AI Capstone Project 1 Report

Chung, Pang-Chun

March 14, 2025

1 Dataset Link

The dataset used in this project is available at this Link. This dataset was collaboratively collected by **鐘邦郡** and **陳冠程**.

2 Research Question

This research investigates methods for text sentiment analysis, aiming to classify textual data into sentiment categories, such as **positive**, **negative** and **neutral**. The study explores different machine learning approaches to improve classification accuracy. In addition, it evaluates the models using metrics such as **recall** and **F1-score** to ensure a balanced assessment of performance, particularly in handling class imbalances.

3 Dataset Documentation

3.1 Features

- **Title**: The title of the Reddit post.
- **Body**: The full text conten of the post.
- **Score**: The net upvotes of the post.
- **Comments**: The total number of comments on the post.
- **Timestamp**: The Unix timestamp indicating when the post was created.
- **Sentiment**: A numerical score generated by the VADER sentiment analysis tool, ranging from -1 (negative) to +1 (positive).
- **Sentiment Label**: A categorical sentiment classification based on the sentiment score.

3.2 Data Source and Collection

The dataset is sourced from **Reddit**, a widely used online platform where users discuss various topics. The data was collected by using the **Python Reddit API Wrapper (PRAW)** [4], which allows programmatic access to Reddit ' s posts. To analyze the sentiment of each post, we utilized the **VADER Sentiment Analyzer** [5]. This tool provides a numberical sentiment score ranging from -1 (most negative) to +1 (most positive). Based on this score, each post was classified into one of the following sentiment categories:

- **Positive**: Sentiment score > 0.05
- **Neutral**: Sentiment score between -0.05 and 0.05
- **Negative**: Sentiment score < -0.05

The labeled dataset was stored in a structured CSV format for further preprocessing and model training.

3.3 Data Composition

The following subreddits were selected to provide diverse types of text:

Subreddit	**Number of Posts**
ArtificialIntelligence	504
StockMarket	423
Music	543
Gaming	664
Business	514
Scams	490
Jobs	705
Community	840
NBA	748
Movies	659
Education	516
ComicBooks	925
Meditation	802
Total	**8333**

Table 1: Distribution of posts across subreddits

In addition to subreddit distribution, we also analyzed the sentiment composition of the dataset. After removing posts with missing texts, the final sentiment distribution is illustrated in Figure 1.

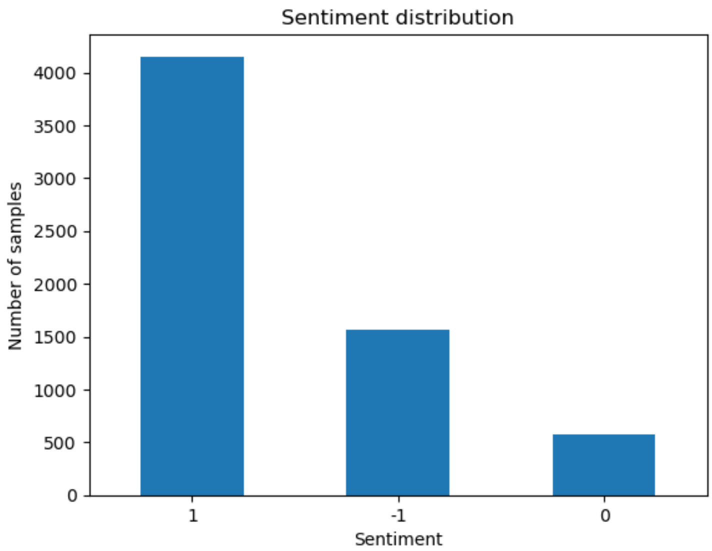

Figure 1: Distribution of sentiment labels (1: Positive, 0: Neutral, -1: Negative).

3.4 Data Example

Due to space constraints and the fact that only the body and sentiment label are used during training, I display only these two fields.

Body	**Sentiment Label**
Elon Musk at Joe Rogan's Experience podcast discusses a new AI feature called Unhinged Mode, which responds aggressively with profanity. Musk also speculates that sex robots might incorporate personality traits through this technology. He expresses concern about AI developments over the next five years.	`Negative`
I would like to see innovative examples other than the classical chat bubble. Does anyone know some interesting websites that integrate AI differently?	`Positive`
I got a text a few days ago claiming to be a debt collector called Unifin. They gave no name or amount owed, only a reference number. I'm sure I have no debt, but I don't know what to do about it.	`Neutral`

Table 2: Example Reddit post with body text and sentiment label.

