

# 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 content 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 numerical 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
<b>Total</b>	<b>8333</b>

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.

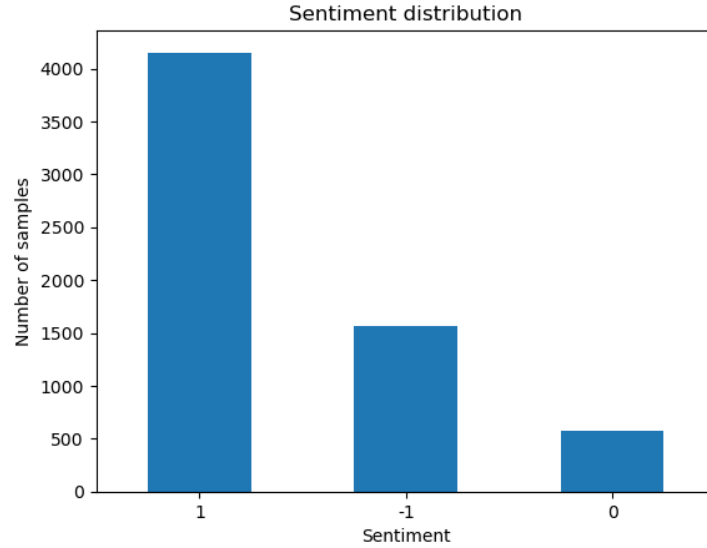


Figure 1: Distribution of sentiment labels (1: Positive, 0: Neutral, -1: Negative).

### 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 pod-cast 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

---

```
1 import praw
2 import numpy as np
3 import pandas as pd
4 from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
5
6 # Reddit API Authentication
7 reddit = praw.Reddit(
8     client_id="YOUR_CLIENT_ID",
9     client_secret="YOUR_CLIENT_SECRET",
10    user_agent="python:RedditScraper:1.0_(by_u/USERNAME)"
11 )
12
13 # Choose subreddits and collect posts
14 subreddits = ["ArtificialIntelligence", "StockMarket", "Music", "gaming",
15              "business", "Scams", "jobs", 'community', 'nba', 'movies',
16              'education', 'comicbooks', 'Meditation']
17 posts = []
18
19 # Scrape posts
20 for sub in subreddits:
21     cnt = 0
22     subreddit = reddit.subreddit(sub)
23     for post in subreddit.hot(limit=5000): # Adjust limit as needed
24         cnt += 1
25         posts.append([post.title, post.selftext, post.score, post.num_comments,
26                      post.created_utc])
27
28     print(f"Subreddit:{sub}, Posts:{cnt}")
29
30 # Convert to DataFrame
31 columns = ["Title", "Body", "Score", "Comments", "Timestamp"]
32 df = pd.DataFrame(posts, columns=columns)
33 df.to_csv("reddit_data.csv", index=False)
34 print("Scraping complete. Data saved!")
35
36 # Sentiment Analysis
37 analyzer = SentimentIntensityAnalyzer()
38
39 def get_sentiment(text):
40     score = analyzer.polarity_scores(text)
41     return score["compound"] # -1 (negative) to +1 (positive)
42
43 # Apply sentiment analysis
44 df["Sentiment"] = df["Body"].apply(get_sentiment)
45 conditions = [df["Sentiment"] > 0.05, df["Sentiment"] < -0.05]
46 labels = ["Positive", "Negative"]
47 df["Sentiment_Label"] = np.select(conditions, labels, default="Neutral")
```



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

---

## A.2 Data Preprocessing

---

```
1 import pandas as pd
2
3 def data_preprocessing(df):
4     """
5     Preprocess the data by removing rows with missing body,
6     keeping only the body and sentiment columns, and transforming
7     the sentiment labels to integers.
8
9     Parameters
10    -----
11    df : pandas.DataFrame
12        The DataFrame containing the data.
13
14    Returns
15    -----
16    df : pandas.DataFrame
17        The preprocessed DataFrame.
18    """
19    # Delete rows with missing body
20    df = df.dropna(subset=['Body'])
21
22    # Keep only the body and sentiment columns
23    df = df[['Body', 'Sentiment_Label']]
24
25    # Transform the sentiment labels to integers
26    sentiment_dict = {'Positive': 1, 'Neutral': 0, 'Negative': -1}
27    df['Sentiment_Label'] = df['Sentiment_Label'].map(sentiment_dict)
28
29    return df
```

