



Feature representation

Feature representation determines how textual data is converted into a format that machine learning models can process.

Types of feature representation

1. Bag of Words (BoW)

Each piece of text (document) is transformed into a list of numbers (a vector), where each number represents how often a specific word appears in that document.

How does this work?

- Document = any text (a sentence, a paragraph, a tweet, etc.).
- Vector = a numeric representation of that text.
- Word frequency = how many times each word appears in the document

Example:

Imagine we have these 3 documents (sentences):

- "The cat eats fish"
- "The dog eats meat"
- "The cat plays"

First, we build a vocabulary (a list of all unique words):

→ ["The", "cat", "eats", "fish", "dog", "meat", "plays"]

Now, we create a vector for each document based on how many times each word appears:

- Doc 1 → [1, 1, 1, 1, 0, 0, 0]
- Doc 2 → [1, 0, 1, 0, 1, 1, 0]
- Doc 3 → [1, 1, 0, 0, 0, 0, 1]

2. TF-IDF (Term Frequency–Inverse Document Frequency)

Weights terms based on how common they are across documents. Helps reduce the impact of frequent but unimportant words (e.g., "the", "is").

Formula: $TF\text{-}IDF = TF(\text{term}) * \log(N / DF(\text{term}))$

3. N-grams (Unigrams, Bigrams, Trigrams)

Considers sequences of n words to capture local word order and context.

Import necessary libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

import re
import nltk
import spacy
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer

# Download NLTK resources (only first time)
```

```

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\compumarts\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\compumarts\AppData\Roaming\nltk_data...
[nltk_data] Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data] C:\Users\compumarts\AppData\Roaming\nltk_data...
[nltk_data] Package wordnet is already up-to-date!

True

```

Reading dataset

```

df = pd.read_csv("Sentiment.csv",encoding="ISO-8859-1")
df.head()

```

	text	sentiment
0	Sooo SAD I will miss you here in San Diego!!!	negative
1	my boss is bullying me...	negative
2	what interview! leave me alone	negative
3	Sons of ****, why couldn't they put them on t...	negative
4	2am feedings for the baby are fun when he is a...	positive

```

df['sentiment'].value_counts()

sentiment
positive    8582
negative    7781
neutral     3194
Name: count, dtype: int64

```

Cleaning dataset

```

stemmer = PorterStemmer()

stop_words = set(stopwords.words('english'))
def preprocess_text(text):
    text = text.lower()
    # 2. Remove URLs

    text = re.sub(r'http\S+|www.\S+', '', text)
    # 3. Remove HTML tags
    text = re.sub(r'<.*?>', '', text)

```

```

# 4. Remove Special Characters, Numbers, Punctuation
text = re.sub(r'^a-zA-Z\s', '', text)
# 5. Tokenization
tokens = word_tokenize(text)
# 6. Remove Stop Words
tokens = [word for word in tokens if word not in stop_words]
# 7. Stemming
stemmed_tokens = [stemmer.stem(word) for word in tokens]
# 8. Final clean text reconstruction (optional)
clean_text = ' '.join(stemmed_tokens)

return clean_text

```

```

df["clean_text"] = df["text"].apply(preprocess_text)
df = df.dropna()

```

```
df['clean_text']
```

```

0          sooo sad miss san diego
1                          boss bulli
2          interview leav alon
3          son couldnt put releas alreadi bought
4          feed babi fun smile coo
...
19552          enjoy ur night
19553  wish could come see u denver husband lost job ...
19554  ive wonder rake client made clear net dont for...
19555  yay good enjoy break probabl need hectic weeke...
19556          worth
Name: clean_text, Length: 19557, dtype: object

```

Define features and target variable

```

X = df['clean_text'].values
y = df['sentiment'].values

```

Bag of Words

```

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(X)

```

Splitting data

```

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split( X, y,
test_size=0.30, random_state=42, stratify=y)

```

Evaluation models

```
def evaluate_model(model_name, y_true, y_pred):  
  
    # Calculate metrics  
    accuracy = accuracy_score(y_true, y_pred)  
    precision = precision_score(y_true, y_pred, average='weighted')  
    recall = recall_score(y_true, y_pred, average='weighted')  
    f1 = f1_score(y_true, y_pred, average='weighted')  
    cm = confusion_matrix(y_true, y_pred)  
  