4 Methods and Models

4.1 Models

In this project, I experimented with two supervised learning models: **Random Forest** and **Support Vector Machine (SVM)**, as well as one unsupervised learning model: **DBSCAN**. These models were directly imported from the **scikit-learn** library [1].

4.2 Methods

There are three different methods applied in my research:

- **Vectorization**: To transform text into numerical representations, I experimented with multiple vectorization techniques: **CountVectorizer** and **TfidfVectorizer** (from scikit-learn [1]), as well as **Word2Vec** (from gensim [2]).
- **Data Resampling**: Due to extreme class imbalance in my dataset(refer figure 1), I applied oversampling (**RandomOverSampler** and **SMOTE** from imbalanced-learn [3]) and undersampling (**RandomUnderSampler**) while using TfidfVectorizer. Imbalanced data can cause classifiers to favor the majority class, leading to biased predictions. By balancing the class distribution, I aimed to improve the model ’ s ability to generalize across all sentiment categories.
- **Data Augmentation**: To improve the diversity of training data, I applied **synonym replacement** from the **WordNet** database [6], where words in the text were replaced with their synonyms using NLP-based synonym retrieval. This augmentation aimed to help models generalize better by learning from slightly varied input representations. For evaluation, I conducted experiments using a **(SVM)** classifier. Additionally, I tested different values of **N** (the number of words randomly replaced per sample) to analyze its impact on classification performance.

5 Experiment Results

5.1 Experiment 1 - Vectorization

Vectorizer	**Accuracy**	**Precision**	**Recall**	**F1-score**
CountVectorizer	0.70	0.70	0.50	0.49
TF-IDF Vectorizer	0.70	0.71	0.48	0.48
Word2Vec	0.71	0.71	0.48	0.51

Table 3: Performance of different Vectorizer using Random Forest.

Vectorizer	Accuracy	Precision	Recall	F1-score
CountVectorizer	0.70	0.61	0.62	0.62
TF-IDF Vectorizer	0.73	0.69	0.55	0.58
Word2Vec	0.71	0.76	0.46	0.49

Table 4: Performance of different Vectorizer using SVM.

Vectorizer	ARI	Silhouette Score	NMI
CountVectorizer	0.10	-0.34	0.08
TF-IDF Vectorizer	0.05	0.10	0.05
Word2Vec	0.15	0.30	0.13

Table 5: Performance of different vectorization methods using DBSCAN.

5.2 Experiment 2 - Data Resampling

Method	Accuracy	Precision	Recall	F1-score
RandomOverSampler	0.70	0.66	0.54	0.53
SMOTE	0.68	0.57	0.55	0.54
RandomUnderSampler	0.71	0.63	0.57	0.57

Table 6: Performance of different Resampling Methods using Random Forest.

Method	Accuracy	Precision	Recall	F1-score
RandomOverSampler	0.72	0.67	0.57	0.60
SMOTE	0.71	0.62	0.64	0.63
RandomUnderSampler	0.71	0.63	0.62	0.62

Table 7: Performance of different Resampling Methods using SVM.

Method	ARI	Silhouette Score	NMI
RandomOverSampler	0.12	0.26	0.22
SMOTE	0.02	0.30	0.12
RandomUnderSampler	0.18	0.25	0.21

Table 8: Performance of different resampling methods using DBSCAN.

5.3 Experiment 3 - Data Augmentation

n	Accuracy	Precision	Recall	F1-score
3	0.73	0.63	0.65	0.64
5	0.73	0.63	0.65	0.64
10	0.72	0.63	0.64	0.63
50	0.71	0.61	0.63	0.62

Table 9: Performance of different values of n in synonym replacement using SVM.

6 Discussion

6.1 Experiment 1 - Vectorization

In Random Forest, all of these three vectorizer achieved the similar accuracy(0.70∼0.71), with Word2Vec slightly outperforming the other two. I initially expected Word2Vec to achieve significantly higher accuracy and F1-score, but the results suggest that Random Forest is not particularly sensitive to different vectorization methods.