---

## A.3 Random Forest

---

```
1 # import libraries
2 import pandas as pd
3 from preprocessing import data_preprocessing
4 from sklearn.model_selection import train_test_split
5 from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
6 from sklearn.model_selection import cross_validate, StratifiedKFold
7 from sklearn.ensemble import RandomForestClassifier
8 from sklearn.metrics import confusion_matrix
9 import matplotlib.pyplot as plt
10 import seaborn as sns
11 import numpy as np
```

```

12
13 # Read Data from CSV
14 df = pd.read_csv('reddit_sentiment.csv')
15
16 # Preprocess the data
17 df = data_preprocessing(df)
18 X = df['Body']
19 y = df['Sentiment_Label']
20
21 # Split the data into training and testing
22 X_train, X_test, y_train, y_test = train_test_split(
23     df['Body'], df['Sentiment_Label'], test_size=0.2, random_state=42,
24     stratify=df['Sentiment_Label'], shuffle=True
25 )
26
27 # Experiment 1 - veovectorizer
28
29 # Vectorize the data using CountVectorizer
30 vectorizer = CountVectorizer(max_features=10000, ngram_range=(1, 2))
31 X_count = vectorizer.fit_transform(X)
32 X_train_count = vectorizer.transform(X_train)
33 X_test_count = vectorizer.transform(X_test)
34
35 # Cross validation for CountVectorizer
36 model = RandomForestClassifier(n_estimators=100, random_state=42)
37 scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
38 skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
39 scores = cross_validate(model, X_count, y, cv=skf, scoring=scorings)
40
41 # Print the results
42 print('Accuracy:', np.mean(scores['test_accuracy']))
43 print('Precision:', np.mean(scores['test_precision_macro']))
44 print('Recall:', np.mean(scores['test_recall_macro']))
45 print('F1:', np.mean(scores['test_f1_macro']))
46
47 # Draw the confusion matrix
48 model.fit(X_train_count, y_train)
49 y_pred = model.predict(X_test_count)
50 conf_matrix = confusion_matrix(y_test, y_pred)
51 print("Confusion_Matrix:\n", conf_matrix)
52 labels = ['Negative', 'Neutral', 'Positive']
53 plt.figure(figsize=(6, 5))
54 sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
55             xticklabels=labels, yticklabels=labels)
56 plt.xlabel("Predicted_Label")
57 plt.ylabel("True_Label")
58 plt.title("Confusion_Matrix")
59 plt.show()
60
61
62 # Vectorize the data using TfidfVectorizer

```

```

63 vectorizer = TfidfVectorizer(max_features=10000, ngram_range=(1, 2))
64 X_tfidf = vectorizer.fit_transform(X)
65 X_train_tfidf = vectorizer.transform(X_train)
66 X_test_tfidf = vectorizer.transform(X_test)
67
68 # Cross validation for TfidfVectorizer
69 model_tfidf = RandomForestClassifier(n_estimators=100, random_state=42)
70 scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
71 skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
72 scores = cross_validate(model_tfidf, X_tfidf, y, cv=skf, scoring=scorings)
73
74 # Print the results
75 print('Accuracy:', np.mean(scores['test_accuracy']))
76 print('Precision:', np.mean(scores['test_precision_macro']))
77 print('Recall:', np.mean(scores['test_recall_macro']))
78 print('F1:', np.mean(scores['test_f1_macro']))
79
80 # Draw the confusion matrix
81 model_tfidf.fit(X_train_tfidf, y_train)
82 y_pred = model_tfidf.predict(X_test_tfidf)
83 conf_matrix = confusion_matrix(y_test, y_pred)
84 print("Confusion_Matrix:\n", conf_matrix)
85 labels = ['Negative', 'Neutral', 'Positive']
86 plt.figure(figsize=(6, 5))
87 sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
88             xticklabels=labels, yticklabels=labels)
89 plt.xlabel("Predicted_Label")
90 plt.ylabel("True_Label")
91 plt.title("Confusion_Matrix")
92 plt.show()
93
94
95 # import the word2vec model
96 import gensim.downloader as api
97 w2v_model = api.load("word2vec-google-news-300")
98
99 # Function to convert text to word2vec
100 def text_to_w2v(text, model=w2v_model):
101     words = text.split()
102     word_vectors = [model[word] for word in words if word in model]
103     return np.mean(word_vectors, axis=0) if word_vectors else np.zeros(300)
104
105 # Convert text to word2vec
106 X_w2v = np.array([text_to_w2v(text) for text in X])
107 X_train_w2v = np.array([text_to_w2v(text) for text in X_train])
108 X_test_w2v = np.array([text_to_w2v(text) for text in X_test])
109
110
111 # Cross validation for word2vec
112 model_w2v = RandomForestClassifier(n_estimators=100, random_state=42)
113 scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']

