

NLP Classification Project

Student Information

Name: Sultan Abdulrahim Hassan

Student ID: 251002896

Course: Practical Image Processing and Natural Language Processing_Third Trimester 2025

Instructor: Dr. Hajar Saleh

Date: 20 - 12 - 2025

1. Environment Setup

Below we install and import all required libraries used in this project.

```
In [17]: import sys

# Install matplotlib in the current environment
!{sys.executable} -m pip install matplotlib
```

```

Collecting matplotlib
  Downloading matplotlib-3.10.8-cp310-cp310-macosx_10_12_x86_64.whl.metadata (52 kB)
Collecting contourpy>=1.0.1 (from matplotlib)
  Downloading contourpy-1.3.2-cp310-cp310-macosx_10_9_x86_64.whl.metadata (5.5 kB)
Collecting cyclers>=0.10 (from matplotlib)
  Using cached cyclers-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib)
  Downloading fonttools-4.61.1-cp310-cp310-macosx_10_9_x86_64.whl.metadata (114 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib)
  Downloading kiwisolver-1.4.9-cp310-cp310-macosx_10_9_x86_64.whl.metadata (6.3 kB)
Requirement already satisfied: numpy>=1.23 in /opt/anaconda3/envs/NLP_Project/lib/python3.10/site-packages (from matplotlib) (2.2.6)
Requirement already satisfied: packaging>=20.0 in /opt/anaconda3/envs/NLP_Project/lib/python3.10/site-packages (from matplotlib) (25.0)
Collecting pillow>=8 (from matplotlib)
  Downloading pillow-12.0.0-cp310-cp310-macosx_10_10_x86_64.whl.metadata (8.8 kB)
Collecting pyparsing>=3 (from matplotlib)
  Using cached pyparsing-3.2.5-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: python-dateutil>=2.7 in /opt/anaconda3/envs/NLP_Project/lib/python3.10/site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/envs/NLP_Project/lib/python3.10/site-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
Downloading matplotlib-3.10.8-cp310-cp310-macosx_10_12_x86_64.whl (8.2 MB)
_____ 8.2/8.2 MB 22.8 MB/s 0:00:00m0:00:0100:01
Downloading contourpy-1.3.2-cp310-cp310-macosx_10_9_x86_64.whl (268 kB)
Using cached cyclers-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.61.1-cp310-cp310-macosx_10_9_x86_64.whl (2.4 MB)
_____ 2.4/2.4 MB 11.0 MB/s 0:00:00
Downloading kiwisolver-1.4.9-cp310-cp310-macosx_10_9_x86_64.whl (66 kB)
Downloading pillow-12.0.0-cp310-cp310-macosx_10_10_x86_64.whl (5.3 MB)
_____ 5.3/5.3 MB 33.1 MB/s 0:00:00
Using cached pyparsing-3.2.5-py3-none-any.whl (113 kB)
Installing collected packages: pyparsing, pillow, kiwisolver, fonttools, cyclers, contourpy, matplotlib
_____ 7/7 [matplotlib]7 [matplotlib]
Successfully installed contourpy-1.3.2 cyclers-0.12.1 fonttools-4.61.1 kiwisolver-1.4.9 matplotlib-3.10.8 pillow-12.0.0 pyparsing-3.2.5

```

2. Load the Dataset

We load the unbalanced sentiment dataset and display the first few rows.

```
In [10]: import pandas as pd
```

```
df = pd.read_csv("unbalanceddataset.csv")
df.head()
```

```
Out[10]:
```

	text	sentiment
0	Java Concurrency in Practice is probably the b...	positive
1	haha aww hun i bet you are more creative tha...	positive
2	_pickle lol, thank you very much Hope you`re h...	positive
3	Out for an evening on the town with jeremy. Sa...	negative
4	- just took over the #1 Most Endorsed spot on...	positive