SVM achieves the highest accuracy with TF-IDF. However, similar to Random Forest, CountVectorizer achieves the best recall, suggesting that CountVectorizer is more suitable for scenarios where recall is a priority.

In DBSCAN, the experiment result is match what I expected, Word2Vec captures semantic relationships, making it more effective for clustering.

6.2 Experiment 2 - Data Resampling

Due to the highly imbalanced distribution of my dataset, the recall in Experiment 1 was relatively low. To address this issue, I applied various data resampling methods to my dataset.

For the Random Forest model, accuracy remained relatively stable, but recall improved from approximately 0.49 to 0.54 ∼ 0.57, indicating that the model placed greater emphasis on the minority class.

For the SVM model, both accuracy and recall improved with resampling. Accuracy increased slightly, while recall rose from 0.49 to 0.57 ∼ 0.64, suggesting that SVM benefited more from resampling techniques in recognizing the minority class.

For the DBSCAN model, performance varied across different resampling methods. The Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI) fluctuated, indicating that resampling had an impact on clustering consistency. Notably, SMOTE resulted in a lower ARI (0.02) and NMI (0.12) compared to other methods, suggesting that synthetic sampling might have introduced noise that affected clustering performance. Meanwhile, RandomUnderSampler achieved the highest ARI (0.18), implying better alignment with the original class structure.

6.3 Experiment 3 - Data Augmentation

In data augmentation, I observed that when n is set to 3 or 5, both accuracy and F1-score show a slight improvement compared to no augmentation. However, as n increases to 10 or even 50, accuracy starts to decline. This suggests that randomly replacing 3–5 words is optimal for enhancing accuracy in sentiment analysis task, whereas excessive augmentation may introduce noise and negatively impact model performance.

6.4 Future Work

If more time available, I would conduct the following additional experiments :

1. **Hyperparameter Optimization for SVM**

   Currently, I just keep kernel type as linear and regularization parameter as 1. In the future, I plan to perform **grid search** or **Bayesian optimization** to fine-tune key hyperparameters such as **kernel type**(linear, RBF, polynomial), regularization parameter and Gamma (for RBF and polynomial kernels), evaluating how different hyperparameter settings influence model performance, particularly in handling class imbalance.

2. **Extended Data Augmentation Experiments**

   In this project, I only applied synonym replacement as my data augmentation method. In the future, I plan to explore additional augmentation techniques, including **Back-translation** and **Paraphrasing**.

References

[1] Scikit-learn: Machine Learning in Python. Available: https://scikit-learn.org/

[2] Gensim: Topic Modelling for Humans. Available: https://radimrehurek.com/gensim/

[3] Imbalanced-learn: A Python Toolbox for Imbalanced Datasets. Available: https://imbalanced-learn.org/

[4] PRAW: The Python Reddit API Wrapper. Available: https://praw.readthedocs.io/en/stable/

[5] VADER: Valence Aware Dictionary and sEntiment Reasoner. Available: https://github.com/cjhutto/vaderSentiment

[6] WordNet: A large lexical database of English. Available: https://wordnet.princeton.edu/

A Appendix

A.1 Data Collection

```python
import praw
import numpy as np
import pandas as pd
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

# Reddit API Authentication
reddit = praw.Reddit(
    client_id="YOUR_CLIENT_ID",
    client_secret="YOUR_CLIENT",
    user_agent="python:RedditScraper:1.0 (by u/USERNAME)"
)

# Choose subreddits and collect posts
subreddits = ["ArtificialInteligence", "StockMarket", "Music", "gaming",
              "business", "Scams", "jobs", 'community', 'nba', 'movies',
              'education', 'comicbooks', 'Meditation']
posts = []

# Scrape posts
for sub in subreddits:
    cnt = 0
    subreddit = reddit.subreddit(sub)
    for post in subreddit.hot(limit=5000):  # Adjust limit as needed
        cnt += 1
        posts.append([post.title, post.selftext, post.score, post.num_comments,
                      post.created_utc])

    print(f"Subreddit: {sub}, Posts: {cnt}")

# Convert to DataFrame
columns = ["Title", "Body", "Score", "Comments", "Timestamp"]
df = pd.DataFrame(posts, columns=columns)
df.to_csv("reddit_data.csv", index=False)
print("Scraping complete. Data saved!")

# Sentiment Analysis
analyzer = SentimentIntensityAnalyzer()

def get_sentiment(text):
    score = analyzer.polarity_scores(text)
    return score["compound"]  # -1 (negative) to +1 (positive)

# Apply sentiment analysis
df["Sentiment"] = df["Body"].apply(get_sentiment)
conditions = [df["Sentiment"] > 0.05, df["Sentiment"] < -0.05]
labels = ["Positive", "Negative"]
df["Sentiment_Label"] = np.select(conditions, labels, default="Neutral")
```

```python
df.to_csv("reddit_sentiment.csv", index=False)
print("Sentiment analysis complete. Data saved!")
```