    # Create a report  
    report = classification_report(y_true, y_pred)  
  
    # Output results  
    metrics = {  
        'Model Name': model_name,  
        'Accuracy': accuracy,  
        'Precision': precision,  
        'Recall': recall,  
        'F1 Score': f1,  
  
        'Classification Report': report  
    }  
  
    # Plot Confusion Matrix  
    plt.figure(figsize=(4, 4))  
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,  
                xticklabels=np.unique(y_true),  
                yticklabels=np.unique(y_true))  
    plt.title(f'Confusion Matrix for {model_name}')  
    plt.xlabel('Predicted Label')  
    plt.ylabel('True Label')  
    plt.show()  
  
    return metrics
```

Random Forest Classifier

```
# Define the Random Forest Classifier  
rf_classifier = RandomForestClassifier(random_state=42)  
  
# Define the hyperparameters for grid search  
param_grid = {
```

```

        'n_estimators': [100, 200, 300],
        'max_depth': [None, 10, 20, 30],
    }

    # Initialize GridSearchCV
    grid_search = GridSearchCV(estimator=rf_classifier,
                               param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)

    # Fit the grid search to the data
    grid_search.fit(X_train, y_train)

    # Get the best estimator from grid search
    best_rf_classifier = grid_search.best_estimator_

    # Make predictions using the best model
    y_pred = best_rf_classifier.predict(X_test)

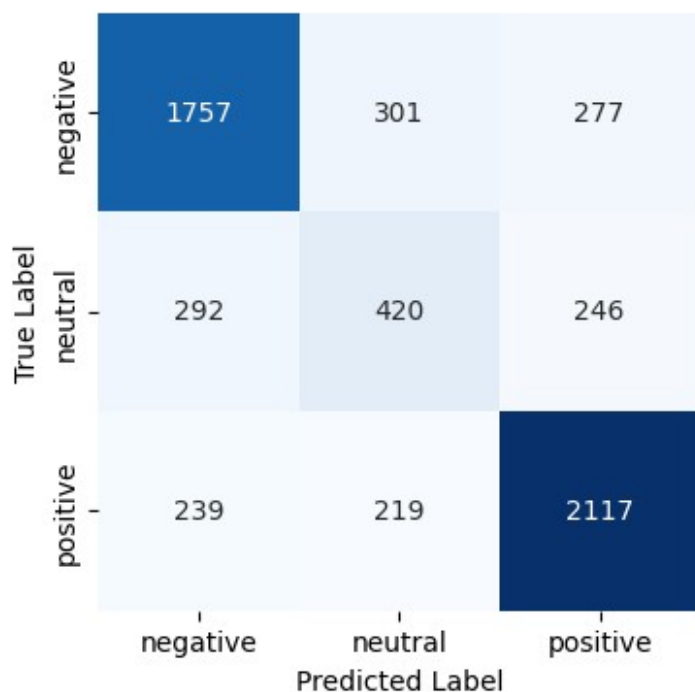
    # Evaluate the model
    evaluation_results = evaluate_model('RandomForestClassifier', y_test,
                                         y_pred)

    # Print the evaluation results
    for key, value in evaluation_results.items():
        if key == 'Classification Report':
            print(value) # Print report separately for better readability
        else:
            print(f"{key}: {value:.4f}" if isinstance(value, float) else
                  f"{key}: \n{value}")

    # Print the best parameters found by grid search
    print("\nBest hyperparameters found by GridSearchCV:")
    print(grid_search.best_params_)

```

Confusion Matrix for RandomForestClassifier



Model Name:
RandomForestClassifier
Accuracy: 0.7318
Precision: 0.7304
Recall: 0.7318
F1 Score: 0.7310

	precision	recall	f1-score	support
negative	0.77	0.75	0.76	2335
neutral	0.45	0.44	0.44	958
positive	0.80	0.82	0.81	2575
accuracy			0.73	5868
macro avg	0.67	0.67	0.67	5868
weighted avg	0.73	0.73	0.73	5868

Best hyperparameters found by GridSearchCV:
{'max_depth': None}

LogisticRegression

