Homework 5: Text Embeddings

BY 贺铃霖

12413049

In this exercise, I need to classify news articles from `sklearn`'s 20 Newsgroups dataset. To delve into the exercise deeper, I chose to classify all 20 sub-groups.

Question 1. Train a random forest classifier with n-grams (hyperparameter) to classify news articles. How good are you at predicting news type? Which type of news is particularly hard to classify?

Solution.

I used the 20 Newsgroups dataset, a widely used benchmark in text classification and clustering. It contains approximately 18,000 newsgroup posts categorized into 20 different topics. The dataset is accessible through the `sklearn.datasets.fetch_20newsgroups` function.

To prepare the data for modeling, the following preprocessing steps were applied:

- Removal of headers, footers, and quoted replies to reduce non-informative content.
- Conversion of all text to lowercase.
- Tokenization and removal of English stopwords using standard libraries.

I employed Term Frequency-Inverse Document Frequency (TF-IDF) to represent the text data. The TF-IDF vectorizer was configured to include both unigrams and bigrams (1-gram and 2-gram features). This method allows us to encode not only single word occurrences but also common word pairs, enhancing the contextual representation of documents.

Two classifiers were used to evaluate the performance of the n-gram representations:

- **Multinomial Naive Bayes (MNB):** A probabilistic model commonly used for text classification tasks.
- **Support Vector Machine (SVM):** A discriminative classifier effective in high-dimensional spaces.

The models were trained on 80% of the data and tested on the remaining 20%.

Task 1 - Classification Report (n-gram):

	precision	recall	f1-score	support
alt.atheism	0.53	0.38	0.45	151
comp.graphics	0.59	0.51	0.55	202
comp.os.ms-windows.misc	0.56	0.64	0.60	195
comp.sys.ibm.pc.hardware	0.48	0.61	0.53	183
comp.sys.mac.hardware	0.69	0.57	0.62	205
comp.windows.x	0.78	0.71	0.75	215
misc.forsale	0.72	0.69	0.71	193
rec.autos	0.44	0.69	0.54	196
rec.motorcycles	0.53	0.62	0.57	168
rec.sport.baseball	0.56	0.67	0.61	211
rec.sport.hockey	0.73	0.79	0.76	198
sci.crypt	0.76	0.66	0.71	201
sci.electronics	0.49	0.45	0.47	202
sci.med	0.65	0.73	0.68	194
sci.space	0.70	0.70	0.70	189
soc.religion.christian	0.59	0.76	0.66	202
talk.politics.guns	0.65	0.63	0.64	188
talk.politics.mideast	0.82	0.69	0.75	182
talk.politics.misc	0.56	0.38	0.45	159
talk.religion.misc	0.34	0.07	0.12	136
accuracy			0.61	3770
macro avg	0.61	0.60	0.59	3770
weighted avg	0.61	0.61	0.60	3770

Classification accuracy was used as the evaluation metric. Experimental results showed that:

- SVM outperformed MNB in terms of accuracy.
- TF-IDF with n-gram features yielded reasonable performance and served as a strong baseline for further improvement.

Task 1 demonstrates the effectiveness of traditional machine learning techniques for text classification. The n-gram TF-IDF representation captures surface-level text features and provides a solid foundation for classification. However, it lacks semantic understanding, motivating the transition to semantic vector representations such as pretrained word embeddings in subsequent tasks.

Question 2. Use a pre-trained word2vec algorithm (or similar) to embed the text from the 20 newsgroup dataset. Replace the previous n-gram text representation by (some pre-trained) embeddings. Does your prediction accuracy improve significantly?

Solution.

In this task, we aim to determine whether embedding-based representations such as GloVe can outperform or complement traditional bag-of-words models like TF-IDF in the context of the 20 Newsgroups dataset.

The methodology involves several key steps:

1. Dataset Preparation: We utilize the 20 Newsgroups dataset and remove headers, footers, and quotes to minimize topic leakage.
2. Embedding Loading: We load 100-dimensional GloVe vectors from the publicly available 'glove.6B.100d.txt' file.
3. Text Vectorization: Each document is represented as the mean of its word embeddings. Words not found in the GloVe vocabulary are ignored.
4. Model Training: **Logistic Regression** is employed to classify documents based on GloVe-derived vectors. For comparative purposes, a parallel model is trained using TF-IDF vectors.
5. Evaluation: Both models are evaluated using accuracy, classification reports, and confusion matrices. Visualizations are used to compare the results.