A.2 Data Preprocessing

```python
import pandas as pd

def data_preprocessing(df):
    """
    Preprocess the data by removing rows with missing body,
    keeping only the body and sentiment columns, and transforming
    the sentiment labels to integers.

    Parameters
    ----------
    df : pandas.DataFrame
        The DataFrame containing the data.

    Returns
    -------
    df : pandas.DataFrame
        The preprocessed DataFrame.
    """
    # Delete rows with missing body
    df = df.dropna(subset=['Body'])

    # Keep only the body and sentiment columns
    df = df[['Body', 'Sentiment_Label']]

    # Trnasform the sentiment labels to integers
    sentiment_dict = {'Positive': 1, 'Neutral': 0, 'Negative': -1}
    df['Sentiment_Label'] = df['Sentiment_Label'].map(sentiment_dict)

    return df
```

A.3 Random Forest

```python
# import libraries
import pandas as pd
from preprocessing import data_preprocessing
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import cross_validate, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```python

# Read Data from CSV
df = pd.read_csv('reddit_sentiment.csv')

# Preprocess the data
df = data_preprocessing(df)
X = df['Body']
y = df['Sentiment_Label']

# Split the data into training and testing
X_train, X_test, y_train, y_test = train_test_split(
    df['Body'], df['Sentiment_Label'], test_size=0.2, random_state=42,
    stratify=df['Sentiment_Label'], shuffle=True
)

# Experiment 1 - veoctorizer

# Vectorize the data using CountVectorizer
vectorizer = CountVectorizer(max_features=10000, ngram_range=(1, 2))
X_count = vectorizer.fit_transform(X)
X_train_count = vectorizer.transform(X_train)
X_test_count = vectorizer.transform(X_test)

# Cross validation for CountVectorizer
model = RandomForestClassifier(n_estimators=100, random_state=42)
scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_validate(model, X_count, y, cv=skf, scoring=scorings)

# Print the results
print('Accuracy:', np.mean(scores['test_accuracy']))
print('Precision:', np.mean(scores['test_precision_macro']))
print('Recall:', np.mean(scores['test_recall_macro']))
print('F1:', np.mean(scores['test_f1_macro']))

# Draw the confusion matrix
model.fit(X_train_count, y_train)
y_pred = model.predict(X_test_count)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
labels = ['Negative', 'Neutral', 'Positive']
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
            xticklabels=labels, yticklabels=labels)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()


# Vectorize the data using TfidfVectorizer
```

```python
vectorizer = TfidfVectorizer(max_features=10000, ngram_range=(1, 2))
X_tfidf = vectorizer.fit_transform(X)
X_train_tfidf = vectorizer.transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Cross validation for TfidfVectorizer
model_tfidf = RandomForestClassifier(n_estimators=100, random_state=42)
scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_validate(model_tfidf, X_tfidf, y, cv=skf, scoring=scorings)