3. Visualize Class Distribution

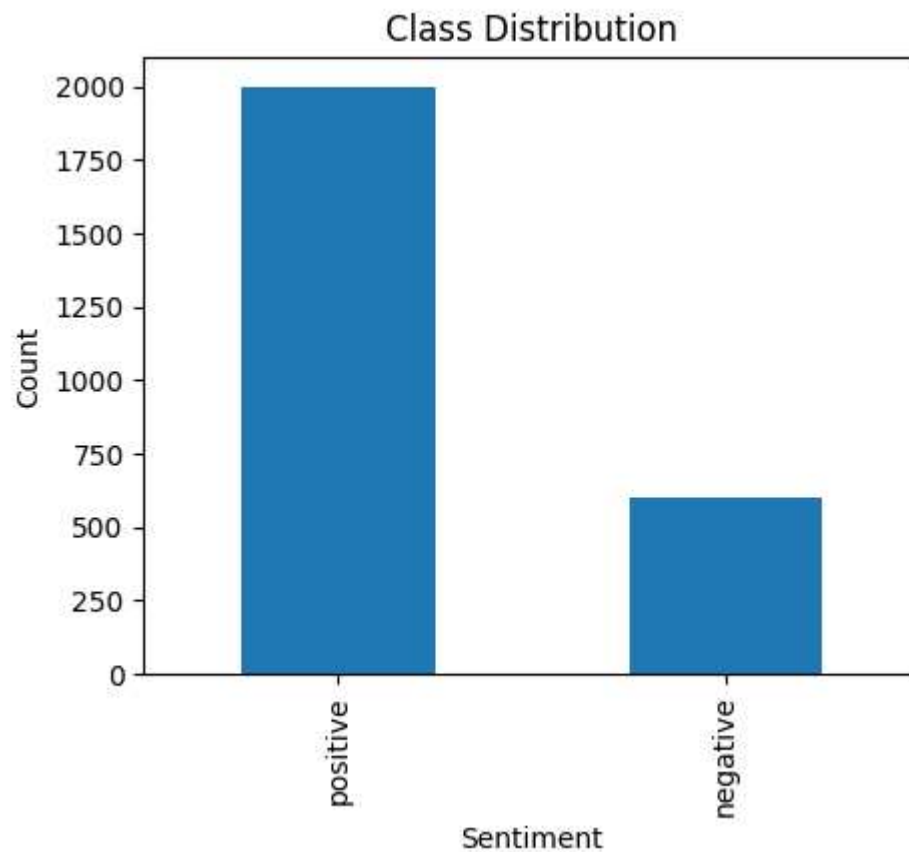
Before preprocessing, we examine whether the dataset is balanced by plotting the class counts.

```
In [18]: import matplotlib.pyplot as plt

# Bar plot for class distribution
class_counts = df["sentiment"].value_counts()

plt.figure(figsize=(5, 4))
class_counts.plot(kind="bar")
plt.title("Class Distribution")
plt.xlabel("Sentiment")
plt.ylabel("Count")
plt.show()

print(class_counts)
```



```
sentiment
positive    2000
negative     600
Name: count, dtype: int64
```

4. Data Preprocessing

The dataset is cleaned using:

- Lowercasing
- Removing URLs
- Removing mentions (@)
- Removing hashtags
- Keeping alphabetic characters only
- Removing extra spaces

A new column called **clean_text** is created.

```
In [11]: import re

def clean_text(text):
    text = str(text).lower() # Lowercase
```

```

text = re.sub(r"http\S+|www.\S+", " ", text) # Remove URLs
text = re.sub(r"@w+", " ", text) # Remove @mentions
text = re.sub(r"#", " ", text) # Remove hashtags
text = re.sub(r"[^a-z\s']", " ", text) # Keep letters only
text = re.sub(r"\s+", " ", text).strip() # Remove extra spaces
return text

```

```

df["clean_text"] = df["text"].apply(clean_text)
df[["text", "clean_text"]].head()

```

Out [11]:

	text	clean_text
0	Java Concurrency in Practice is probably the b...	java concurrency in practice is probably the b...
1	haha aww hun i bet you are more creative tha...	haha aww hun i bet you are more creative than me
2	_pickle lol, thank you very much Hope you`re h...	pickle lol thank you very much hope you re hav...
3	Out for an evening on the town with jeremy. Sa...	out for an evening on the town with jeremy sad...
4	- just took over the #1 Most Endorsed spot on...	just took over the most endorsed spot on twind...