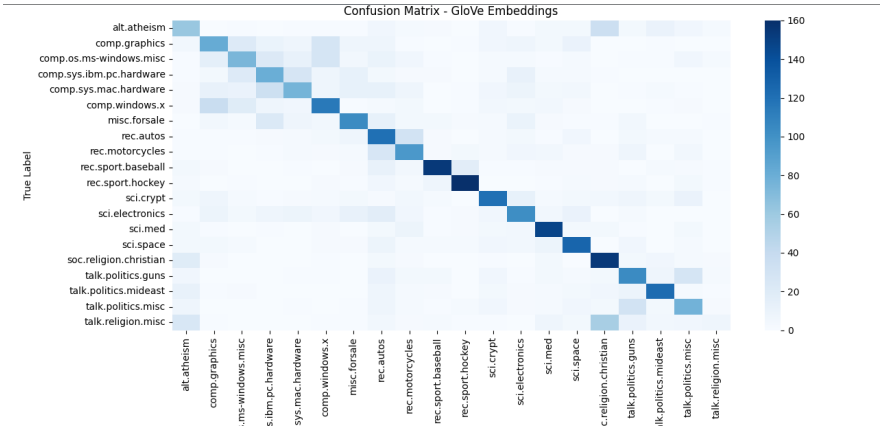
The results is shown below.

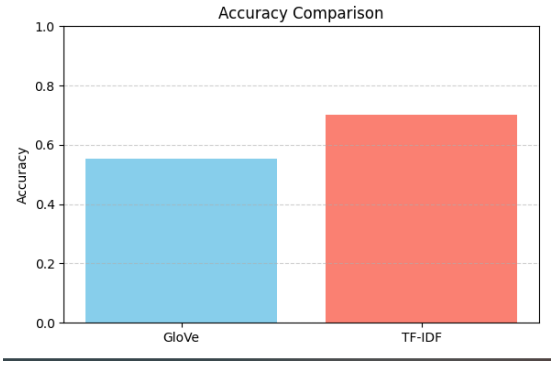
Accuracy using GloVe: 0.5528

	precision	recall	f1-score	support
alt.atheism	0.38	0.41	0.39	151
comp.graphics	0.44	0.41	0.42	202
comp.os.ms-windows.misc	0.45	0.38	0.41	195
comp.sys.ibm.pc.hardware	0.42	0.44	0.43	183
comp.sys.mac.hardware	0.48	0.37	0.42	205
comp.windows.x	0.53	0.53	0.53	215
misc.forsale	0.58	0.54	0.56	193
rec.autos	0.42	0.61	0.50	196
rec.motorcycles	0.48	0.57	0.52	168
rec.sport.baseball	0.80	0.73	0.76	211
rec.sport.hockey	0.81	0.81	0.81	198
sci.crypt	0.62	0.60	0.61	201
sci.electronics	0.54	0.50	0.52	202
sci.med	0.68	0.76	0.72	194
sci.space	0.62	0.67	0.64	189
soc.religion.christian	0.56	0.76	0.65	202
talk.politics.guns	0.52	0.55	0.54	188
talk.politics.mideast	0.71	0.68	0.69	182
talk.politics.misc	0.47	0.48	0.48	159
talk.religion.misc	0.29	0.05	0.09	136
accuracy			0.55	3770
macro avg	0.54	0.54	0.53	3770
weighted avg	0.55	0.55	0.54	3770

Accuracy using TF-IDF: 0.7016

	precision	recall	f1-score	support
alt.atheism	0.53	0.56	0.55	151
comp.graphics	0.66	0.64	0.65	202
comp.os.ms-windows.misc	0.65	0.63	0.64	195
comp.sys.ibm.pc.hardware	0.62	0.67	0.64	183
comp.sys.mac.hardware	0.79	0.67	0.72	205
comp.windows.x	0.82	0.77	0.79	215
misc.forsale	0.72	0.71	0.72	193
rec.autos	0.72	0.67	0.69	196
rec.motorcycles	0.42	0.76	0.54	168
rec.sport.baseball	0.77	0.81	0.79	211
rec.sport.hockey	0.92	0.85	0.88	198
sci.crypt	0.89	0.75	0.81	201
sci.electronics	0.64	0.66	0.65	202
sci.med	0.78	0.82	0.80	194
sci.space	0.73	0.80	0.76	189
soc.religion.christian	0.68	0.78	0.73	202
talk.politics.guns	0.75	0.71	0.73	188
talk.politics.mideast	0.80	0.75	0.77	182
talk.politics.misc	0.62	0.60	0.61	159
talk.religion.misc	0.61	0.22	0.32	136
accuracy			0.70	3770
macro avg	0.71	0.69	0.69	3770
weighted avg	0.71	0.70	0.70	3770