```

```

114 skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
115 scores = cross_validate(model_w2v, X_w2v, y, cv=skf, scoring=scorings)
116
117 # Print the results
118 print('Accuracy:', np.mean(scores['test_accuracy']))
119 print('Precision:', np.mean(scores['test_precision_macro']))
120 print('Recall:', np.mean(scores['test_recall_macro']))
121 print('F1:', np.mean(scores['test_f1_macro']))
122
123 # Experiment 2 - RandomOverSampler, SMOTE, RandomUnderSampler
124
125 # Import the libraries
126 from imblearn.over_sampling import RandomOverSampler, SMOTE
127 from imblearn.under_sampling import RandomUnderSampler
128 from imblearn.pipeline import Pipeline
129 from sklearn.model_selection import StratifiedKFold
130
131 # RandomOverSampler
132 # Note that we need to use the pipeline to avoid data leakage
133 model_ros = RandomForestClassifier(n_estimators=100, random_state=42)
134 ros = RandomOverSampler(random_state=42)
135 pipeline = Pipeline([
136     ('ROS', ros),
137     ('RF', model_ros)
138 ])
139 scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
140 cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
141
142 # Cross validation for RandomOverSampler
143 scores = cross_validate(pipeline, X_tfidf, y, cv=cv, scoring=scorings)
144
145 # Print the results
146 print('Accuracy:', np.mean(scores['test_accuracy']))
147 print('Precision:', np.mean(scores['test_precision_macro']))
148 print('Recall:', np.mean(scores['test_recall_macro']))
149 print('F1:', np.mean(scores['test_f1_macro']))
150
151
152 # SMOTE
153 model_smote = RandomForestClassifier(n_estimators=100, random_state=42)
154 smote = SMOTE(random_state=42)
155 pipeline = Pipeline([
156     ('SMOTE', smote),
157     ('RF', model_smote)
158 ])
159 scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
160 cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
161
162 # Cross validation for SMOTE
163 scores = cross_validate(pipeline, X_tfidf, y, cv=cv, scoring=scorings)
164

```

```

165 # Print the results
166 print('Accuracy:', np.mean(scores['test_accuracy']))
167 print('Precision:', np.mean(scores['test_precision_macro']))
168 print('Recall:', np.mean(scores['test_recall_macro']))
169 print('F1:', np.mean(scores['test_f1_macro']))
170
171 # RandomUnderSampler
172 model_rus = RandomForestClassifier(n_estimators=100, random_state=42)
173 rus = RandomUnderSampler(random_state=42, sampling_strategy={1: 1800})
174 pipeline = Pipeline([
175     ('RUS', rus),
176     ('RF', model_rus)
177 ])
178 scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
179 cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
180
181 # Cross validation for RandomUnderSampler
182 scores = cross_validate(pipeline, X_tfidf, y, cv=cv, scoring=scorings)
183
184 # Print the results
185 print('Accuracy:', np.mean(scores['test_accuracy']))
186 print('Precision:', np.mean(scores['test_precision_macro']))
187 print('Recall:', np.mean(scores['test_recall_macro']))
188 print('F1:', np.mean(scores['test_f1_macro']))

