học máy phổ biến, giúp xây dựng và triển khai các mô hình học máy phức tạp một cách nhanh chóng.

II.2. Đọc và khám phá bộ dữ liệu

Trong bài này, chúng ta sẽ sử dụng bộ dữ liệu UniverseTBD/arxiv-abstracts-large được cung cấp trên HuggingFace. Bộ dữ liệu này bao gồm các abstract (tóm tắt) của các bài báo khoa học được đăng trên arXiv, một kho lưu trữ trực tuyến cho các bài báo khoa học trong nhiều lĩnh vực khác nhau. Bộ dữ liệu này có thể được sử dụng để phân loại các abstract theo các chủ đề khác nhau hoặc để phân tích nội dung của các bài báo khoa học, bao gồm các thông tin như tiêu đề, tác giả, ngày cập nhật, v.v. Bộ dữ liệu này có kích thước lớn với hơn 2 triệu bản ghi.
Chúng ta load bộ dữ liệu này bằng thư viện `datasets` với hàm `load_dataset` và in ra một số thông tin cơ bản về bộ dữ liệu như sau:

```
ds = load_dataset("UniverseTBD/arxiv-abstracts-large")
ds
```

Nội dung được in ra màn hình như sau:

```
DatasetDict(
train: Dataset(
features: ['id', 'submitter', 'authors', 'title', 'comments', 'journal-ref', 'doi', 'report-no',
'categories', 'license', 'abstract', 'versions', 'update_date', 'authors_parsed'],
num_rows: 2292057
)
)
```

Các fields của bộ dữ liệu bao gồm:

- **id:** ID của paper trên arXiv.
- **submitter:** Người nộp bài báo.
- **authors:** Danh sách các tác giả của bài báo.
- **title:** Tiêu đề của bài báo.

- **comments:** Các bình luận liên quan đến bài báo.
- **journal-ref:** Tham chiếu đến tạp chí nơi bài báo được xuất bản.
- **doi:** Digital Object Identifier, mã định danh duy nhất cho bài báo.
- **report-no:** Số báo cáo liên quan đến bài báo.
- **categories:** Các chủ đề hoặc lĩnh vực mà bài báo thuộc về.
- **license:** Giấy phép sử dụng của bài báo.
- **abstract:** Tóm tắt nội dung của bài báo (đây là field chúng ta sẽ sử dụng để phân loại).
- **versions:** Phiên bản của bài báo.
- **update_date:** Ngày cập nhật của bài báo.
- **authors_parsed:** Danh sách các tác giả đã được phân tích và chuẩn hóa.

Chúng ta sẽ sử dụng field `abstract` để làm input cho mô hình phân loại, và field `categories` để làm nhãn (label) cho mô hình.
Để quan sát dữ liệu, chúng ta in ra màn hình 3 bản ghi đầu tiên của bộ dữ liệu như sau:

```
# print three first examples
for i in range(3):
    print(f"Example {i+1}:")
    print(ds['train'][i]['abstract'])
    print(ds['train'][i]['categories'])
    print("---" * 20)
```

Dựa vào dữ liệu ta thấy dữ liệu abstract của chúng ta có chứa nhiều kí tự đặc biệt như dấu chấm, dấu phẩy, dấu ngoặc kép, v.v. Ngoài ra, các abstract này cũng có độ dài khác nhau, có thể từ vài câu đến vài đoạn văn. Các nhãn (categories) của các abstract này cũng rất đa dạng, bao gồm nhiều lĩnh vực khác nhau như vật lý, toán học, khoa học máy tính, v.v. Chúng ta in toàn bộ các categories của abstract để xem xét các nhãn này có phù hợp với bài toán phân loại hay không như sau:

```
all_categories = ds['train']['categories']
print(set(all_categories))
```

Categories của abstract tuân theo format `<primary category> <secondary category> ...`, ví dụ như `hep-ph` hay `math.CO cs.CG`. Chúng ta sẽ sử dụng field `abstract` để làm input cho mô hình phân loại, và phần primary category (category đầu tiên) trong field `categories` để làm nhãn (label) cho mô hình.
Để quan sát số lượng primary categories trong bộ dữ liệu, chúng ta sẽ đếm số lượng các nhãn này và in ra như sau:

```python
all_categories = ds['train']['categories']
category_set = set()

# Collect unique labels
for category in all_categories:
    parts = category.split(' ')
    for part in parts:
        topic = part.split('.')[0]
        category_set.add(topic)

# Sort the labels and print them
sorted_categories= sorted(list(category_set), key=lambda x: x.lower())
print(f'There are {len(sorted_categories)} unique primary categories in the dataset:')
for category in sorted_categories:
    print(category)
```

Kết quả in ra sẽ là danh sách các primary categories trong bộ dữ liệu như sau:

```
There are 38 unique primary categories in the dataset:
acc-phys
adap-org
alg-geom
ao-sci
astro-ph
atom-ph
bayes-an
chao-dyn
chem-ph
cmp-lg
comp-gas
cond-mat
cs
...
```

Chúng ta sẽ lấy 1000 samples, các samples này có primary là một trong 5 categories sau: `astro-ph`, `cond-mat`, `cs`, `math`, `physics`.

```python
# load 1000 samples with single label belonging to specific categories
samples = []
CATEGORIES_TO_SELECT = ['astro-ph', 'cond-mat', 'cs', 'math', 'physics']
for s in ds['train']:
    if len(s['categories'].split(' ')) != 1:
        continue

    cur_category = s['categories'].strip().split('.')[0]
    if cur_category not in CATEGORIES_TO_SELECT:
        continue

```

```
    samples.append(s)

    if len(samples) >= 1000:
        break
print(f"Number of samples: {len(samples)}") # Number of samples: 1000
```

II.3. Tiền xử lý dữ liệu

Tiền xử lý dữ liệu đặc trưng: Nội dung của abstract có chứa nhiều kí tự đặc biệt (newline, trailing space, v.v.), viết hoa, cũng như dấu câu. Do đó, chúng ta cần thực hiện các bước tiền xử lý để chuẩn hóa dữ liệu.
Đối với mỗi sample trong danh sách 1000 samples chúng ta đã chọn ra, chúng ta sẽ thực hiện các bước tiền xử lý như sau:

- Loại bỏ các kí tự `\n` và khoảng trắng ở đầu và cuối chuỗi.
- Loại bỏ các kí tự đặc biệt (dấu câu, kí tự không phải chữ cái hoặc số).
- Loại bỏ các chữ số.
- Chuyển đổi tất cả các chữ cái thành chữ thường (lowercase).
- Lấy nhãn (label) là primary category (phần đầu tiên) trong field `categories`.