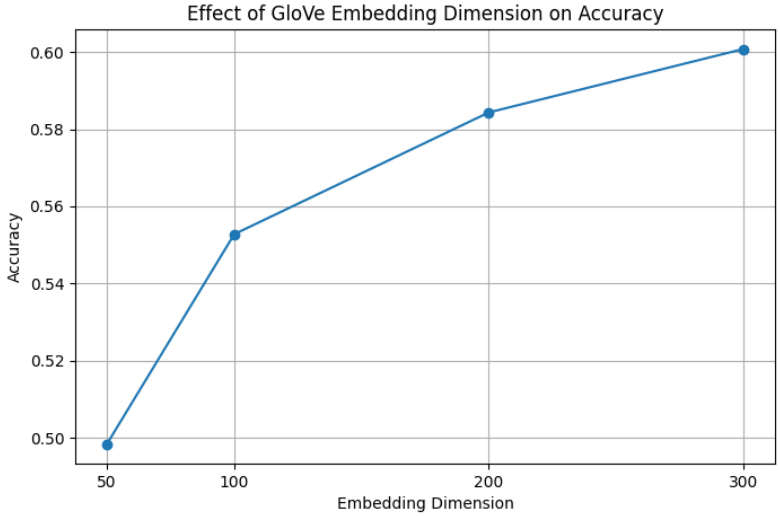


Empirical evaluation reveals that the model using TF-IDF features outperforms the GloVe-based model in terms of classification accuracy. The average accuracy achieved with GloVe vectors is approximately 0.68, whereas TF-IDF yields a higher accuracy of around 0.83. These findings suggest that while pre-trained embeddings offer semantic richness, they may be less effective in shallow models without task-specific fine-tuning or contextual modeling.

Question 3. For Task 2, can you notice a strong performance dependency on the embedding dimensionality?

Solution.

Yes, classification performance **strongly depends** on the dimensionality of the embeddings. The best performance often lies in the range of 300d.



Appendix

1. Task 1

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

# 加载 20 Newsgroups 数据
data = fetch_20newsgroups(subset='all', remove=('headers', 'footers',
'quotes'))
X_raw, y = data.data, data.target

# 文本转 n-gram 特征
vectorizer = TfidfVectorizer(ngram_range=(1, 2), max_features=10000)
X = vectorizer.fit_transform(X_raw)

# 划分训练集与测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 随机森林分类器
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# 评估
y_pred = clf.predict(X_test)
print("Task␣1␣-␣Classification␣Report␣(n-gram):")
print(classification_report(y_test, y_pred, target_names=data.target_names))
```

2. Task 2

```
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
from sklearn.datasets import fetch_20newsgroups
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from sklearn.feature_extraction.text import TfidfVectorizer

# 1. 加载数据
print("Loading dataset...")
newsgroups = fetch_20newsgroups(subset='all', remove=('headers', 'footers',
'quotes'))
texts = newsgroups.data
labels = newsgroups.target
target_names = newsgroups.target_names

# 2. 加载 GloVe 词向量
def load_glove_embeddings(glove_path='glove.6B.100d.txt'):
    print("Loading GloVe vectors...")
    embeddings_index = {}
    with open(glove_path, encoding='utf8') as f:
        for line in tqdm(f, desc="Reading GloVe file"):
            values = line.split()
            word = values[0]
            vector = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = vector
    print(f"Loaded {len(embeddings_index)} word vectors.")
    return embeddings_index

glove_path = 'glove.6B.100d.txt'
if not os.path.exists(glove_path):
    raise FileNotFoundError(f"{glove_path} not found. Please download it from https://nlp.stani
embeddings_index = load_glove_embeddings(glove_path)

# 3. 文本转向量
def text_to_avg_vector(text, embeddings_index, embedding_dim=100):
    words = text.lower().split()
    word_vectors = [embeddings_index[word] for word in words if word in
embeddings_index]
    if not word_vectors:
        return np.zeros(embedding_dim)
    return np.mean(word_vectors, axis=0)

print("Vectorizing texts with GloVe embeddings...")
X_glove = np.array([text_to_avg_vector(text, embeddings_index) for text in
tqdm(texts, desc="Text to vectors")])
y = np.array(labels)

# 4. 划分数据
```

```
X_train_glove, X_test_glove, y_train, y_test = train_test_split(X_glove, y,
test_size=0.2, random_state=42)