```

---

## A.4 SVM

---

```

1 # import libraries
2 import pandas as pd
3 from preprocessing import data_preprocessing
4 from sklearn.model_selection import train_test_split
5 from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
6 from sklearn.model_selection import cross_validate, StratifiedKFold
7 from sklearn.svm import SVC
8 from sklearn.metrics import accuracy_score, classification_report
9 from sklearn.metrics import confusion_matrix
10 import matplotlib.pyplot as plt
11 import seaborn as sns
12 import numpy as np
13
14 # Read Data from CSV
15 df = pd.read_csv('reddit_sentiment.csv')
16
17 # Show the Data Distribution
18 sentiment_distribution = df['Sentiment_Label'].value_counts()
19 sentiment_distribution.plot(kind='bar')
20 plt.xlabel('Sentiment')
21 plt.ylabel('Number_of_samples')
22 plt.title('Sentiment_distribution')

```

```

23 plt.xticks(rotation=0)
24 plt.show()
25
26 # Preprocess the data
27 df = data_preprocessing(df)
28 print(df.head())
29
30 # Show the Data Distribution after Preprocessing
31 sentiment_distribution = df['Sentiment_Label'].value_counts()
32 sentiment_distribution.plot(kind='bar')
33 plt.xlabel('Sentiment')
34 plt.ylabel('Number_of_samples')
35 plt.title('Sentiment_distribution')
36 plt.xticks(rotation=0)
37 plt.show()
38
39 X = df['Body']
40 y = df['Sentiment_Label']
41
42 # Split the data into training and testing
43 X_train, X_test, y_train, y_test = train_test_split(
44     df['Body'], df['Sentiment_Label'], test_size=0.2, random_state=42,
45     stratify=df['Sentiment_Label'], shuffle=True
46 )
47
48
49 # Experiment 1 - Vectorizer
50
51 # Vectorize the data using CountVectorizer
52 vectorizer = CountVectorizer(max_features=10000, ngram_range=(1, 2))
53 X_count = vectorizer.fit_transform(X)
54 X_train_count = vectorizer.transform(X_train)
55 X_test_count = vectorizer.transform(X_test)
56
57 # Cross validation for CountVectorizer
58 model = SVC(kernel='linear', C = 1.0)
59 scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
60 skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
61 scores = cross_validate(model, X_count, y, cv=skf, scoring=scorings)
62
63 # Print the results
64 print('Accuracy:', np.mean(scores['test_accuracy']))
65 print('Precision:', np.mean(scores['test_precision_macro']))
66 print('Recall:', np.mean(scores['test_recall_macro']))
67 print('F1:', np.mean(scores['test_f1_macro']))
68
69 # Draw the confusion matrix
70
71 model = SVC(kernel='linear', C=1)
72 model.fit(X_train_count, y_train)
73 y_pred = model.predict(X_test_count)

```

```

74 conf_matrix = confusion_matrix(y_test, y_pred)
75 print("Confusion_Matrix:\n", conf_matrix)
76 labels = ['Negative', 'Neutral', 'Positive']
77 plt.figure(figsize=(6, 5))
78 sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
79             xticklabels=labels, yticklabels=labels)
80 plt.xlabel("Predicted_Label")
81 plt.ylabel("True_Label")
82 plt.title("Confusion_Matrix")
83 plt.show()
84
85 # Vectorize the data using TfidfVectorizer
86 vectorizer = TfidfVectorizer(max_features=10000, ngram_range=(1, 2))
87 X_tfidf = vectorizer.fit_transform(X)
88 X_train_tfidf = vectorizer.transform(X_train)
89 X_test_tfidf = vectorizer.transform(X_test)
90
91 # Cross validation for TfidfVectorizer
92 model_tfidf = SVC(kernel='linear', C = 1.0)
93 scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
94 skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
95 scores = cross_validate(model_tfidf, X_tfidf, y, cv=skf, scoring=scorings)
96
97 # Print the results
98 print('Accuracy:', np.mean(scores['test_accuracy']))
99 print('Precision:', np.mean(scores['test_precision_macro']))
100 print('Recall:', np.mean(scores['test_recall_macro']))
101 print('F1:', np.mean(scores['test_f1_macro']))
102
103 # import word2vec model
104 import gensim.downloader as api
105 w2v_model = api.load("word2vec-google-news-300")
106
107 # Function to convert text to word2vec
108 def text_to_w2v(text, model=w2v_model):
109     words = text.split()
110     word_vectors = [model[word] for word in words if word in model]
111     return np.mean(word_vectors, axis=0) if word_vectors else np.zeros(300)
112 # 300 維向量
113
114 # Convert text to word2vec
115 X_w2v = np.array([text_to_w2v(text) for text in X])
116 X_train_w2v = np.array([text_to_w2v(text) for text in X_train])
117 X_test_w2v = np.array([text_to_w2v(text) for text in X_test])
118
119 # Cross validation for word2vec
120 model_w2v = SVC(kernel='linear', C = 1.0)
121 scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
122 skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
123 scores = cross_validate(model_w2v, X_w2v, y, cv=skf, scoring=scorings)