```
# Define the Logistic Regression model
LR = LogisticRegression(random_state=42)

# Define the hyperparameters for grid search
```

```

param_grid = {
    'solver': ['liblinear', 'lbfgs'],      # Solver type
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=LR, param_grid=param_grid, cv=5,
scoring='accuracy', n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best estimator from grid search
best_LR = grid_search.best_estimator_

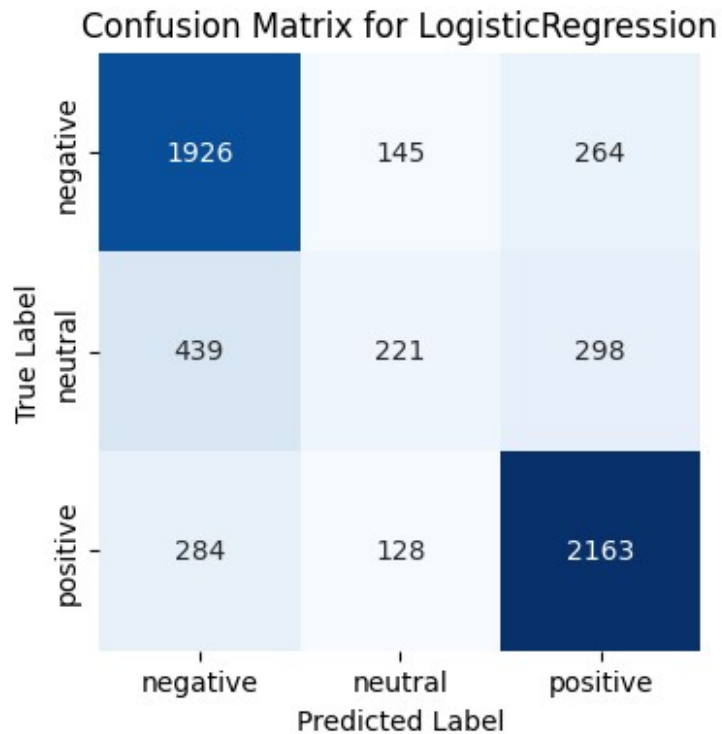
# Make predictions using the best model
y_pred = best_LR.predict(X_test)

# Evaluate the model
evaluation_results = evaluate_model('LogisticRegression', y_test,
y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```



Model Name:
LogisticRegression
Accuracy: 0.7345
Precision: 0.7107
Recall: 0.7345
F1 Score: 0.7154

	precision	recall	f1-score	support
negative	0.73	0.82	0.77	2335
neutral	0.45	0.23	0.30	958
positive	0.79	0.84	0.82	2575
accuracy			0.73	5868
macro avg	0.66	0.63	0.63	5868
weighted avg	0.71	0.73	0.72	5868