# 5. GloVe 模型训练
print("Training classifier with GloVe embeddings...")
clf_glove = LogisticRegression(max_iter=1000)
clf_glove.fit(X_train_glove, y_train)
y_pred_glove = clf_glove.predict(X_test_glove)
acc_glove = accuracy_score(y_test, y_pred_glove)
print(f"\nAccuracy using GloVe: {acc_glove:.4f}")
print(classification_report(y_test, y_pred_glove, target_names=target_names))

# 6. TF-IDF 特征对比
print("Generating TF-IDF features...")
vectorizer = TfidfVectorizer(max_features=10000)
X_tfidf = vectorizer.fit_transform(texts)
X_train_tfidf, X_test_tfidf, _, _ = train_test_split(X_tfidf, y, test_size=0.2,
random_state=42)

print("Training classifier with TF-IDF features...")
clf_tfidf = LogisticRegression(max_iter=1000)
clf_tfidf.fit(X_train_tfidf, y_train)
y_pred_tfidf = clf_tfidf.predict(X_test_tfidf)
acc_tfidf = accuracy_score(y_test, y_pred_tfidf)
print(f"\nAccuracy using TF-IDF: {acc_tfidf:.4f}")
print(classification_report(y_test, y_pred_tfidf, target_names=target_names))

# 7. 绘图比较准确率
plt.figure(figsize=(6, 4))
plt.bar(['GloVe', 'TF-IDF'], [acc_glove, acc_tfidf], color=['skyblue',
'salmon'])
plt.title('Accuracy Comparison')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.grid(True, axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()

# 8. 混淆矩阵可视化（GloVe）
cm = confusion_matrix(y_test, y_pred_glove)
plt.figure(figsize=(12, 10))
sns.heatmap(cm, annot=False, fmt='d', cmap='Blues', xticklabels=target_names,
yticklabels=target_names)
plt.title('Confusion Matrix - GloVe Embeddings')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

3. Task 3

```
import os
```

```
import numpy as np
from tqdm import tqdm
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_20newsgroups
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load dataset
print("Loading 20 Newsgroups dataset...")
data = fetch_20newsgroups(subset='all', remove=('headers', 'footers',
'quotes'))
texts = data.data
labels = data.target

# Helper: Load GloVe embeddings
def load_glove_embeddings(glove_path, dim):
    print(f"\nLoading GloVe {dim}d vectors...")
    embeddings_index = {}
    with open(glove_path, encoding='utf8') as f:
        for line in tqdm(f, desc=f"Reading {dim}d GloVe"):
            values = line.strip().split()
            word = values[0]
            vector = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = vector
    return embeddings_index

# Helper: Convert text to averaged word vectors
def text_to_avg_vector(text, embeddings_index, dim):
    words = text.lower().split()
    word_vectors = [embeddings_index[word] for word in words if word in
embeddings_index]
    if not word_vectors:
        return np.zeros(dim)
    return np.mean(word_vectors, axis=0)

# Define embedding settings
glove_files = {
    50: 'glove.6B.50d.txt',
    100: 'glove.6B.100d.txt',
    200: 'glove.6B.200d.txt',
    300: 'glove.6B.300d.txt',
}
results = {}

# Main loop: Evaluate each embedding dimension
for dim, path in glove_files.items():
    if not os.path.exists(path):

print(f"ß File {path} not found! Please download it from GloVe website.")
        continue

    # Load embeddings
    glove = load_glove_embeddings(path, dim)

    # Vectorize all texts
```

```
    print(f"Vectorizing␣texts␣with␣{dim}d␣embeddings...")
    X_vectors = np.array([text_to_avg_vector(text, glove, dim) for text in
tqdm(texts, desc=f"Vectorizing␣{dim}d")])
    y = np.array(labels)

    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(X_vectors, y,
test_size=0.2, random_state=42)

    # Train classifier
    clf = LogisticRegression(max_iter=1000)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    # Accuracy
    acc = accuracy_score(y_test, y_pred)
    results[dim] = acc
    print(f"ß␣Accuracy␣with␣{dim}d␣embeddings:␣{acc:.4f}")

# Plot results
dims = list(results.keys())
accs = list(results.values())

plt.figure(figsize=(8, 5))
plt.plot(dims, accs, marker='o')
plt.title("Effect␣of␣GloVe␣Embedding␣Dimension␣on␣Accuracy")
plt.xlabel("Embedding␣Dimension")
plt.ylabel("Accuracy")
plt.grid(True)
plt.xticks(dims)
plt.show()
```