```

```

124 # Print the results
125 print('Accuracy:', np.mean(scores['test_accuracy']))
126 print('Precision:', np.mean(scores['test_precision_macro']))
127 print('Recall:', np.mean(scores['test_recall_macro']))
128 print('F1:', np.mean(scores['test_f1_macro']))
129
130 # Confusion matrix for w2v
131 model_w2v = SVC(kernel='linear', C=1)
132 model_w2v.fit(X_train_w2v, y_train)
133 y_pred_w2v = model_w2v.predict(X_test_w2v)
134 conf_matrix = confusion_matrix(y_test, y_pred_w2v)
135 print("Confusion_Matrix:\n", conf_matrix)
136 labels = ['Negative', 'Neutral', 'Positive']
137 plt.figure(figsize=(6, 5))
138 sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
139             xticklabels=labels, yticklabels=labels)
140 plt.xlabel("Predicted_Label")
141 plt.ylabel("True_Label")
142 plt.title("Confusion_Matrix")
143 plt.show()
144
145
146 # Experiment 2 - RandomOverSampler, SMOTE, RandomUnderSampler
147 # import libraries
148 from imblearn.over_sampling import RandomOverSampler, SMOTE
149 from imblearn.under_sampling import RandomUnderSampler
150 from imblearn.pipeline import Pipeline
151 from sklearn.model_selection import StratifiedKFold
152
153 # RandomOverSampler
154 # Note that we need to use the pipeline to avoid data leakage
155 model_ros = SVC(kernel='linear', C=1.0)
156 ros = RandomOverSampler(random_state=42, sampling_strategy={-1: 2500})
157 pipeline = Pipeline([
158     ('oversample', ros),
159     ('model', model_ros)
160 ])
161 scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
162 cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
163
164 # Cross validation for RandomOverSampler
165 scores = cross_validate(pipeline, X_tfidf, y, cv=cv, scoring=scorings)
166
167 # Print the results
168 print('Accuracy:', np.mean(scores['test_accuracy']))
169 print('Precision:', np.mean(scores['test_precision_macro']))
170 print('Recall:', np.mean(scores['test_recall_macro']))
171 print('F1:', np.mean(scores['test_f1_macro']))
172
173 # SMOTE
174 model_smote = SVC(kernel='linear', C=1.0)

```



```

175 smote = SMOTE(random_state=42)
176 pipeline = Pipeline([
177     ('smote', smote),
178     ('model', model_smote)
179 ])
180 scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
181 cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
182
183 # Cross validation for SMOTE
184 scores = cross_validate(pipeline, X_tfidf, y, cv=cv, scoring=scorings)
185
186 # Print the results
187 print('Accuracy:', np.mean(scores['test_accuracy']))
188 print('Precision:', np.mean(scores['test_precision_macro']))
189 print('Recall:', np.mean(scores['test_recall_macro']))
190 print('F1:', np.mean(scores['test_f1_macro']))
191
192 # RandomUnderSampler
193 model_rus = SVC(kernel='linear', C=1.0)
194 rus = RandomUnderSampler(random_state=42, sampling_strategy={1: 1800})
195 pipeline = Pipeline([
196     ('undersample', rus),
197     ('model', model_rus)
198 ])
199 scorings = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
200 cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
201
202 # Cross validation for RandomUnderSampler
203 scores = cross_validate(pipeline, X_tfidf, y, cv=cv, scoring=scorings)
204
205 # Print the results
206 print('Accuracy:', np.mean(scores['test_accuracy']))
207 print('Precision:', np.mean(scores['test_precision_macro']))
208 print('Recall:', np.mean(scores['test_recall_macro']))
209 print('F1:', np.mean(scores['test_f1_macro']))
210
211 # Experiment 3 - Data Augmentation
212 # import libraries
213 import nltk
214 from nltk.corpus import wordnet
215 import random
216 nltk.download('wordnet')
217
218 # Function to replace words with synonyms
219 def synonym_replacement(text, n=2):
220     if len(text) == 0:
221         return text
222     words = text.split()
223     if len(words) == 0:
224         return text
225     new_words = words.copy()