Các bước tiền xử lý trên tương ứng với đoạn code sau:

```
preprocessed_samples = []
for s in samples:
    abstract = s['abstract']

    # Remove \n characters in the middle and leading/trailing spaces
    abstract = abstract.strip().replace("\n", " ")

    # Remove special characters
    abstract = re.sub(r'[^\w\s]', '', abstract)

    # Remove digits
    abstract = re.sub(r'\d+', '', abstract)

    # Remove extra spaces
    abstract = re.sub(r'\s+', ' ', abstract).strip()

    # Convert to lower case
    abstract = abstract.lower()

    # for the label, we only keep the first part
    parts = s['categories'].split(' ')
    category = parts[0].split('.')[0]

    preprocessed_samples.append({
```

```
        "text": abstract,
        "label": category
    })
```

Tiếp theo, chúng ta cần tạo hai dictionaries để lưu trữ các primary categories dưới dạng số và ngược lại bằng đoạn code sau:

```
label_to_id = {label: i for i, label in enumerate(sorted_labels)}
id_to_label = {i: label for i, label in enumerate(sorted_labels)}

# Print label to ID mapping
print("Label to ID mapping:")
for label, id_ in label_to_id.items():
    print(f"{label} --> {id_}")
```

Kết quả in ra sẽ là danh sách các nhãn và ID tương ứng như sau:

Label to ID mapping:
astro-ph –> 0
cond-mat –> 1
cs –> 2
math –> 3
physics –> 4

Cuối cùng, chúng ta sẽ chia dữ liệu trên thành 2 tập: tập huấn luyện (train) và tập kiểm tra (test) với tỉ lệ 80% cho tập huấn luyện và 20% cho tập kiểm tra. Chúng ta sẽ sử dụng hàm `train_test_split` từ thư viện `sklearn` để thực hiện việc này.

```
X_full = [sample['text'] for sample in preprocessed_samples]
y_full = [label_to_id[sample['label']] for sample in preprocessed_samples]

X_train, X_test, y_train, y_test = train_test_split(X_full, y_full, test_size=0.2,
                                                    random_state=42, stratify=y_full)

print(f"Training samples: {len(X_train)}")
print(f"Test samples: {len(X_test)}")
```

Số lượng mẫu trong tập huấn luyện và tập kiểm tra sẽ được in ra như sau:

Training samples: 800
Test samples: 200

II.4. Mã hóa văn bản

Văn bản của chúng ta sau khi tiền xử lý sẽ được mã hóa thành các vector số để có thể sử dụng trong mô hình học máy. Có nhiều phương pháp để mã hóa văn bản, nhưng trong bài này, chúng ta sẽ sử dụng từng phương pháp trong ba phương pháp sau và so sánh chúng với nhau: *Bag-of-Words* (BoW), *TF-IDF* (Term Frequency-Inverse Document Frequency), và *Sentence Embeddings*.

II.4.1. Bag-of-Words (BoW)

Bag-of-Words (BoW) là một kỹ thuật biểu diễn văn bản đơn giản và thường được sử dụng. Nó chuyển đổi văn bản thành một vector có độ dài cố định bằng cách đếm số lần xuất hiện của mỗi từ trong văn bản. Phương pháp này bỏ qua ngữ pháp và thứ tự từ nhưng vẫn giữ lại tần suất của các từ.
Ví dụ về cách sử dụng BoW để mã hóa văn bản với class `CountVectorizer` từ thư viện `sklearn` như sau:

```python
docs = [
    "I am going to school to study for the final exam.",
    "The weather is nice today and I feel happy.",
    "I love programming in Python and exploring new libraries.",
    "Data science is an exciting field with many opportunities.",
]

bow = CountVectorizer()
vectors = bow.fit_transform(docs)

for i, vec in enumerate(vectors):
    print(f"Document {i+1}: {vec.toarray()}")
```

Đoạn code trên khai báo một danh sách các văn bản (docs) và sử dụng class `CountVectorizer` để mã hóa chúng thành các vector BoW với hàm `fit_transform`. Sau đó, nó in ra các vector tương ứng với từng văn bản như sau:

Vector BoW khi sử dụng CountVectorizer

```
Document 1: [[1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 1 1 2 0 0 0]]
Document 2: [[0 0 1 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 0]]
Document 3: [[0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 1 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0]]
Document 4: [[0 1 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1]]
```

Chúng ta thấy được mỗi văn bản được mã hóa thành một vector với kích thước bằng với số lượng từ trong từ điển. Mỗi phần tử trong vector tương ứng với tần suất xuất hiện của từ đó trong văn bản.

II.4.2. TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) là một phương pháp mã hóa văn bản nâng cao hơn BoW, nó không chỉ tính tần suất của từ trong văn bản mà còn xem xét tần suất

của từ trong toàn bộ tập dữ liệu. Điều này giúp giảm trọng số của các từ phổ biến và tăng trọng số của các từ hiếm gặp, từ đó cải thiện khả năng phân loại. Ví dụ về cách sử dụng TF-IDF để mã hóa văn bản với class `TfidfVectorizer` từ thư viện `sklearn` như sau:

```
docs = [
    "I am going to school to study for the final exam.",
    "The weather is nice today and I feel happy.",
    "I love programming in Python and exploring new libraries.",
    "Data science is an exciting field with many opportunities.",
]

vectorizer = TfidfVectorizer()
tfidf_vectors = vectorizer.fit_transform(docs)

for i, vec in enumerate(tfidf_vectors):
    print(f"TF-IDF for Document {i+1}:")
    print(vec.toarray())
```

Đoạn code trên khai báo một danh sách các văn bản (docs) và sử dụng class `TfidfVectorizer` để mã hóa chúng thành các vector TF-IDF với hàm `fit_transform`. Sau đó, nó in ra các vector tương ứng với từng văn bản như sau:

Vector TF-IDF khi sử dụng TfidfVectorizer

TF-IDF for Document 1:

[[0.29333722 0. 0. 0. 0.29333722 0. 0. 0. 0. 0.29333722 0.29333722 0.29333722 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.29333722 0. 0.29333722 0.23127044 0.58667444 0. 0. 0.]]

TF-IDF for Document 2:

[[0. 0. 0.30091213 0. 0. 0. 0. 0.38166888 0. 0. 0. 0. 0.38166888 0. 0.30091213 0. 0. 0. 0. 0.38166888 0. 0. 0. 0. 0. 0. 0.30091213 0. 0.38166888 0.38166888 0.]]

TF-IDF for Document 3:

[[0. 0. 0.2855815 0. 0. 0. 0.36222393 0. 0. 0. 0. 0. 0. 0.36222393 0. 0.36222393 0.36222393 0. 0.36222393 0. 0. 0.36222393 0.36222393 0. 0. 0. 0. 0. 0. 0. 0.]]

TF-IDF for Document 4:

[[0. 0.34056989 0. 0.34056989 0. 0.34056989 0. 0. 0.34056989 0. 0. 0. 0. 0. 0.26850921 0. 0. 0.34056989 0. 0. 0.34056989 0. 0. 0. 0.34056989 0. 0. 0. 0. 0. 0.34056989]]

Khác với BoW, các vector TF-IDF có trọng số khác nhau cho từng từ, do đó các vector này biểu diễn từng trọng số với kiểu số thực (float) thay vì số nguyên (int). Điều này giúp mô hình phân loại có thể nhận biết được tầm quan trọng của từng từ trong văn bản.

II.4.3. Sentence Embeddings

Sentence Embeddings là một phương pháp mã hóa văn bản nâng cao hơn, nó chuyển đổi toàn bộ câu hoặc đoạn văn thành một vector có kích thước cố định. Các vector này thường được huấn luyện trên các tập dữ liệu lớn và có thể nắm bắt được ngữ nghĩa của câu, từ đó cải thiện khả năng phân loại.
Chúng ta sẽ implement class `EmbeddingVectorizer` để mã hóa văn bản thành các vector embeddings. Class này sẽ sử dụng mô hình pre-trained từ thư viện `sentence-transformers` để chuyển đổi văn bản thành vector.

Xây dựng EmbeddingVectorizer để mã hóa văn bản

```python
class EmbeddingVectorizer:
    def __init__(
        self,
        model_name: str = 'intfloat/multilingual-e5-base',
        normalize: bool = True
    ):
        self.model = SentenceTransformer(model_name)
        self.normalize = normalize

    def _format_inputs(
        self,
        texts: List[str],
        mode: Literal['query', 'passage']
    ) -> List[str]:
        if mode not in {"query", "passage"}:
            raise ValueError("Mode must be either 'query' or 'passage'")
        return [f"{mode}: {text.strip()}" for text in texts]

    def transform(
        self,
        texts: List[str],
        mode: Literal['query', 'passage'] = 'query'
    ) -> List[List[float]]:
        if mode == 'raw':
            inputs = texts
        else:
            inputs = self._format_inputs(texts, mode)

        embeddings = self.model.encode(inputs, normalize_embeddings=self.normalize)
        return embeddings.tolist()

    def transform_numpy(
        self,
        texts: List[str],
        mode: Literal['query', 'passage'] = 'query'
    ) -> np.ndarray:
        return np.array(self.transform(texts, mode=mode))
```

Class `EmbeddingVectorizer` được xây dựng với các methods sau:

- `__init__`: Khởi tạo mô hình SentenceTransformer với tên mô hình và tùy chọn chuẩn hóa.