5. Split the Dataset

We split the dataset into training and testing sets using an 80–20 ratio. Stratify is used to preserve class proportions.

In [12]: `from sklearn.model_selection import train_test_split`

```

X = df["clean_text"]
y = df["sentiment"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    stratify=y,
    random_state=42
)

len(X_train), len(X_test)

```

Out [12]: (2080, 520)

6. Feature Representation (TF-IDF)

We convert clean text into numerical vectors using TF-IDF with:

- max_features=10000
- ngram_range = (1,2)

In [13]: `from sklearn.feature_extraction.text import TfidfVectorizer`

```

tfidf = TfidfVectorizer(
    max_features=10000,
    ngram_range=(1, 2),
    min_df=2
)

```

7. Model 1: Logistic Regression

We train a logistic regression classifier using `class_weight='balanced'`. Below is the classification report and confusion matrix.

```
In [14]: from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix

logreg_model = Pipeline([
    ("tfidf", tfidf),
    ("clf", LogisticRegression(class_weight="balanced", max_iter=1000))
])

logreg_model.fit(X_train, y_train)
y_pred = logreg_model.predict(X_test)

print("Logistic Regression Results:\n")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

Logistic Regression Results:

	precision	recall	f1-score	support
negative	0.59	0.76	0.66	120
positive	0.92	0.84	0.88	400
accuracy			0.82	520
macro avg	0.76	0.80	0.77	520
weighted avg	0.84	0.82	0.83	520

Confusion Matrix:

```
[[ 91  29]
 [ 63 337]]
```

8. Model 2: Linear SVM (Support Vector Machine)

We train an SVM classifier using the same TF-IDF features. Here are the performance metrics and confusion matrix.

```
In [15]: from sklearn.svm import LinearSVC

svm_model = Pipeline([
    ("tfidf", tfidf),          # same TF-IDF settings
    ("clf", LinearSVC())       # Linear Support Vector Machine
])

svm_model.fit(X_train, y_train)
y_pred2 = svm_model.predict(X_test)

print("Linear SVM Results:\n")
print(classification_report(y_test, y_pred2))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred2))
```

Linear SVM Results:

	precision	recall	f1-score	support
negative	0.78	0.53	0.63	120
positive	0.87	0.95	0.91	400
accuracy			0.86	520
macro avg	0.82	0.74	0.77	520
weighted avg	0.85	0.86	0.85	520

Confusion Matrix:

```
[[ 63  57]
 [ 18 382]]
```

9. Model 3: Naive Bayes

We evaluate a Naive Bayes classifier for comparison with the previous models.

```
In [22]: from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, confusion_matrix

nb_model = Pipeline([
    ("tfidf", tfidf),
    ("clf", MultinomialNB())
])

nb_model.fit(X_train, y_train)
y_pred_nb = nb_model.predict(X_test)

print("Naive Bayes Results:\n")
print(classification_report(y_test, y_pred_nb))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_nb))
```

Naive Bayes Results:

	precision	recall	f1-score	support
negative	1.00	0.07	0.14	120
positive	0.78	1.00	0.88	400
accuracy			0.79	520
macro avg	0.89	0.54	0.51	520
weighted avg	0.83	0.79	0.71	520

Confusion Matrix:

```
[[ 9 111]
 [ 0 400]]
```

Confusion Matrix Visualization

The heatmaps below show the errors and correct predictions of each classifier.

```
In [20]: from sklearn.metrics import confusion_matrix
import numpy as np
import matplotlib.pyplot as plt

# Confusion matrix for Logistic Regression
cm_logreg = confusion_matrix(y_test, y_pred)
labels = sorted(y_test.unique()) # ['negative', 'positive']

fig, ax = plt.subplots(figsize=(4, 4))
```

```

im = ax.imshow(cm_logreg, cmap="Blues")