Best hyperparameters found by GridSearchCV:
{'solver': 'liblinear'}

DecisionTreeClassifier

```
# Define the Decision Tree model
DT = DecisionTreeClassifier(random_state=42)

# Define the hyperparameters for grid search
param_grid = {
```

```

        'criterion': ['gini', 'entropy'],           # The function to
        measure the quality of a split
        'max_depth': [None, 10, 20, 30],          # The maximum depth
        of the tree
    }

    # Initialize GridSearchCV
    grid_search = GridSearchCV(estimator=DT, param_grid=param_grid, cv=5,
                               scoring='accuracy', n_jobs=-1)

    # Fit the grid search to the data
    grid_search.fit(X_train, y_train)

    # Get the best estimator from grid search
    best_DT = grid_search.best_estimator_

    # Make predictions using the best model
    y_pred = best_DT.predict(X_test)

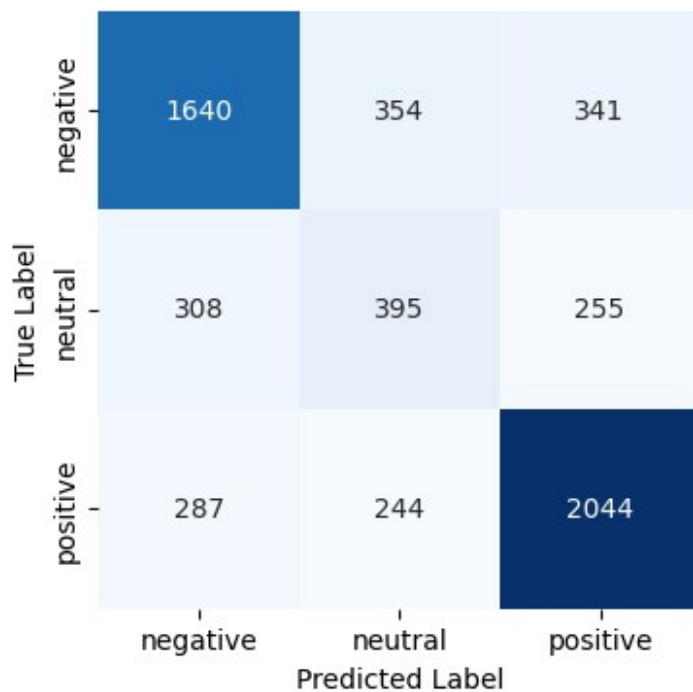
    # Evaluate the model
    evaluation_results = evaluate_model('DecisionTreeClassifier', y_test,
                                        y_pred)

    # Print the evaluation results
    for key, value in evaluation_results.items():
        if key == 'Classification Report':
            print(value) # Print report separately for better readability
        else:
            print(f"{key}: {value:.4f}" if isinstance(value, float) else
                  f"{key}: \n{value}")

    # Print the best hyperparameters found by grid search
    print("\nBest hyperparameters found by GridSearchCV:")
    print(grid_search.best_params_)

```

Confusion Matrix for DecisionTreeClassifier



Model Name:
DecisionTreeClassifier
Accuracy: 0.6951
Precision: 0.6967
Recall: 0.6951
F1 Score: 0.6957

	precision	recall	f1-score	support
negative	0.73	0.70	0.72	2335
neutral	0.40	0.41	0.40	958
positive	0.77	0.79	0.78	2575
accuracy			0.70	5868
macro avg	0.64	0.64	0.64	5868
weighted avg	0.70	0.70	0.70	5868

Best hyperparameters found by GridSearchCV:
{'criterion': 'entropy', 'max_depth': None}

Support Vector Machine

```
SVM = SVC(random_state=42)

# Define the hyperparameters for grid search
param_grid = {
```

```

        'kernel': ['linear', 'rbf', 'poly'],      # Kernel type
    }

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=SVM, param_grid=param_grid, cv=5,
scoring='accuracy', n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best estimator from grid search
best_SVM = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_SVM.predict(X_test)

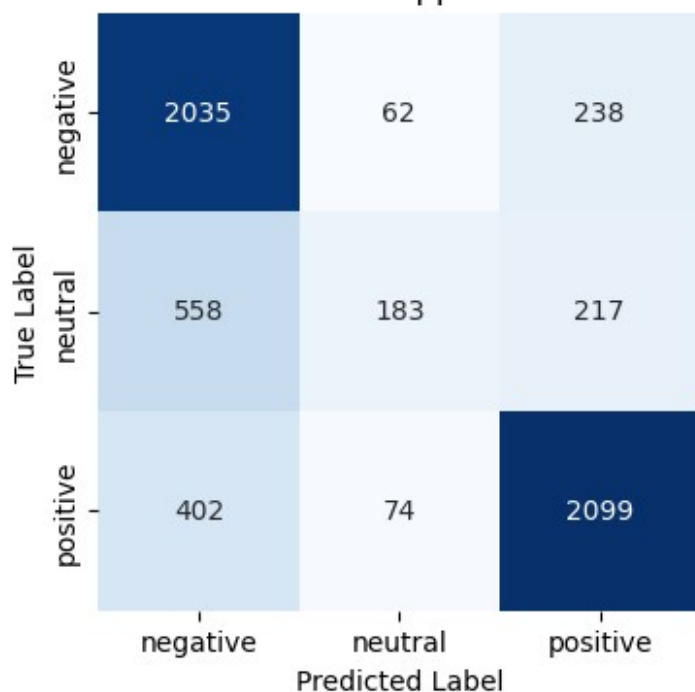
# Evaluate the model
evaluation_results = evaluate_model('Support Vector Machine', y_test,
y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

Confusion Matrix for Support Vector Machine



Model Name:
Support Vector Machine
Accuracy: 0.7357
Precision: 0.7247
Recall: 0.7357
F1 Score: 0.7098

	precision	recall	f1-score	support
negative	0.68	0.87	0.76	2335
neutral	0.57	0.19	0.29	958
positive	0.82	0.82	0.82	2575
accuracy			0.74	5868
macro avg	0.69	0.63	0.62	5868
weighted avg	0.72	0.74	0.71	5868

Best hyperparameters found by GridSearchCV:
{'kernel': 'rbf'}

KNeighborsClassifier