- `_format_inputs`: Định dạng đầu vào cho mô hình, thêm tiền tố "query"hoặc "passage"vào văn bản.
- `transform`: Chuyển đổi danh sách văn bản thành các vector embeddings.
- `transform_numpy`: Chuyển đổi danh sách văn bản thành các vector embeddings dưới dạng numpy array.

Tương tự hai phương pháp trên, chúng ta sẽ sử dụng class `EmbeddingVectorizer` để mã hóa văn bản trong bộ dữ liệu của mình. Đoạn code sau sẽ thực hiện việc này:

```
docs = [
    "I am going to school to study for the final exam.",
    "The weather is nice today and I feel happy.",
    "I love programming in Python and exploring new libraries.",
    "Data science is an exciting field with many opportunities.",
]

vectorizer = EmbeddingVectorizer()
embeddings = vectorizer.transform(docs)

for i, emb in enumerate(embeddings):
    print(f"Embedding for Document {i+1}:")
    print(emb[:3])  # Print first 10 dimensions for brevity
```

Đoạn code trên sẽ mã hóa các văn bản trong danh sách `docs` thành các vector embeddings và in ra 10 chiều đầu tiên của mỗi vector như sau:

Vector Embeddings khi sử dụng EmbeddingVectorizer

Embedding for Document 1:
-0.014805682934820652, 0.031276583671569824, -0.01615864224731922
Embedding for Document 2:
0.011917386204004288, 0.03327357769012451, -0.025739021599292755
Embedding for Document 3:
0.012662884779274464, 0.03936118632555008, -0.024181174114346504
Embedding for Document 4:
0.006393593735992908 5, 0.04922199621796608, -0.028402965515851974

Các vectors này tương tự như các vectors TF-IDF, nhưng chúng có kích thước cố định và có thể nắm bắt được ngữ nghĩa của câu. Bên cạnh đó, có rất ít từ trong các vectors này có giá trị bằng 0, điều này cho thấy các vectors embeddings có thể biểu diễn tốt hơn ngữ nghĩa của văn bản so với các phương pháp mã hóa khác.

II.4.4. Khai báo và khởi tạo các vectorizers

Chúng ta sẽ khai báo và khởi tạo các vectorizers và huấn luyện chúng trên tập dữ liệu của chúng ta cho từng phương pháp mã hóa văn bản đã đề cập ở trên. Đoạn code sau sẽ thực hiện việc này:

```python
bow_vectorizer = CountVectorizer()
X_train_bow = # Your code here
X_test_bow = bow_vectorizer.transform(X_test)

tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = # Your code here
X_test_tfidf = tfidf_vectorizer.transform(X_test)

embedding_vectorizer = EmbeddingVectorizer()
X_train_embeddings = # Your code here
X_test_embeddings = embedding_vectorizer.transform(X_test)

# convert all to numpy arrays for consistency
X_train_bow, X_test_bow = np.array(X_train_bow.toarray()), np.array(X_test_bow.toarray
                                                    ())
X_train_tfidf, X_test_tfidf = np.array(X_train_tfidf.toarray()), np.array(X_test_tfidf.
                                                    toarray())
X_train_embeddings, X_test_embeddings = np.array(X_train_embeddings), np.array(
                                                    X_test_embeddings)

# Print shapes of the transformed datasets
print(f"Shape of X_train_bow: {X_train_bow.shape}")
print(f"Shape of X_test_bow: {X_test_bow.shape}\n")
print(f"Shape of X_train_tfidf: {X_train_tfidf.shape}")
print(f"Shape of X_test_tfidf: {X_test_tfidf.shape}\n")
print(f"Shape of X_train_embeddings: {X_train_embeddings.shape}")
print(f"Shape of X_test_embeddings: {X_test_embeddings.shape}\n")
```

Kết quả in ra màn hình lần lượt sẽ là kích thước của các tập dữ liệu đã được mã hóa:

Kích thước của các tập dữ liệu đã được mã hóa

Shape of X_train_bow: (800, 10373)
Shape of X_test_bow: (200, 10373)

Shape of X_train_tfidf: (800, 10373)
Shape of X_test_tfidf: (200, 10373)

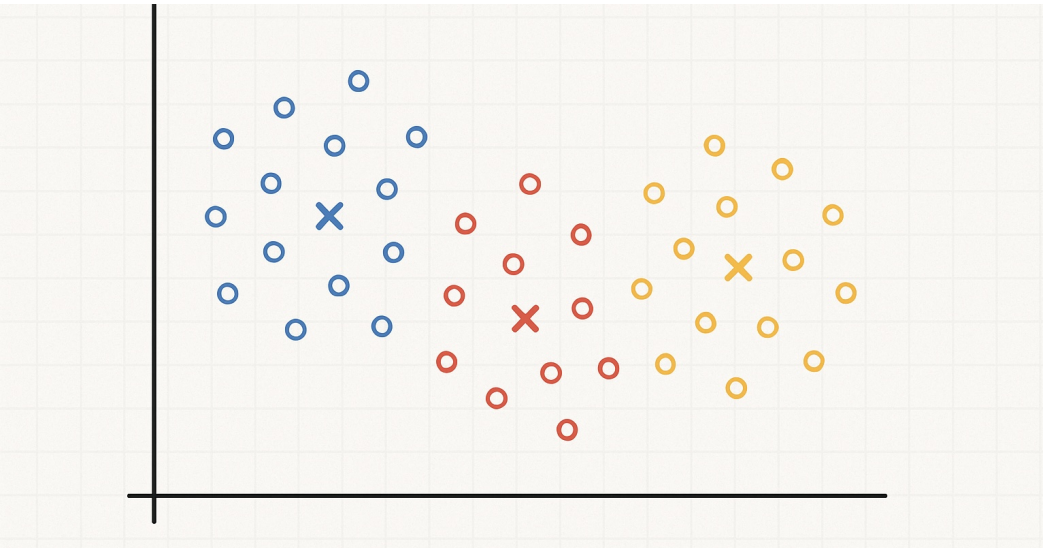
Shape of X_train_embeddings: (800, 768)
Shape of X_test_embeddings: (200, 768)

Chúng ta thấy rằng các vector BoW và TF-IDF có kích thước bằng nhau, tương ứng với số lượng từ trong từ điển (10373 từ), trong khi các vector embeddings có kích thước cố định là 768, cho thấy chúng được mã hóa thành các vector có kích thước nhỏ hơn nhiều so với BoW và TF-IDF.

II.5. Huấn luyện và đánh giá mô hình phân loại

II.5.1. KMeans Clustering

KMean là một thuật toán phân cụm không giám sát, nó sẽ phân chia các vector thành các cụm dựa trên khoảng cách giữa chúng. Chúng ta sẽ sử dụng KMeans để phân cụm các vector BoW, TF-IDF và embeddings đã được mã hóa ở trên.