# Add colorbar
plt.colorbar(im)

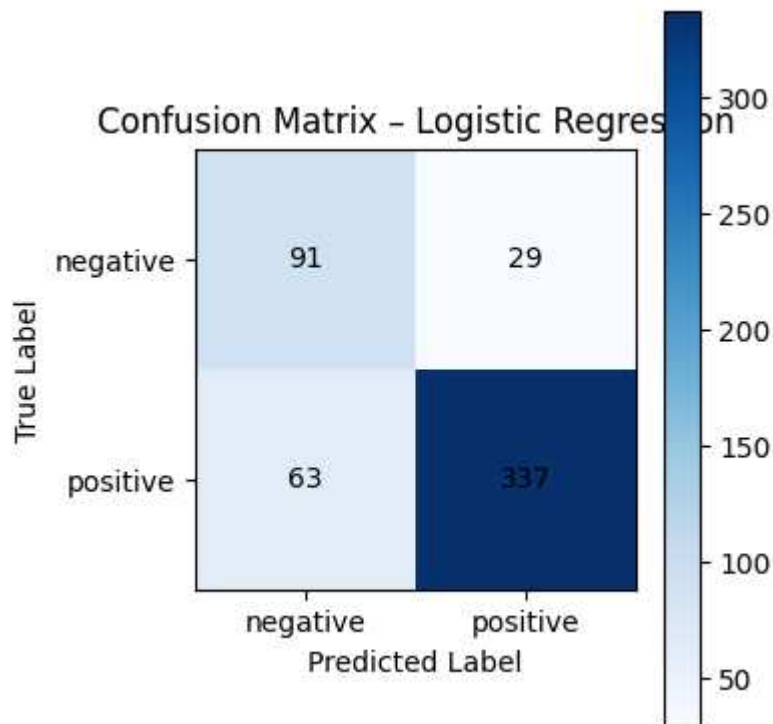
# Set tick labels
ax.set_xticks(np.arange(len(labels)))
ax.set_yticks(np.arange(len(labels)))
ax.set_xticklabels(labels)
ax.set_yticklabels(labels)

ax.set_xlabel("Predicted Label")
ax.set_ylabel("True Label")
ax.set_title("Confusion Matrix – Logistic Regression")

# Annotate cells with numbers
for i in range(cm_logreg.shape[0]):
    for j in range(cm_logreg.shape[1]):
        ax.text(j, i, cm_logreg[i, j], ha="center", va="center", color="black")

plt.tight_layout()
plt.show()

```



Confusion Matrix – Logistic Regression

```

In [21]: from sklearn.metrics import confusion_matrix
import numpy as np
import matplotlib.pyplot as plt

# Confusion matrix for Linear SVM
cm_svm = confusion_matrix(y_test, y_pred2)
labels = sorted(y_test.unique()) # ['negative', 'positive']

fig, ax = plt.subplots(figsize=(4, 4))
im = ax.imshow(cm_svm, cmap="Blues")

# Add colorbar
plt.colorbar(im)

# Set tick labels
ax.set_xticks(np.arange(len(labels)))
ax.set_yticks(np.arange(len(labels)))

```

```

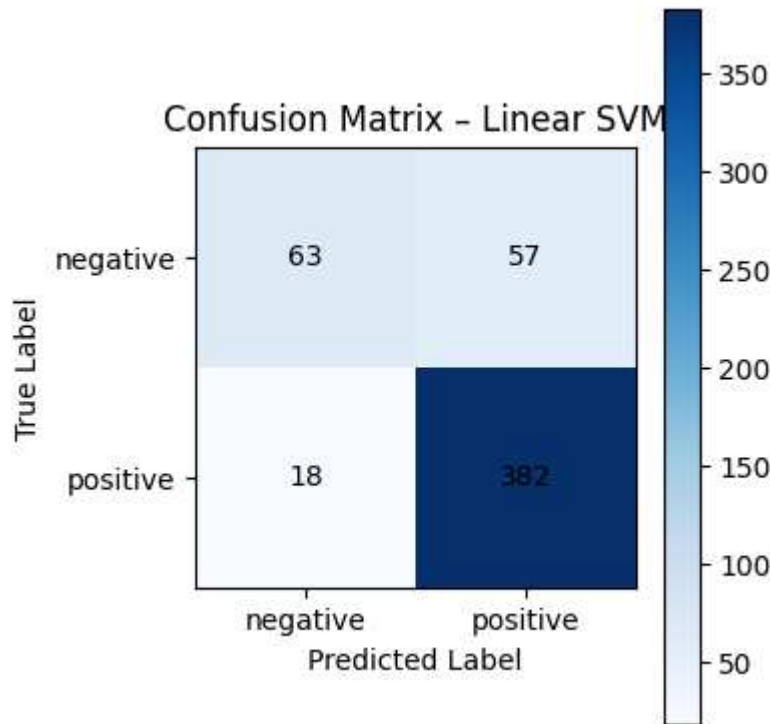
ax.set_xticklabels(labels)
ax.set_yticklabels(labels)

ax.set_xlabel("Predicted Label")
ax.set_ylabel("True Label")
ax.set_title("Confusion Matrix – Linear SVM")