# Print the results
print('Accuracy:', np.mean(scores['test_accuracy']))
print('Precision:', np.mean(scores['test_precision_macro']))
print('Recall:', np.mean(scores['test_recall_macro']))
print('F1:', np.mean(scores['test_f1_macro']))

# Draw the confusion matrix
model_tfidf.fit(X_train_tfidf, y_train)
y_pred = model_tfidf.predict(X_test_tfidf)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
labels = ['Negative', 'Neutral', 'Positive']
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
            xticklabels=labels, yticklabels=labels)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()


# import the word2vec model
import gensim.downloader as api
w2v_model = api.load("word2vec-google-news-300")

# Function to convert text to word2vec
def text_to_w2v(text, model=w2v_model):
    words = text.split()
    word_vectors = [model[word] for word in words if word in model]
    return np.mean(word_vectors, axis=0) if word_vectors else np.zeros(300)

# Convert text to word2vec
X_w2v = np.array([text_to_w2v(text) for text in X])
X_train_w2v = np.array([text_to_w2v(text) for text in X_train])
X_test_w2v = np.array([text_to_w2v(text) for text in X_test])


# Cross validation for word2vec
model_w2v = RandomForestClassifier(n_estimators=100, random_state=42)
scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
```

```
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_validate(model_w2v, X_w2v, y, cv=skf, scoring=scorings)

# Print the results
print('Accuracy:', np.mean(scores['test_accuracy']))
print('Precision:', np.mean(scores['test_precision_macro']))
print('Recall:', np.mean(scores['test_recall_macro']))
print('F1:', np.mean(scores['test_f1_macro']))

# Experiment 2 - RandomOverSampler, SMOTE, RandomUnderSampler

# Import the libraries
from imblearn.over_sampling import  RandomOverSampler, SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
from sklearn.model_selection import StratifiedKFold

# RandomOverSampler
# Note that we need to use the pipeline to avoid data leakage
model_ros = RandomForestClassifier(n_estimators=100, random_state=42)
ros = RandomOverSampler(random_state=42)
pipeline = Pipeline([
    ('ROS', ros),
    ('RF', model_ros)
])
scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Cross validation for RandomOverSampler
scores = cross_validate(pipeline, X_tfidf, y, cv=cv, scoring=scorings)

# Print the results
print('Accuracy:', np.mean(scores['test_accuracy']))
print('Precision:', np.mean(scores['test_precision_macro']))
print('Recall:', np.mean(scores['test_recall_macro']))
print('F1:', np.mean(scores['test_f1_macro']))


# SMOTE
model_smote = RandomForestClassifier(n_estimators=100, random_state=42)
smote = SMOTE(random_state=42)
pipeline = Pipeline([
    ('SMOTE', smote),
    ('RF', model_smote)
])
scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Cross validation for SMOTE
scores = cross_validate(pipeline, X_tfidf, y, cv=cv, scoring=scorings)