Hình 3: KMean Clustering Algorithm

Đối với mỗi phương pháp mã hóa, chúng ta sẽ khởi tạo một mô hình KMeans với số lượng cụm (clusters) là 5 và huấn luyện nó trên tập dữ liệu huấn luyện. Để thực hiện việc này, chúng ta sẽ viết hàm `train_and_test_kmeans` để huấn luyện mô hình KMeans và in ra các cụm đã được phân chia như sau:

Hàm `train_and_test_kmeans` để huấn luyện mô hình KMeans

```python
def train_and_test_kmeans(X_train, y_train, X_test, y_test, n_clusters: int):
    kmeans = # Your code here
    cluster_ids = kmeans.fit_predict(X_train)

    # Assign label to clusters
    cluster_to_label = {}
    for cluster_id in set(cluster_ids):
        # Get all labels in this cluster
        labels_in_cluster = [y_train[i] for i in range(len(y_train)) if cluster_ids[i]
                                                == cluster_id]
        most_common_label = Counter(labels_in_cluster).most_common(1)[0][0]
        cluster_to_label[cluster_id] = most_common_label

    # Predict labels for test set
    test_cluster_ids = kmeans.predict(X_test)
    y_pred = [cluster_to_label[cluster_id] for cluster_id in test_cluster_ids]
    accuracy = accuracy_score(y_test, y_pred)
```

```
    report = classification_report(y_test, y_pred, target_names=[id_to_label[i] for i
                                                 in range(len(id_to_label))], output_dict=
                                                 True)

    return y_pred, accuracy, report
```

Hàm `train_and_test_kmeans` sẽ thực hiện các bước sau:

- Khởi tạo mô hình KMeans với số lượng cụm là `n_clusters`.
- Huấn luyện mô hình trên tập dữ liệu huấn luyện và dự đoán các cụm cho tập dữ liệu huấn luyện.
- Gán nhãn cho các cụm dựa trên nhãn phổ biến nhất trong mỗi cụm.
- Dự đoán nhãn cho tập dữ liệu kiểm tra và tính toán độ chính xác (accuracy) cũng như báo cáo phân loại (classification report).

Sau đó, chúng ta chạy hàm `train_and_test_kmeans` cho từng phương pháp mã hóa và in ra kết quả như sau:

Huấn luyện mô hình KMeans cho các phương pháp mã hóa

```
    # Train and test K-Means on different vectorized datasets
km_bow_labels, km_bow_accuracy, km_bow_report = train_and_test_kmeans(X_train_bow,
                                                 y_train, X_test_bow, y_test, n_clusters=len
                                                 (label_to_id))
km_tfidf_labels, km_tfidf_accuracy, km_tfidf_report = train_and_test_kmeans(
                                                 X_train_tfidf, y_train, X_test_tfidf,
                                                 y_test, n_clusters=len(label_to_id))
km_embeddings_labels, km_embeddings_accuracy, km_embeddings_report =
                                                 train_and_test_kmeans(X_train_embeddings,
                                                 y_train, X_test_embeddings, y_test,
                                                 n_clusters=len(label_to_id))

# Print K-Means results
print("Accuracies for K-Means:")
print(f"Bag of Words: {km_bow_accuracy:.4f}")
print(f"TF-IDF: {km_tfidf_accuracy:.4f}")
print(f"Embeddings: {km_embeddings_accuracy:.4f}")
```

Kết quả in ra sẽ là độ chính xác của mô hình KMeans cho từng phương pháp mã hóa:

Độ chính xác của mô hình KMeans cho từng phương pháp mã hóa

Accuracies for K-Means:
Bag of Words: 0.5600
TF-IDF: 0.6150
Embeddings: 0.8400

Trong đó, mô hình KMeans với phương pháp mã hóa embeddings đạt độ chính xác cao nhất là 84%, trong khi phương pháp BoW và TF-IDF chỉ đạt độ chính xác lần lượt là 56% và 61.5%. Điều này chứng minh rằng phương pháp mã hóa embeddings có thể nắm bắt được ngữ nghĩa của văn bản tốt hơn so với các phương pháp mã hóa khác mà tiêu tốn ít tài nguyên hơn (vector embeddings có kích thước nhỏ hơn nhiều so với BoW và TF-IDF). Bên cạnh đó, việc tích hợp thông tin tần suất của từ để đánh giá độ quan trọng của từ trong văn bản cũng giúp cải thiện độ chính xác của mô hình (accuracy của mô hình KMeans với phương pháp mã hóa TF-IDF cao hơn so với BoW).
Cuối cùng, chúng ta vẽ confusion matrix cho từng phương pháp mã hóa để trực quan hóa kết quả phân cụm với hàm `plot_confusion_matrix` như sau:

Vẽ confusion matrix cho từng phương pháp mã hóa với hàm `plot_confusion_matrix`

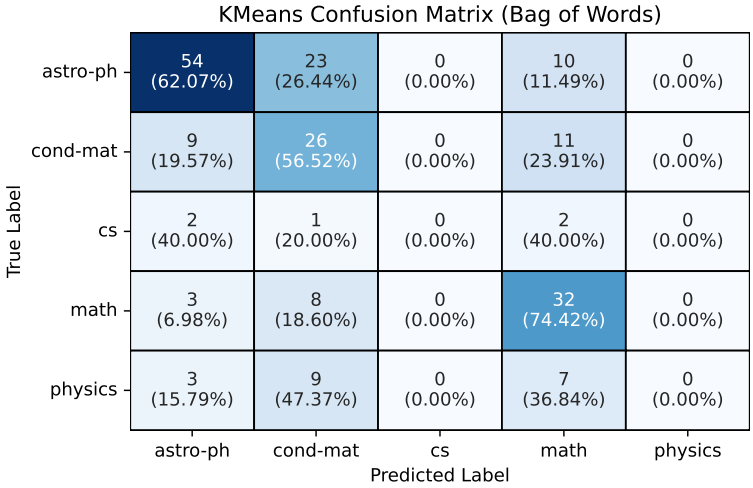
```
def plot_confusion_matrix(y_true, y_pred, label_list, figure_name="Confusion Matrix",
                                                    save_path=None):
    """
    Plots a confusion matrix with raw counts and normalized values using Seaborn.

    Parameters:
        y_true (array-like): True class labels.
        y_pred (array-like): Predicted class labels.
        label_list (list or dict): Class names (list or dict from ID to name).
        figure_name (str): Title of the plot.
        save_path (str, optional): Path to save the figure. If None, the figure will
                                                    not be saved.
    """
    # Compute confusion matrix and normalize
    cm = confusion_matrix(y_true, y_pred)
    cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    # Map class indices to names
    labels = np.unique(y_true)
    if isinstance(label_list, dict):
        class_names = [label_list[i] for i in labels]
    else:
        class_names = [label_list[i] for i in labels]

    # Create annotations with raw + normalized values
    annotations = np.empty_like(cm).astype(str)
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            raw = cm[i, j]
            norm = cm_normalized[i, j]
            annotations[i, j] = f"{raw}\n({norm:.2%})"

    # Plot
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=annotations, fmt="", cmap="Blues",
                xticklabels=class_names, yticklabels=class_names,
                cbar=False, linewidths=1, linecolor='black')

    plt.xlabel('Predicted Label')
```

```
    plt.ylabel('True Label')
    plt.title(figure_name)
    plt.tight_layout()

    if save_path:
        plt.savefig(save_path)

    plt.show()
```