```

```

226     for _ in range(n):
227         word_idx = random.randint(0, len(words)-1)
228         synonyms = wordnet.synsets(words[word_idx])
229         if synonyms:
230             LEN = len(synonyms)
231             new_word = synonyms[random.randint(0, LEN-1)].lemmas()[0].name()
232             new_words[word_idx] = new_word
233     return " ".join(new_words)
234
235 # Replace words with synonyms and vectorize the data using TfidfVectorizer
236 X_train_augmented = [synonym_replacement(text, n=50) for text in X_train]
237 X_train_augmented_tfidf = vectorizer.transform(X_train_augmented)
238 X_train_augmented_tfidf_smote, y_train_smote = smote.fit_resample(
239     X_train_augmented_tfidf, y_train)
240 # Build the model and train it
241 model_augmented_smote = SVC(kernel='linear', C=1.0)
242 model_augmented_smote.fit(X_train_augmented_tfidf_smote, y_train_smote)
243
244 # Test the model
245 y_pred = model_augmented_smote.predict(X_test_tfidf)
246 accuracy = accuracy_score(y_test, y_pred)
247 print(f'Augmented Accuracy: {accuracy:.2f}')
248 conf_matrix = confusion_matrix(y_test, y_pred)
249 print("Confusion Matrix:\n", conf_matrix)
250 labels = ['Negative', 'Neutral', 'Positive']
251 plt.figure(figsize=(6, 5))
252 sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
253             xticklabels=labels, yticklabels=labels)
254 plt.xlabel("Predicted Label")
255 plt.ylabel("True Label")
256 plt.title("Confusion Matrix")
257 plt.show()
258
259 print(classification_report(y_test, y_pred, target_names=labels))

```

---

## A.5 DBSCAN

---

```

1 # import libraries
2 from sklearn.metrics import adjusted_rand_score, silhouette_score,
3 from sklearn.metrics import normalized_mutual_info_score
4 from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
5 import pandas as pd
6 from preprocessing import data_preprocessing
7 from sklearn.cluster import DBSCAN
8 import numpy as np
9
10 # Read Data from CSV
11 df = pd.read_csv('reddit_sentiment.csv')
12

```

```

13 # Preprocess the data
14 df = data_preprocessing(df)
15 X = df['Body']
16 y = df['Sentiment_Label']
17
18 # Set the number of clusters
19 num_clusters = 3
20
21 # Experiment 1 - Vectorizer
22
23 # Vectorize the data using TfidfVectorizer
24 vectorizer = TfidfVectorizer(max_features=10)
25 X_tfidf = vectorizer.fit_transform(X)
26 dbscan = DBSCAN(eps=0.5, min_samples=5)
27 dbscan.fit(X_tfidf)
28
29 # Evaluate the clustering
30 ari = adjusted_rand_score(y, dbscan.labels_)
31 silhouette = silhouette_score(X_tfidf, dbscan.labels_)
32 nmi = normalized_mutual_info_score(y, dbscan.labels_)
33
34 # Print the results
35 print(f'Adjusted_Rand_Index:_{ari}')
36 print(f'Silhouette_Score:_{silhouette}')
37 print(f'Normalized_Mutual_Information:_{nmi}')
38
39 # Vectorize the data using CountVectorizer
40 vectorizer = CountVectorizer(max_features=10)
41 X_count = vectorizer.fit_transform(X)
42 dbscan_count = DBSCAN(eps=0.5, min_samples=5)
43 dbscan_count.fit(X_count)
44
45 # Evaluate the clustering
46 ari = adjusted_rand_score(y, dbscan_count.labels_)
47 silhouette = silhouette_score(X_count, dbscan_count.labels_)
48 nmi = normalized_mutual_info_score(y, dbscan_count.labels_)
49
50 # Print the results
51 print(f'Adjusted_Rand_Index:_{ari}')
52 print(f'Silhouette_Score:_{silhouette}')
53 print(f'Normalized_Mutual_Information:_{nmi}')
54
55 # import word2vec model
56 import gensim.downloader as api
57 w2v_model = api.load("word2vec-google-news-300")
58
59 # Function to convert text to word2vec
60 def text_to_w2v(text, model=w2v_model):
61     words = text.split()
62     word_vectors = [model[word] for word in words if word in model]
63     return np.mean(word_vectors, axis=0) if word_vectors else np.zeros(300)