```

```
# Print the results
print('Accuracy:', np.mean(scores['test_accuracy']))
print('Precision:', np.mean(scores['test_precision_macro']))
print('Recall:', np.mean(scores['test_recall_macro']))
print('F1:', np.mean(scores['test_f1_macro']))

# RandomUnderSampler
model_rus = RandomForestClassifier(n_estimators=100, random_state=42)
rus = RandomUnderSampler(random_state=42, sampling_strategy={1: 1800})
pipeline = Pipeline([
    ('RUS', rus),
    ('RF', model_rus)
])
scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Cross validation for RandomUnderSampler
scores = cross_validate(pipeline, X_tfidf, y, cv=cv, scoring=scorings)

# Print the results
print('Accuracy:', np.mean(scores['test_accuracy']))
print('Precision:', np.mean(scores['test_precision_macro']))
print('Recall:', np.mean(scores['test_recall_macro']))
print('F1:', np.mean(scores['test_f1_macro']))
```

A.4 SVM

```
# import libraries
import pandas as pd
from preprocessing import data_preprocessing
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import cross_validate, StratifiedKFold
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Read Data from CSV
df = pd.read_csv('reddit_sentiment.csv')

# Show the Data Distribution
sentiment_distribution = df['Sentiment_Label'].value_counts()
sentiment_distribution.plot(kind='bar')
plt.xlabel('Sentiment')
plt.ylabel('Number of samples')
plt.title('Sentiment distribution')
```

```
plt.xticks(rotation=0)
plt.show()

# Preprocess the data
df = data_preprocessing(df)
print(df.head())

# Show the Data Distribution after Preprocessing
sentiment_distribution = df['Sentiment_Label'].value_counts()
sentiment_distribution.plot(kind='bar')
plt.xlabel('Sentiment')
plt.ylabel('Number␣of␣samples')
plt.title('Sentiment␣distribution')
plt.xticks(rotation=0)
plt.show()

X = df['Body']
y = df['Sentiment_Label']

# Split the data into training and testing
X_train, X_test, y_train, y_test = train_test_split(
    df['Body'], df['Sentiment_Label'], test_size=0.2, random_state=42,
    stratify=df['Sentiment_Label'], shuffle=True
)


# Experiment 1 - Vectorizer

# Vectorize the data using CountVectorizer
vectorizer = CountVectorizer(max_features=10000, ngram_range=(1, 2))
X_count = vectorizer.fit_transform(X)
X_train_count = vectorizer.transform(X_train)
X_test_count = vectorizer.transform(X_test)

# Cross validation for CountVectorizer
model = SVC(kernel='linear', C = 1.0)
scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_validate(model, X_count, y, cv=skf, scoring=scorings)

# Print the results
print('Accuracy:', np.mean(scores['test_accuracy']))
print('Precision:', np.mean(scores['test_precision_macro']))
print('Recall:', np.mean(scores['test_recall_macro']))
print('F1:', np.mean(scores['test_f1_macro']))

# Draw the confusion matrix

model = SVC(kernel='linear', C=1)
model.fit(X_train_count, y_train)
y_pred = model.predict(X_test_count)
```

```python
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion␣Matrix:\n", conf_matrix)
labels = ['Negative', 'Neutral', 'Positive']
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
            xticklabels=labels, yticklabels=labels)
plt.xlabel("Predicted␣Label")
plt.ylabel("True␣Label")
plt.title("Confusion␣Matrix")
plt.show()

# Vectorize the data using TfidfVectorizer
vectorizer = TfidfVectorizer(max_features=10000, ngram_range=(1, 2))
X_tfidf = vectorizer.fit_transform(X)
X_train_tfidf = vectorizer.transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Cross validation for TfidfVectorizer
model_tfidf = SVC(kernel='linear', C = 1.0)
scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_validate(model_tfidf, X_tfidf, y, cv=skf, scoring=scorings)

# Print the results
print('Accuracy:', np.mean(scores['test_accuracy']))
print('Precision:', np.mean(scores['test_precision_macro']))
print('Recall:', np.mean(scores['test_recall_macro']))
print('F1:', np.mean(scores['test_f1_macro']))

# import word2vec model
import gensim.downloader as api
w2v_model = api.load("word2vec-google-news-300")

# Function to convert text to word2vec
def text_to_w2v(text, model=w2v_model):
    words = text.split()
    word_vectors = [model[word] for word in words if word in model]
    return np.mean(word_vectors, axis=0) if word_vectors else np.zeros(300)  # 300 維向量

# Convert text to word2vec
X_w2v = np.array([text_to_w2v(text) for text in X])
X_train_w2v = np.array([text_to_w2v(text) for text in X_train])
X_test_w2v = np.array([text_to_w2v(text) for text in X_test])

# Cross validation for word2vec
model_w2v = SVC(kernel='linear', C = 1.0)
scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_validate(model_w2v, X_w2v, y, cv=skf, scoring=scorings)
```

```
# Print the results
print('Accuracy:', np.mean(scores['test_accuracy']))
print('Precision:', np.mean(scores['test_precision_macro']))
print('Recall:', np.mean(scores['test_recall_macro']))
print('F1:', np.mean(scores['test_f1_macro']))

# Confusion matrix for w2v
model_w2v = SVC(kernel='linear', C=1)
model_w2v.fit(X_train_w2v, y_train)
y_pred_w2v = model_w2v.predict(X_test_w2v)
conf_matrix = confusion_matrix(y_test, y_pred_w2v)
print("Confusion␣Matrix:\n", conf_matrix)
labels = ['Negative', 'Neutral', 'Positive']
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
            xticklabels=labels, yticklabels=labels)
plt.xlabel("Predicted␣Label")
plt.ylabel("True␣Label")
plt.title("Confusion␣Matrix")
plt.show()


# Experiment 2 - RandomOverSampler, SMOTE, RandomUnderSampler
# import libraries
from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
from sklearn.model_selection import StratifiedKFold

# RandomOverSampler
# Note that we need to use the pipeline to avoid data leakage
model_ros = SVC(kernel='linear', C=1.0)
ros = RandomOverSampler(random_state=42, sampling_strategy={-1: 2500})
pipeline = Pipeline([
    ('oversample', ros),
    ('model', model_ros)
])
scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Cross validation for RandomOverSampler
scores = cross_validate(pipeline, X_tfidf, y, cv=cv, scoring=scorings)

# Print the results
print('Accuracy:', np.mean(scores['test_accuracy']))
print('Precision:', np.mean(scores['test_precision_macro']))
print('Recall:', np.mean(scores['test_recall_macro']))
print('F1:', np.mean(scores['test_f1_macro']))

# SMOTE
model_smote = SVC(kernel='linear', C=1.0)
```

```
smote = SMOTE(random_state=42)
pipeline = Pipeline([
    ('smote', smote),
    ('model', model_smote)
])
scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Cross validation for SMOTE
scores = cross_validate(pipeline, X_tfidf, y, cv=cv, scoring=scorings)

# Print the results
print('Accuracy:', np.mean(scores['test_accuracy']))
print('Precision:', np.mean(scores['test_precision_macro']))
print('Recall:', np.mean(scores['test_recall_macro']))
print('F1:', np.mean(scores['test_f1_macro']))

# RandomUnderSampler
model_rus = SVC(kernel='linear', C=1.0)
rus = RandomUnderSampler(random_state=42, sampling_strategy={1: 1800})
pipeline = Pipeline([
    ('undersample', rus),
    ('model', model_rus)
])
scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Cross validation for RandomUnderSampler
scores = cross_validate(pipeline, X_tfidf, y, cv=cv, scoring=scorings)

# Print the results
print('Accuracy:', np.mean(scores['test_accuracy']))
print('Precision:', np.mean(scores['test_precision_macro']))
print('Recall:', np.mean(scores['test_recall_macro']))
print('F1:', np.mean(scores['test_f1_macro']))

# Experiment 3 - Data Augmentation
# import libraries
import nltk
from nltk.corpus import wordnet
import random
nltk.download('wordnet')

# Function to replace words with synonyms
def synonym_replacement(text, n=2):
    if len(text) == 0:
        return text
    words = text.split()
    if(len(words) == 0):
        return text
    new_words = words.copy()
```

```
    for _ in range(n):
        word_idx = random.randint(0, len(words)-1)
        synonyms = wordnet.synsets(words[word_idx])
        if synonyms:
            LEN = len(synonyms)
            new_word = synonyms[random.randint(0, LEN-1)].lemmas()[0].name()
            new_words[word_idx] = new_word
    return "␣".join(new_words)

# Replace words with synonyms and vectorize the data using TfidfVectorizer
X_train_augmented = [synonym_replacement(text, n=50) for text in X_train]
X_train_augmented_tfidf = vectorizer.transform(X_train_augmented)
X_train_augmented_tfidf_smote, y_train_smote = smote.fit_resample(
    X_train_augmented_tfidf, y_train)
# Build the model and train it
model_augmented_smote = SVC(kernel='linear', C=1.0)
model_augmented_smote.fit(X_train_augmented_tfidf_smote, y_train_smote)

# Test the model
y_pred = model_augmented_smote.predict(X_test_tfidf)
accuracy = accuracy_score(y_test, y_pred)
print(f'Augmented␣Accuracy:␣{accuracy:.2f}')
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion␣Matrix:\n", conf_matrix)
labels = ['Negative', 'Neutral', 'Positive']
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
            xticklabels=labels, yticklabels=labels)
plt.xlabel("Predicted␣Label")
plt.ylabel("True␣Label")
plt.title("Confusion␣Matrix")
plt.show()

print(classification_report(y_test, y_pred, target_names=labels))
```