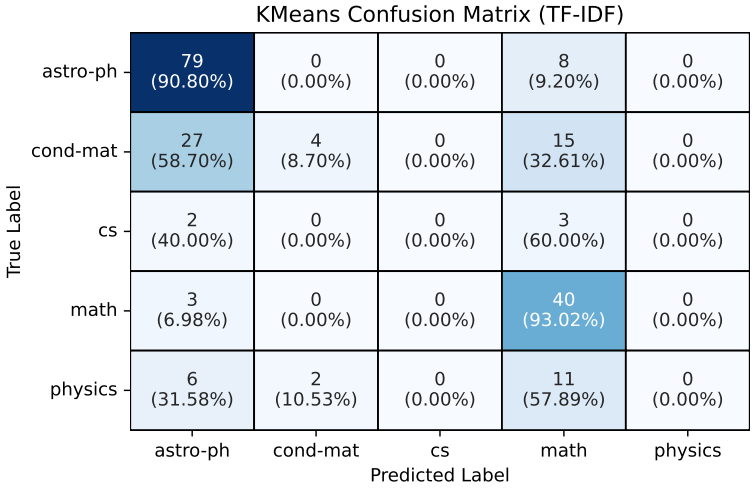
Và sử dụng hàm này để vẽ confusion matrix cho từng phương pháp mã hóa:

```
# Draw confusion matrices
plot_confusion_matrix(y_test, km_bow_labels, sorted_labels, "KMeans Confusion Matrix (Bag of Words)")
plot_confusion_matrix(y_test, km_tfidf_labels, sorted_labels, "KMeans Confusion Matrix (TF-IDF)")
plot_confusion_matrix(y_test, km_embeddings_labels, sorted_labels, "KMeans Confusion Matrix (Embeddings)")
```

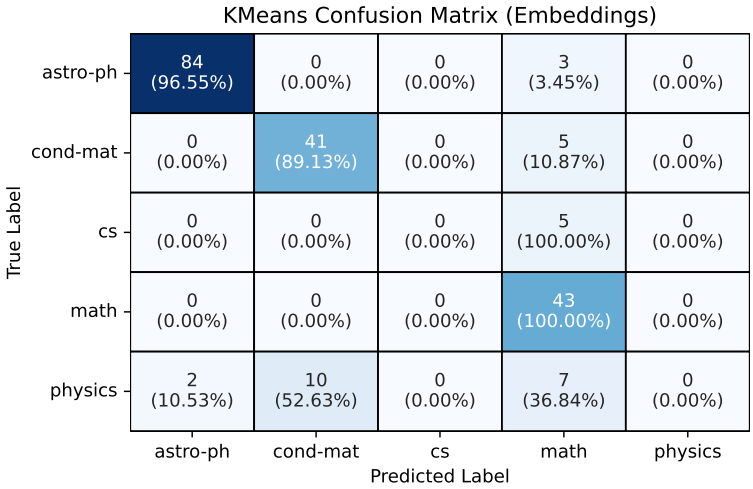
Hình dưới đây là các confusion matrix cho từng phương pháp mã hóa:



(a) Confusion Matrix KMeans với BoW



(b) Confusion Matrix KMeans với TF-IDF



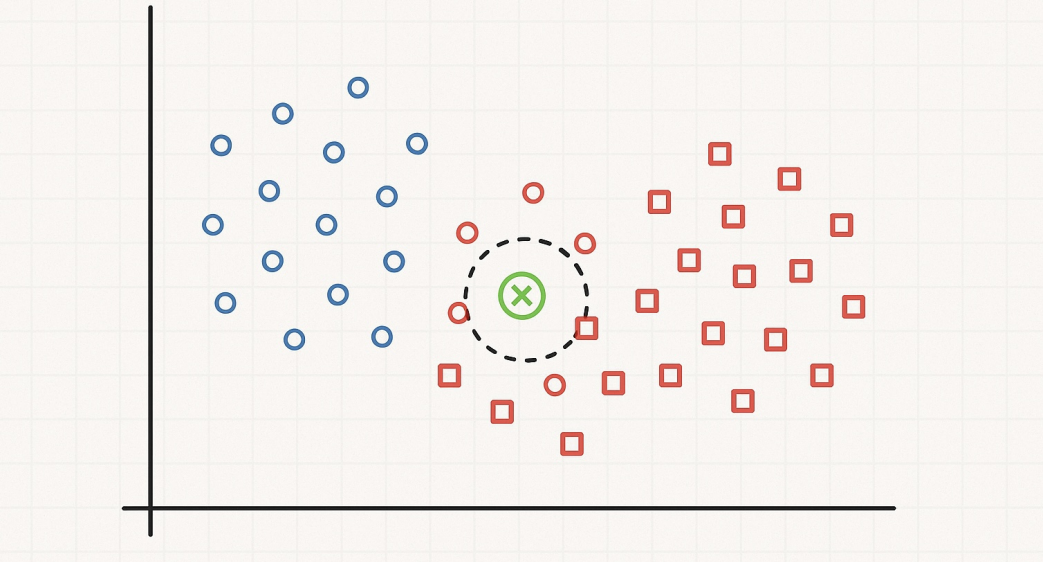
(c) Confusion Matrix KMeans với Embeddings

Hình 4: Confusion Matrix cho các phương pháp mã hóa với KMeans

Chúng ta có thể thấy class astro-ph và math luôn được dự đoán đúng nhiều nhất. Trong khi class cs và physics có số lượng dự đoán sai nhiều nhất. Điều này có thể do các class này có nội dung tương tự nhau, hoặc do số lượng mẫu trong các class này không đủ lớn để mô hình học được các đặc trưng phân biệt.

II.5.2. K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) là một thuật toán phân loại đơn giản và hiệu quả, nó phân loại một điểm dữ liệu dựa trên lớp đa số của k láng giềng gần nhất trong không gian đặc trưng. Chúng ta sẽ sử dụng KNN để phân loại các vector BoW, TF-IDF và embeddings đã được mã hóa ở trên.



Hình 5: K-Nearest-Neighbor Algorithm

Đầu tiên, chúng ta sẽ thiết kế một hàm `train_and_test_knn` để huấn luyện mô hình KNN và in ra các kết quả phân loại như sau:

Hàm `train_and_test_knn` để huấn luyện mô hình KNN

```python
def train_and_test_knn(X_train, y_train, X_test, y_test, n_neighbors: int = 5):
    knn = # Your code here
    knn.fit(X_train, y_train)

    # Predict on the test set
    y_pred = # Your code here

    # Calculate accuracy and classification report
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, target_names=sorted_labels,
                                   output_dict=True)

    return y_pred, accuracy, report
```

Hàm `train_and_test_knn` sẽ thực hiện các bước sau:

- Khởi tạo mô hình KNN với số lượng láng giềng là `n_neighbors`.
- Huấn luyện mô hình trên tập dữ liệu huấn luyện.
- Dự đoán nhãn cho tập dữ liệu kiểm tra và tính toán độ chính xác (accuracy) cũng như báo cáo phân loại (classification report).
- Trả về các nhãn dự đoán, độ chính xác và báo cáo phân loại.

Sau đó, chúng ta sẽ chạy hàm `train_and_test_knn` cho từng phương pháp mã hóa và in ra kết quả như sau:

Huấn luyện mô hình KNN cho các phương pháp mã hóa

```
knn_bow_labels, knn_bow_accuracy, knn_bow_report = train_and_test_knn(X_train_bow,
                                                    y_train, X_test_bow, y_test)
knn_tfidf_labels, knn_tfidf_accuracy, knn_tfidf_report = train_and_test_knn(
                                                    X_train_tfidf, y_train, X_test_tfidf,
                                                    y_test)
knn_embeddings_labels, knn_embeddings_accuracy, knn_embeddings_report =
                                                    train_and_test_knn(X_train_embeddings,
                                                    y_train, X_test_embeddings, y_test)

# Print KNN results
print("Accuracies for KNN:")
print(f"Bag of Words: {knn_bow_accuracy:.4f}")
print(f"TF-IDF: {knn_tfidf_accuracy:.4f}")
print(f"Embeddings: {knn_embeddings_accuracy:.4f}")
```

Kết quả in ra sẽ là độ chính xác của mô hình KNN cho từng phương pháp mã hóa:

Độ chính xác của mô hình KNN cho từng phương pháp mã hóa

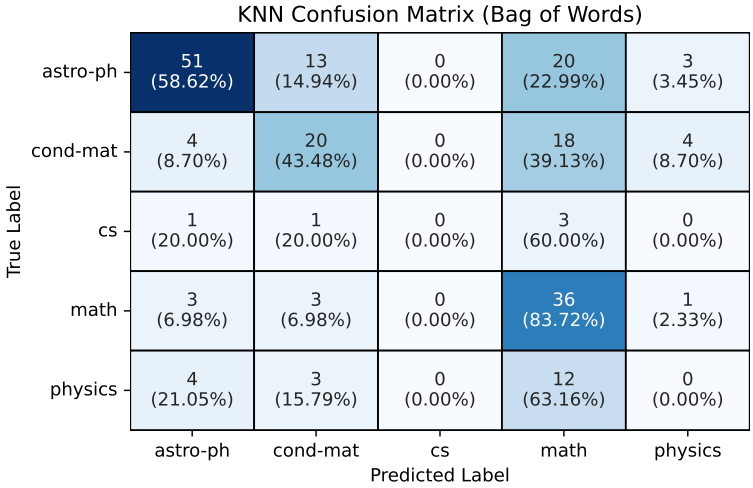
```
Accuracies for KNN:
Bag of Words: 0.5350
TF-IDF: 0.8150
Embeddings: 0.8900
```

Trong đó, mô hình KNN với phương pháp mã hóa embeddings đạt độ chính xác cao nhất là 89%, trong khi phương pháp BoW chỉ đạt độ chính xác là 53.5%. Điểm chung khi so sánh với KMeans là mô hình KNN với phương pháp mã hóa TF-IDF đạt độ chính xác cao hơn so với BoW, cho thấy việc sử dụng tần suất từ để đánh giá độ quan trọng của từ trong văn bản có thể cải thiện độ chính xác của mô hình. Mô hình KNN với phương pháp mã hóa embeddings cũng đạt độ chính xác cao nhất, vượt trội hơn cả KMeans với cùng phương pháp mã hóa (84% của KMeans so với 89% của KNN).
Cuối cùng, chúng ta vẽ confusion matrix cho từng phương pháp mã hóa để trực quan hóa kết quả phân loại với hàm `plot_confusion_matrix` như sau:

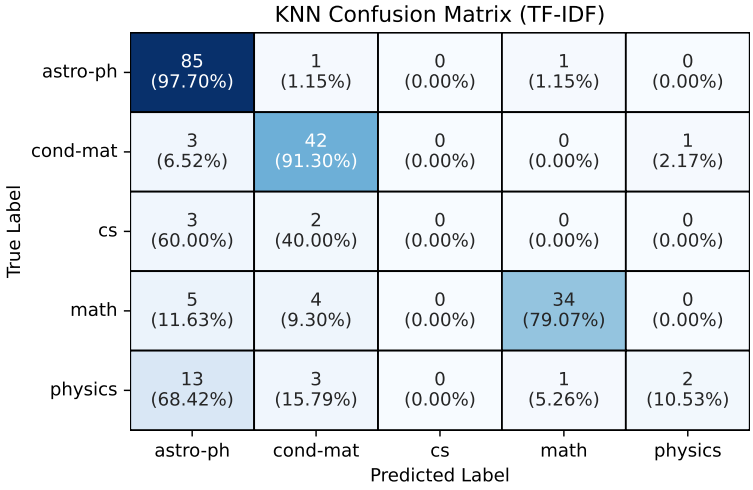
Vẽ confusion matrix cho từng phương pháp mã hóa với hàm `plot_confusion_matrix`

```
# Draw confusion matrices
plot_confusion_matrix(y_test, knn_bow_labels, sorted_labels, "KNN Confusion Matrix (Bag of Words)", "pdf/Figures/knn_bow_confusion_matrix.pdf")
plot_confusion_matrix(y_test, knn_tfidf_labels, sorted_labels, "KNN Confusion Matrix (TF-IDF)", "pdf/Figures/knn_tfidf_confusion_matrix.pdf")
plot_confusion_matrix(y_test, knn_embeddings_labels, sorted_labels, "KNN Confusion Matrix (Embeddings)", "pdf/Figures/knn_embeddings_confusion_matrix.pdf")
```

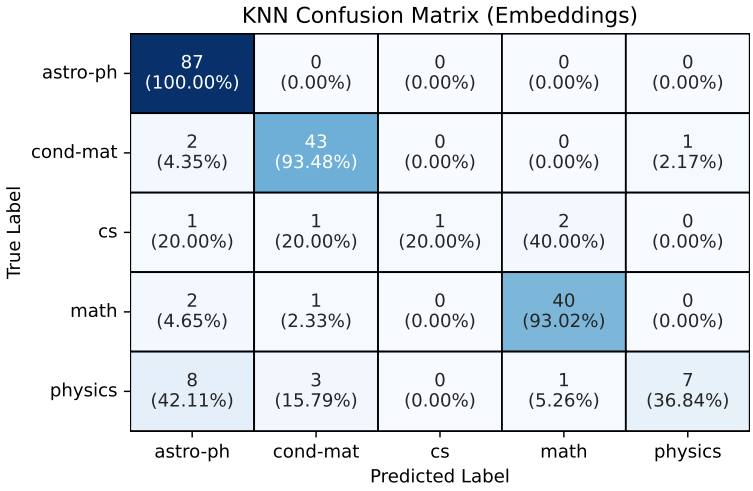
Confusion matrix cho từng phương pháp mã hóa kết hợp với KNN được hiển thị trong Hình 6. Chúng ta có thể thấy rằng mô hình KNN với phương pháp mã hóa embeddings đạt được độ chính xác cao nhất, với hầu hết các dự đoán đều đúng. Trong khi đó, mô hình KNN với phương pháp mã hóa BoW có nhiều dự đoán sai hơn, đặc biệt là đối với các class như astro-ph và cs.



(a) Confusion Matrix KNN với BoW



(b) Confusion Matrix KNN với TF-IDF

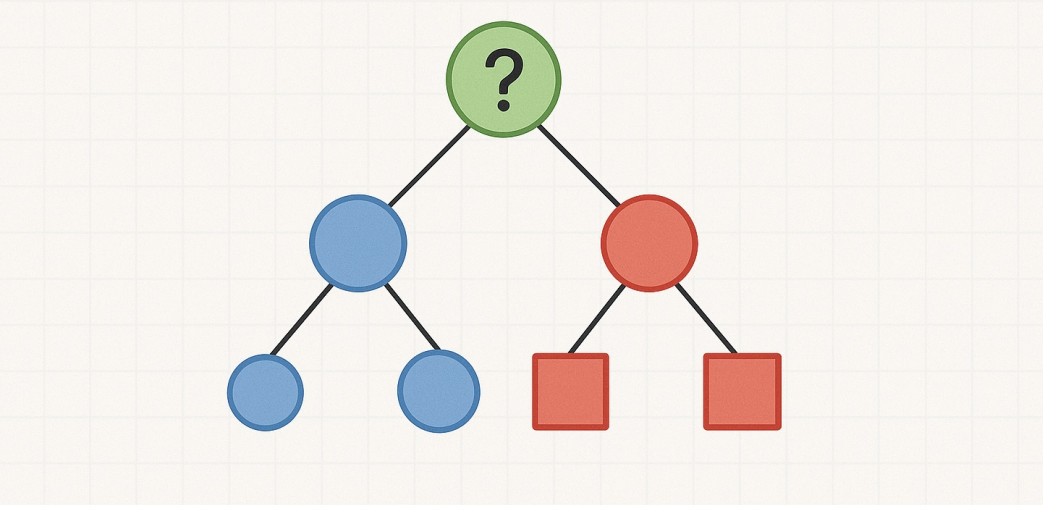


(c) Confusion Matrix KNN với Embeddings

Hình 6: Confusion Matrix cho từng phương pháp mã hóa với KNN

II.5.3. Decision Tree

Decision Tree là một thuật toán phân loại dựa trên cấu trúc cây, nó phân chia dữ liệu thành các nhánh dựa trên các đặc trưng của dữ liệu để đưa ra quyết định cuối cùng. Chúng ta sẽ sử dụng Decision Tree để phân loại các vector BoW, TF-IDF và embeddings đã được mã hóa ở trên.