```
# Define the K-Nearest Neighbors model
knn = KNeighborsClassifier()
```

```

# Define the hyperparameters for grid search
param_grid = {
    'n_neighbors': [1, 3, 5, 7, 9],          # Number of neighbors
    # to use
    'weights': ['uniform', 'distance'],      # Weight function used
    # in prediction
    'metric': ['euclidean', 'manhattan'],    # Distance metric
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'] #
    # Algorithm to compute the nearest neighbors
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5,
    scoring='accuracy', n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best estimator from grid search
best_knn = grid_search.best_estimator_

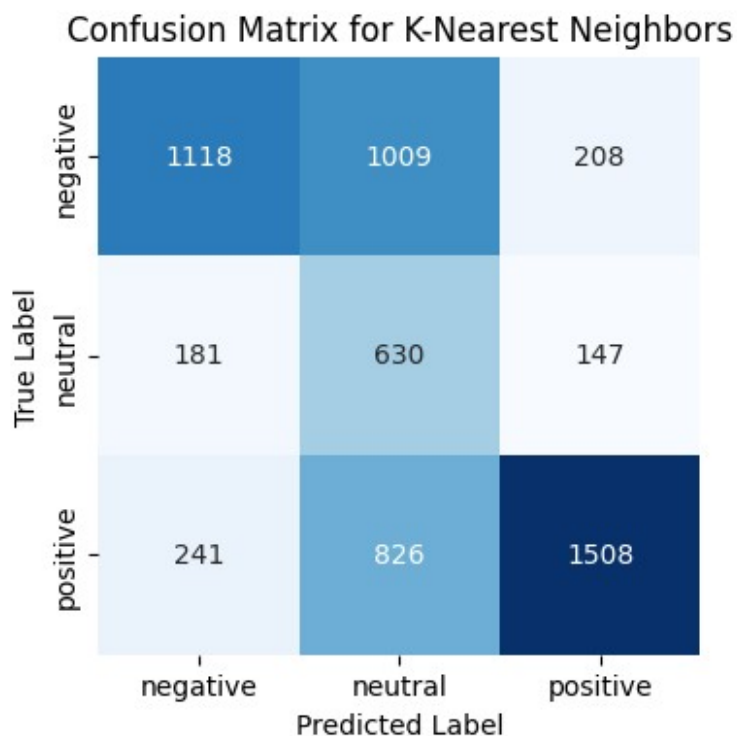
# Make predictions using the best model
y_pred = best_knn.predict(X_test)

# Evaluate the model
evaluation_results = evaluate_model('K-Nearest Neighbors', y_test,
    y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
            f"{key}: \n{value}")

# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```



Model Name:
K-Nearest Neighbors
Accuracy: 0.5549
Precision: 0.6858
Recall: 0.5549
F1 Score: 0.5879

	precision	recall	f1-score	support
negative	0.73	0.48	0.58	2335
neutral	0.26	0.66	0.37	958
positive	0.81	0.59	0.68	2575
accuracy			0.55	5868
macro avg	0.60	0.57	0.54	5868
weighted avg	0.69	0.55	0.59	5868

Best hyperparameters found by GridSearchCV:
{'algorithm': 'auto', 'metric': 'manhattan', 'n_neighbors': 5,
'weights': 'distance'}