A.5 DBSCAN

```
# import libraries
from sklearn.metrics import adjusted_rand_score, silhouette_score,
from sklearn.metrics import normalized_mutual_info_score
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import pandas as pd
from preprocessing import  data_preprocessing
from sklearn.cluster import DBSCAN
import numpy as np

# Read Data from CSV
df = pd.read_csv('reddit_sentiment.csv')

```

```python
# Preprocess the data
df = data_preprocessing(df)
X = df['Body']
y = df['Sentiment_Label']

# Set the number of clusters
num_clusters = 3

# Experiment 1 - Vectorizer

# Vectorize the data using TfidfVectorizer
vectorizer = TfidfVectorizer(max_features=10)
X_tfidf = vectorizer.fit_transform(X)
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan.fit(X_tfidf)

# Evaluate the clustering
ari = adjusted_rand_score(y, dbscan.labels_)
silhouette = silhouette_score(X_tfidf, dbscan.labels_)
nmi = normalized_mutual_info_score(y, dbscan.labels_)

# Print the results
print(f'Adjusted Rand Index: {ari}')
print(f'Silhouette Score: {silhouette}')
print(f'Normalized Mutual Information: {nmi}')

# Vectorize the data using CountVectorizer
vectorizer = CountVectorizer(max_features=10)
X_count = vectorizer.fit_transform(X)
dbscan_count = DBSCAN(eps=0.5, min_samples=5)
dbscan_count.fit(X_count)

# Evaluate the clustering
ari = adjusted_rand_score(y, dbscan_count.labels_)
silhouette = silhouette_score(X_count, dbscan_count.labels_)
nmi = normalized_mutual_info_score(y, dbscan_count.labels_)

# Print the results
print(f'Adjusted Rand Index: {ari}')
print(f'Silhouette Score: {silhouette}')
print(f'Normalized Mutual Information: {nmi}')

# import word2vec model
import gensim.downloader as api
w2v_model = api.load("word2vec-google-news-300")

# Function to convert text to word2vec
def text_to_w2v(text, model=w2v_model):
    words = text.split()
    word_vectors = [model[word] for word in words if word in model]
    return np.mean(word_vectors, axis=0) if word_vectors else np.zeros(300)
```

```python

# Convert the text data to word2vec
X_w2v = np.array([text_to_w2v(text) for text in X])

# Build the DBSCAN model and fit it
dbscan_w2v = DBSCAN(eps=0.5, min_samples=5)
dbscan_w2v.fit(X_w2v)

# Evaluate the clustering
ari = adjusted_rand_score(y, dbscan_w2v.labels_)
silhouette = silhouette_score(X_w2v, dbscan_w2v.labels_)
nmi = normalized_mutual_info_score(y, dbscan_w2v.labels_)

# Print the results
print(f'Adjusted Rand Index: {ari}')
print(f'Silhouette Score: {silhouette}')
print(f'Normalized Mutual Information: {nmi}')

# Experiment 2 - Over-sampling, SMOTE and Under-sampling
# import libraries
from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.under_sampling import RandomUnderSampler

# RandomOverSampler
ros = RandomOverSampler(random_state=42)
X_ros, y_ros = ros.fit_resample(X_w2v, y)

# Build the DBSCAN model and fit it
model_ros = DBSCAN(eps=0.5, min_samples=5)
model_ros.fit(X_ros)

# Evaluate the clustering
ari = adjusted_rand_score(y_ros, model_ros.labels_)
silhouette = silhouette_score(X_ros, model_ros.labels_)
nmi = normalized_mutual_info_score(y_ros, model_ros.labels_)

# Print the results
print(f'Adjusted Rand Index: {ari}')
print(f'Silhouette Score: {silhouette}')
print(f'Normalized Mutual Information: {nmi}')

# SMOTE
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_smote, y_smote = smote.fit_resample(X_w2v, y)

# Build the DBSCAN model and fit it
model_smote = DBSCAN(eps=0.5, min_samples=5)
model_smote.fit(X_smote)

# Evaluate the clustering
```

```
ari = adjusted_rand_score(y_smote, model_smote.labels_)
silhouette = silhouette_score(X_smote, model_smote.labels_)
nmi = normalized_mutual_info_score(y_smote, model_smote.labels_)

# Print the results
print(f'Adjusted Rand Index: {ari}')
print(f'Silhouette Score: {silhouette}')
print(f'Normalized Mutual Information: {nmi}')

# RandomUnderSampler
rus = RandomUnderSampler(random_state=42)
X_rus, y_rus = rus.fit_resample(X_w2v, y)

# Build the DBSCAN model and fit it
model_rus = DBSCAN(eps=0.5, min_samples=5)
model_rus.fit(X_rus)

# Evaluate the clustering
ari = adjusted_rand_score(y_rus, model_rus.labels_)
silhouette = silhouette_score(X_rus, model_rus.labels_)
nmi = normalized_mutual_info_score(y_rus, model_rus.labels_)

# Print the results
print(f'Adjusted Rand Index: {ari}')
print(f'Silhouette Score: {silhouette}')
print(f'Normalized Mutual Information: {nmi}')
```