Hình 7: Decision Tree Algorithm

Đầu tiên, chúng ta sẽ thiết kế một hàm `train_and_test_decision_tree` để huấn luyện mô hình Decision Tree và in ra các kết quả phân loại như sau:

Hàm `train_and_test_decision_tree` để huấn luyện mô hình Decision Tree

```python
def train_and_test_decision_tree(X_train, y_train, X_test, y_test):
    dt = # Your code here
    dt.fit(X_train, y_train)

    # Predict on the test set
    y_pred = # Your code here

    # Calculate accuracy and classification report
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, target_names=sorted_labels,
                                   output_dict=True)

    return y_pred, accuracy, report
```

Hàm `train_and_test_decision_tree` sẽ thực hiện các bước sau:

- Khởi tạo mô hình Decision Tree với một seed cố định để đảm bảo tính tái lập.
- Huấn luyện mô hình trên tập dữ liệu huấn luyện.
- Dự đoán nhãn cho tập dữ liệu kiểm tra và tính toán độ chính xác (accuracy) cũng như báo cáo phân loại (classification report).

- Trả về các nhãn dự đoán, độ chính xác và báo cáo phân loại.

Sau đó, chúng ta sẽ chạy hàm `train_and_test_decision_tree` cho từng phương pháp mã hóa và in ra kết quả như sau:

Huấn luyện mô hình Decision Tree cho các phương pháp mã hóa

```
dt_bow_labels, dt_bow_accuracy, dt_bow_report = train_and_test_decision_tree(
                                                X_train_bow, y_train, X_test_bow, y_test)
dt_tfidf_labels, dt_tfidf_accuracy, dt_tfidf_report = train_and_test_decision_tree(
                                                X_train_tfidf, y_train, X_test_tfidf,
                                                y_test)
dt_embeddings_labels, dt_embeddings_accuracy, dt_embeddings_report =
                                                train_and_test_decision_tree(
                                                X_train_embeddings, y_train,
                                                X_test_embeddings, y_test)

# Print Decision Tree results
print("Accuracies for Decision Tree:")
print(f"Bag of Words: {dt_bow_accuracy:.4f}")
print(f"TF-IDF: {dt_tfidf_accuracy:.4f}")
print(f"Embeddings: {dt_embeddings_accuracy:.4f}")
```

Kết quả in ra sẽ là độ chính xác của mô hình Decision Tree cho từng phương pháp mã hóa:

Độ chính xác của mô hình Decision Tree cho từng phương pháp mã hóa

Accuracies for Decision Tree:
Bag of Words: 0.6350
TF-IDF: 0.5950
Embeddings: 0.7150

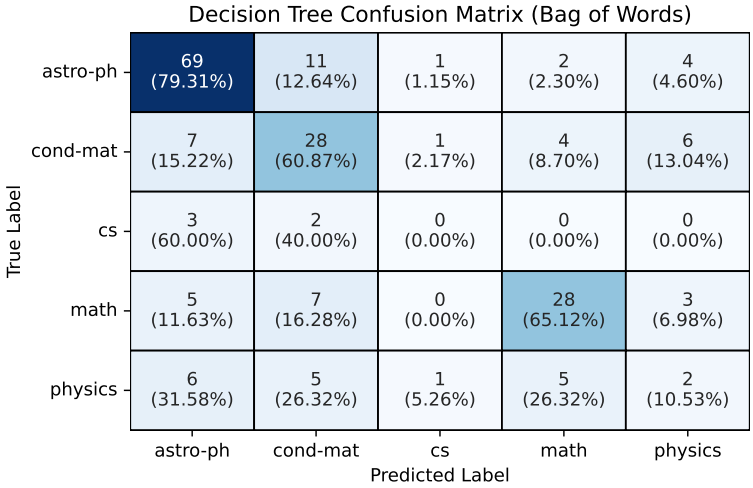
Trái ngược với KMeans và KNN, mô hình Decision Tree với phương pháp mã hóa BoW đạt độ chính xác cao hơn so với TF-IDF (63.5% so với 59.5%). Điều này có thể do mô hình Decision Tree có khả năng phân chia dữ liệu tốt hơn khi sử dụng các đặc trưng tần suất từ trong BoW. Tuy nhiên, mô hình Decision Tree với phương pháp mã hóa embeddings vẫn đạt độ chính xác cao nhất là 71.5%, cho thấy việc sử dụng embeddings có thể cải thiện độ chính xác của mô hình. Cuối cùng, chúng ta vẽ confusion matrix cho từng phương pháp mã hóa để trực quan hóa kết quả phân loại với hàm `plot_confusion_matrix` như sau:

Vẽ confusion matrix cho từng phương pháp mã hóa với hàm `plot_confusion_matrix`

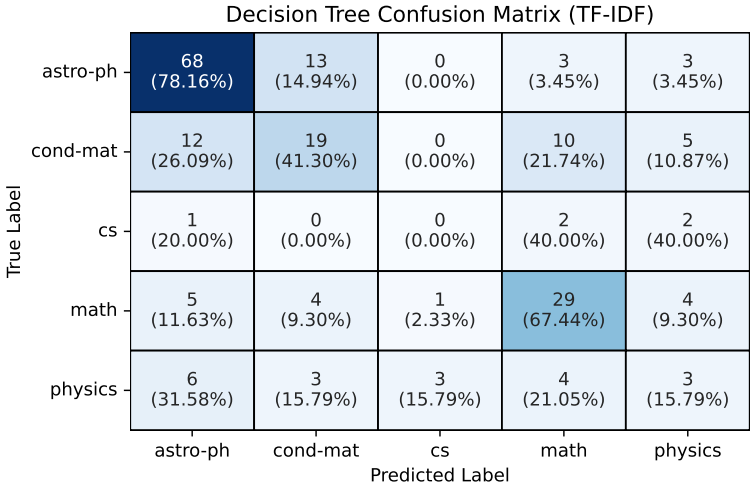
```
# Draw confusion matrices
plot_confusion_matrix(y_test, dt_bow_labels, sorted_labels, "Decision Tree Confusion
                                                Matrix (Bag of Words)", "pdf/Figures/
                                                dt_bow_confusion_matrix.pdf")
plot_confusion_matrix(y_test, dt_tfidf_labels, sorted_labels, "Decision Tree Confusion
                                                Matrix (TF-IDF)", "pdf/Figures/
                                                dt_tfidf_confusion_matrix.pdf")
plot_confusion_matrix(y_test, dt_embeddings_labels, sorted_labels, "Decision Tree
```

```
                                    Confusion Matrix (Embeddings)", "pdf/
                                    Figures/dt_embeddings_confusion_matrix.pdf"
                                    )
```

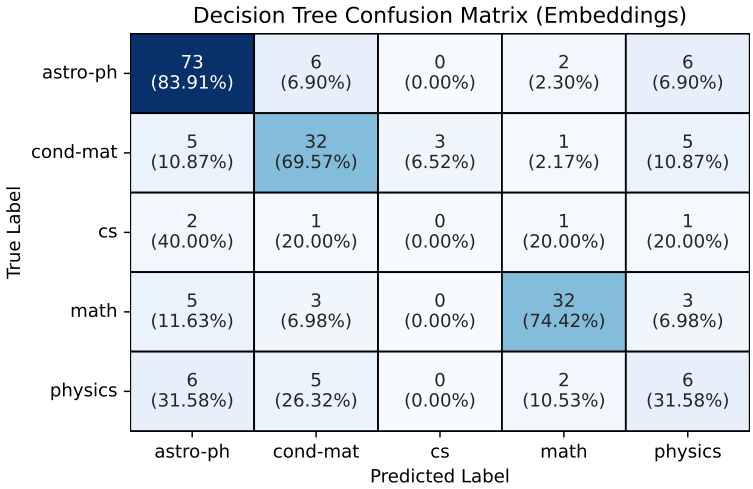
Hình dưới đây là các confusion matrix cho từng phương pháp mã hóa với mô hình Decision Tree:



(a) Confusion Matrix Decision Tree với BoW



(b) Confusion Matrix Decision Tree với TF-IDF



(c) Confusion Matrix Decision Tree với Embeddings

Hình 8: Confusion Matrix cho từng phương pháp mã hóa với mô hình Decision Tree

III. Câu hỏi trắc nghiệm

1. Trong bài toán phân loại abstract bài báo khoa học, mục tiêu chính của mô hình là gì?
   (*a*) Dịch abstract sang tiếng Việt
   (*b*) Tóm tắt nội dung abstract
   (*c*) Dự đoán chủ đề (topic) của abstract
   (*d*) Phát hiện đạo văn trong abstract

2. Đâu là ba phương pháp mã hóa văn bản được sử dụng trong project?
   (*a*) BoW, SVM, CNN
   (*b*) TF-IDF, BoW, Sentence Embeddings
   (*c*) One-hot Encoding, PCA, LSTM
   (*d*) TF-IDF, Word2Vec, LDA

3. Trong project, mô hình KMeans đạt độ chính xác cao nhất khi dùng với phương pháp mã hóa nào?
   (*a*) TF-IDF
   (*b*) Bag-of-Words
   (*c*) Sentence Embeddings
   (*d*) One-hot Encoding

4. Phương pháp mã hóa nào giúp giảm ảnh hưởng của các từ phổ biến và tăng trọng số của các từ hiếm?
   (*a*) BoW
   (*b*) Embeddings
   (*c*) TF-IDF
   (*d*) PCA

5. Số lượng mẫu được sử dụng để huấn luyện và đánh giá mô hình là bao nhiêu?
   (*a*) 500 mẫu
   (*b*) 1000 mẫu
   (*c*) 2000 mẫu
   (*d*) 5000 mẫu

6. Trong các mô hình được sử dụng, mô hình nào là mô hình không giám sát?
   (*a*) Decision Tree
   (*b*) KMeans
   (*c*) KNN
   (*d*) Logistic Regression

7. Câu nào đúng về kích thước vector được sinh ra bởi Sentence Embeddings?
   (*a*) Luôn bằng số lượng từ trong từ điển
   (*b*) Thay đổi theo độ dài văn bản
   (*c*) Cố định, thường là 768 chiều
   (*d*) Luôn là số nguyên dương

8. Label (nhãn) cho mỗi abstract được lấy từ đâu?
   (*a*) Tiêu đề bài báo
   (*b*) Trường "submitter"
   (*c*) Trường "categories", lấy phần primary
   (*d*) Trường "authors_parsed"

9. Khi sử dụng KNN, phương pháp mã hóa nào đạt độ chính xác cao nhất?
   (*a*) Bag-of-Words
   (*b*) TF-IDF
   (*c*) Sentence Embeddings
   (*d*) Không có sự khác biệt

10. Trong pipeline, bước tiền xử lý văn bản **KHÔNG** bao gồm thao tác nào sau đây?
    (*a*) Chuyển về chữ thường
    (*b*) Xoá ký tự đặc biệt
    (*c*) Thêm tiêu đề bài báo vào abstract
    (*d*) Xoá số và dấu câu

Phụ lục

1. **Hint:** Các file code gợi ý và dữ liệu được lưu trong thư mục có thể được tải tại đây.

2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại đây (Lưu ý: Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

3. **Kiến thức cần cho project:**

   Để hoàn thành và hiểu rõ phần project một cách hiệu quả nhất, các học viên cần nắm rõ các kiến thức được trang bị

4. **Đề xuất phương pháp cải tiến project:**

   Project tiếp cận bài toán phân tích chủ đề văn bản (topic modeling) theo hướng sử dụng các phương pháp biểu diễn vector như Bag of Words (BoW), TF-IDF, S-BERT kết hợp với các thuật toán KNN, K-Means clustering và Decision Tree. Để tiếp tục cải tiến, tối ưu hệ thống; nhóm tác biên soạn gợi ý một số phương pháp như sau:

   - **Tiền xử lý nâng cao:** Sử dụng các kỹ thuật tiền xử lý chuyên sâu hơn như loại bỏ từ ngữ ít xuất hiện (rare words), lemmatization thay vì stemming, phát hiện và xử lý stopwords theo ngữ cảnh, và phát hiện cụm từ (phrase detection) để tăng chất lượng vector biểu diễn.
   - **Thử nghiệm nhiều phương pháp biểu diễn:**
     - So sánh hiệu quả giữa BoW, TF-IDF và mô hình embedding hiện đại như S-BERT.
     - Kết hợp nhiều phương pháp biểu diễn (feature fusion) để tận dụng ưu điểm của từng phương pháp.
   - **Tối ưu thuật toán phân cụm và phân loại:**
     - Với K-Means: Sử dụng phương pháp tìm k tối ưu (Elbow Method, Silhouette Score).
     - Với KNN: Tối ưu số láng giềng (k) và sử dụng weighted KNN để cải thiện độ chính xác.
     - Với Decision Tree: Áp dụng kỹ thuật pruning (cắt tỉa) để giảm overfitting và cải thiện khả năng tổng quát hóa.
   - **Topic Modeling nâng cao:**
     - Áp dụng clustering để nhóm tài liệu trước khi xây dựng Decision Tree nhằm tăng độ rõ ràng của các chủ đề.
     - Sử dụng hierarchical clustering để khám phá các chủ đề con (sub-topics).
   - **Đánh giá đa chỉ số:** Áp dụng thêm các chỉ số như Silhouette Score, Davies–Bouldin Index cho clustering, và F1-score, ROC-AUC cho phân loại để đánh giá mô hình toàn diện.
   - **Tăng cường dữ liệu (Data Augmentation):** Áp dụng các phương pháp như thay thế từ đồng nghĩa, paraphrasing hoặc back-translation để làm phong phú tập dữ liệu huấn luyện.

- **Kết hợp tri thức tìm kiếm và biểu diễn:** Thử nghiệm việc kết hợp kết quả từ vector database (FAISS hoặc Annoy) với các thuật toán clustering để hỗ trợ cho việc gán nhãn chủ đề và tăng độ chính xác trong việc phân loại.

Trên đây là một số gợi ý, giúp học viên có thể cải tiến thêm hệ thống để đạt hiểu quả tốt hơn về cả tốc độ xử lý và độ chính xác.

5. Rubric:

Mục	Kiến Thức	Đánh Giá
II.	- Lý thuyết về bài toán Topic Modeling trong Natural Language Processing. - Lý thuyết về các phương pháp biểu diễn văn bản: Bag of Words (BoW), TF-IDF, S-BERT và các thuật toán K-Means, KNN, Decision Tree.	- Hiểu được khái niệm topic modeling và các phương pháp biểu diễn văn bản. - Có khả năng sử dụng Python để triển khai các thuật toán K-Means, KNN và Decision Tree cho bài toán topic modeling.
III.	- Ứng dụng K-Means clustering để phân cụm chủ đề và Decision Tree để gán nhãn chủ đề. - So sánh hiệu quả các phương pháp biểu diễn văn bản (BoW, TF-IDF, S-BERT) trong bài toán topic modeling.	- Thực hiện phân cụm chủ đề và đánh giá mô hình bằng các chỉ số như Silhouette Score. - Đánh giá, so sánh và chọn phương pháp biểu diễn văn bản phù hợp cho dữ liệu thực tế.