```

```

64
65 # Convert the text data to word2vec
66 X_w2v = np.array([text_to_w2v(text) for text in X])
67
68 # Build the DBSCAN model and fit it
69 dbscan_w2v = DBSCAN(eps=0.5, min_samples=5)
70 dbscan_w2v.fit(X_w2v)
71
72 # Evaluate the clustering
73 ari = adjusted_rand_score(y, dbscan_w2v.labels_)
74 silhouette = silhouette_score(X_w2v, dbscan_w2v.labels_)
75 nmi = normalized_mutual_info_score(y, dbscan_w2v.labels_)
76
77 # Print the results
78 print(f'Adjusted_Rand_Index:_{ari}')
79 print(f'Silhouette_Score:_{silhouette}')
80 print(f'Normalized_Mutual_Information:_{nmi}')
81
82 # Experiment 2 - Over-sampling, SMOTE and Under-sampling
83 # import libraries
84 from imblearn.over_sampling import RandomOverSampler, SMOTE
85 from imblearn.under_sampling import RandomUnderSampler
86
87 # RandomOverSampler
88 ros = RandomOverSampler(random_state=42)
89 X_ros, y_ros = ros.fit_resample(X_w2v, y)
90
91 # Build the DBSCAN model and fit it
92 model_ros = DBSCAN(eps=0.5, min_samples=5)
93 model_ros.fit(X_ros)
94
95 # Evaluate the clustering
96 ari = adjusted_rand_score(y_ros, model_ros.labels_)
97 silhouette = silhouette_score(X_ros, model_ros.labels_)
98 nmi = normalized_mutual_info_score(y_ros, model_ros.labels_)
99
100 # Print the results
101 print(f'Adjusted_Rand_Index:_{ari}')
102 print(f'Silhouette_Score:_{silhouette}')
103 print(f'Normalized_Mutual_Information:_{nmi}')
104
105 # SMOTE
106 from imblearn.over_sampling import SMOTE
107 smote = SMOTE(random_state=42)
108 X_smote, y_smote = smote.fit_resample(X_w2v, y)
109
110 # Build the DBSCAN model and fit it
111 model_smote = DBSCAN(eps=0.5, min_samples=5)
112 model_smote.fit(X_smote)
113
114 # Evaluate the clustering

```

```

115 ari = adjusted_rand_score(y_smote, model_smote.labels_)
116 silhouette = silhouette_score(X_smote, model_smote.labels_)
117 nmi = normalized_mutual_info_score(y_smote, model_smote.labels_)
118
119 # Print the results
120 print(f'Adjusted_Rand_Index:_{ari}')
121 print(f'Silhouette_Score:_{silhouette}')
122 print(f'Normalized_Mutual_Information:_{nmi}')
123
124 # RandomUnderSampler
125 rus = RandomUnderSampler(random_state=42)
126 X_rus, y_rus = rus.fit_resample(X_w2v, y)
127
128 # Build the DBSCAN model and fit it
129 model_rus = DBSCAN(eps=0.5, min_samples=5)
130 model_rus.fit(X_rus)
131
132 # Evaluate the clustering
133 ari = adjusted_rand_score(y_rus, model_rus.labels_)
134 silhouette = silhouette_score(X_rus, model_rus.labels_)
135 nmi = normalized_mutual_info_score(y_rus, model_rus.labels_)
136
137 # Print the results
138 print(f'Adjusted_Rand_Index:_{ari}')
139 print(f'Silhouette_Score:_{silhouette}')
140 print(f'Normalized_Mutual_Information:_{nmi}')

```

---