# Annotate cells with numbers
for i in range(cm_svm.shape[0]):
    for j in range(cm_svm.shape[1]):
        ax.text(j, i, cm_svm[i, j], ha="center", va="center", color="black")

plt.tight_layout()
plt.show()

```



Confusion Matrix – Linear SVM

```

In [23]: from sklearn.metrics import confusion_matrix
import numpy as np
import matplotlib.pyplot as plt

cm_nb = confusion_matrix(y_test, y_pred_nb)
labels = sorted(y_test.unique())

fig, ax = plt.subplots(figsize=(4, 4))
im = ax.imshow(cm_nb, cmap="Blues")
plt.colorbar(im)

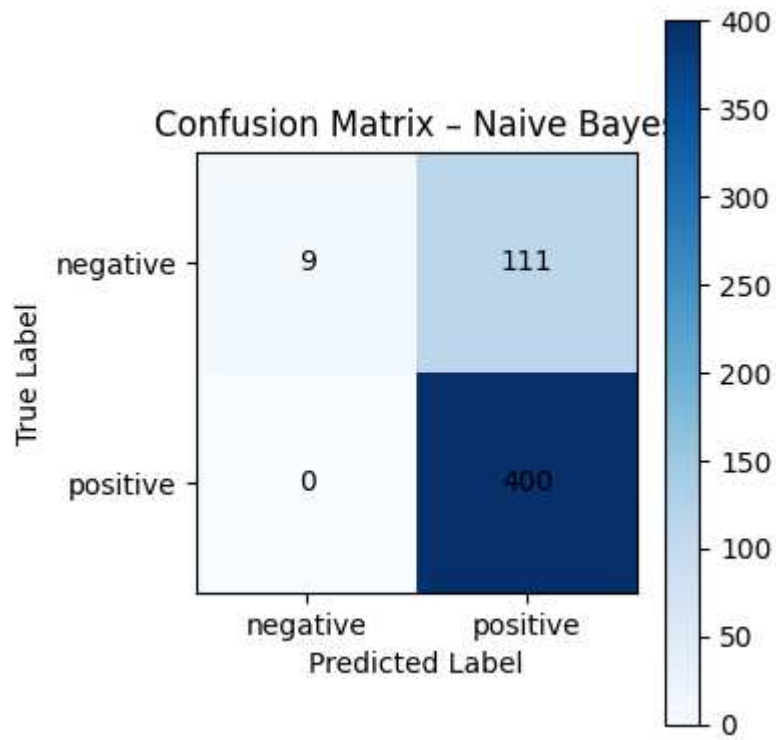
ax.set_xticks(np.arange(len(labels)))
ax.set_yticks(np.arange(len(labels)))
ax.set_xticklabels(labels)
ax.set_yticklabels(labels)

ax.set_xlabel("Predicted Label")
ax.set_ylabel("True Label")
ax.set_title("Confusion Matrix – Naive Bayes")

for i in range(cm_nb.shape[0]):
    for j in range(cm_nb.shape[1]):
        ax.text(j, i, cm_nb[i, j], ha="center", va="center", color="black")

```

```
plt.tight_layout()
plt.show()
```



Confusion Matrix – Naive Bayes

10. Conclusion

- The Linear SVM achieved the highest accuracy among all models.
- Logistic Regression also performed strongly and provided stable results.
- Naive Bayes was fast but less accurate due to feature independence assumptions.
- TF-IDF proved effective for sentiment classification.
- Future improvements may include:
 - Using word embeddings (Word2Vec, GloVe)
 - Fine-tuning transformer models like BERT
 - Performing hyper-parameter optimization