

```
from IPython.display import Image, display
display(Image(filename='Lecture 6.png'))
```



What is NLP (Natural Language Processing)?

Natural Language Processing (NLP) is a field of Artificial Intelligence (AI) that focuses on enabling computers to understand, interpret, generate, and respond to human languages in a way that is both meaningful and useful.

Different Applications of NLP Classification

- Sentiment Analysis (Emotion Classification)
- Spam Detection (Email & Message Classification)
- Topic Classification (News and Document Categorization)

Main steps of developing NLP for classification

- Data Collection
- Data Preprocessing
- Feature Extraction
- Model Selection and Training
- Evaluation

The main types of data preprocessing

1. Text Cleaning: Preprocessing the raw text to remove noise and irrelevant content.
 - Lowercasing: Convert all text to lowercase to ensure uniformity. Example: "Hello World" → "hello world"

- Removing Punctuation: Eliminating symbols like .!/?@#. Example: "hello, world!" → "hello world"
 - Removing Special Characters: Remove non-alphanumeric characters. Example: "I love #AI & NLP" → "I love AI NLP"
 - Removing Numbers (optional): If numbers are not meaningful for the task. Example: "Today is 13th" → "Today is"
 - Removing URLs, Emails, Mentions: For social media, forums, emails. Example: "Check <http://example.com>" → "Check"
 - Removing HTML tags:
2. Tokenization: Splitting text into units: words, subwords, or sentences.
 - Word Tokenization: "I love NLP." → ["I", "love", "NLP", "."]
 - Sentence Tokenization: "I love NLP. It is powerful." → ["I love NLP.", "It is powerful."]
 3. Normalization: Reducing text to a canonical form.
 - Stemming: Reduce words to root form (may not be meaningful). Example: "playing", "played", "plays" → "play" Tool: PorterStemmer, SnowballStemmer
 - Lemmatization: Reduce words to dictionary form using context. Example: "am", "are", "is" → "be" Tool: WordNetLemmatizer
 4. Stop Words Removal: Eliminate common but insignificant words (e.g., "the", "is", "in").

Example: "This is a book" → "book"

```
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

```
stemmer = PorterStemmer()
```

```
# Stop words list
```

```
stop_words = set(stopwords.words('english'))
```

```
def preprocess_text(text):
    print(f"\nOriginal Text: {text}")

    # 1. Lowercasing
    text = text.lower()
    print(f"Lowercased Text: {text}")

    # 2. Remove URLs
    text = re.sub(r'http\S+|www.\S+', '', text)
    print(f"Removed URLs: {text}")

    # 3. Remove HTML tags
    text = re.sub(r'<.*?>', '', text)
    print(f"Removed HTML Tags: {text}")

    # 4. Remove Special Characters, Numbers, Punctuation
    text = re.sub(r'^a-zA-Z\s]', '', text)
    print(f"Removed Special Characters: {text}")

    # 5. Tokenization
    tokens = word_tokenize(text)
    print(f"Tokens: {tokens}")

    # 6. Remove Stop Words
    tokens = [word for word in tokens if word not in stop_words]
    print(f"Removed Stopwords: {tokens}")

    # 7. Stemming
    stemmed_tokens = [stemmer.stem(word) for word in tokens]
    print(f"Stemmed Tokens: {stemmed_tokens}")

    # 8. Final clean text reconstruction (optional)
    clean_text = ' '.join(stemmed_tokens)
    print(f"Final Preprocessed Text: {clean_text}")

    return clean_text
```

```
sample_text = """
Hello! This is a test document. Visit https://example.com for more
info.
I bought 5 products from Amazon. Let's meet at 5:00 PM. I can't wait
to see you! 😊
"""
```

```
final_text = preprocess_text(sample_text)
```

Original Text:

Hello! This is a test document. Visit https://example.com for more info.
I bought 5 products from Amazon. Let's meet at 5:00 PM. I can't wait to see you! 😊

Lowercased Text:

hello! this is a test document. visit https://example.com for more info.
i bought 5 products from amazon. let's meet at 5:00 pm. i can't wait to see you! 😊

Removed URLs:

hello! this is a test document. visit for more info.
i bought 5 products from amazon. let's meet at 5:00 pm. i can't wait to see you! 😊

Removed HTML Tags:

hello! this is a test document. visit for more info.
i bought 5 products from amazon. let's meet at 5:00 pm. i can't wait to see you! 😊

Removed Special Characters:

hello this is a test document visit for more info
i bought products from amazon lets meet at pm i cant wait to see you

Tokens: ['hello', 'this', 'is', 'a', 'test', 'document', 'visit', 'for', 'more', 'info', 'i', 'bought', 'products', 'from', 'amazon', 'lets', 'meet', 'at', 'pm', 'i', 'cant', 'wait', 'to', 'see', 'you']

Removed Stopwords: ['hello', 'test', 'document', 'visit', 'info', 'bought', 'products', 'amazon', 'lets', 'meet', 'pm', 'cant', 'wait', 'see']

Stemmed Tokens: ['hello', 'test', 'document', 'visit', 'info', 'bought', 'product', 'amazon', 'let', 'meet', 'pm', 'cant', 'wait', 'see']

```
Final Preprocessed Text: hello test document visit info bought product
amazon let meet pm cant wait see
```

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
```

Reading dataset

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

	text	sentiment
0	Last session of the day http://twitpic.com/67ezh	neutral
1	Shanghai is also really exciting (precisely -...	positive
2	Recession hit Veronique Branquinho, she has to...	negative
3	happy bday!	positive
4	http://twitpic.com/4w75p - I like it!!	positive

```
df['sentiment'].value_counts()

sentiment
neutral    1430
positive   1103
negative   1001
Name: count, dtype: int64

stemmer = PorterStemmer()

# Stop words list
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
```



```
3530     alon old hous thank net keep aliv kick whoever...
3531     know mean littl dog sink depress want move som...
3532             sutra next youtub video gon na love video
3533             omgssh ang cute ng bbi
Name: clean_text, Length: 3534, 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

```

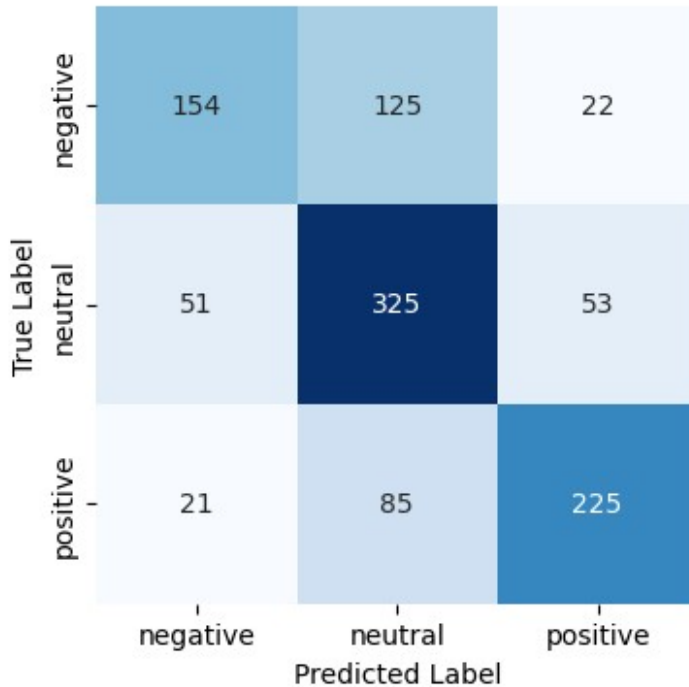
```

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



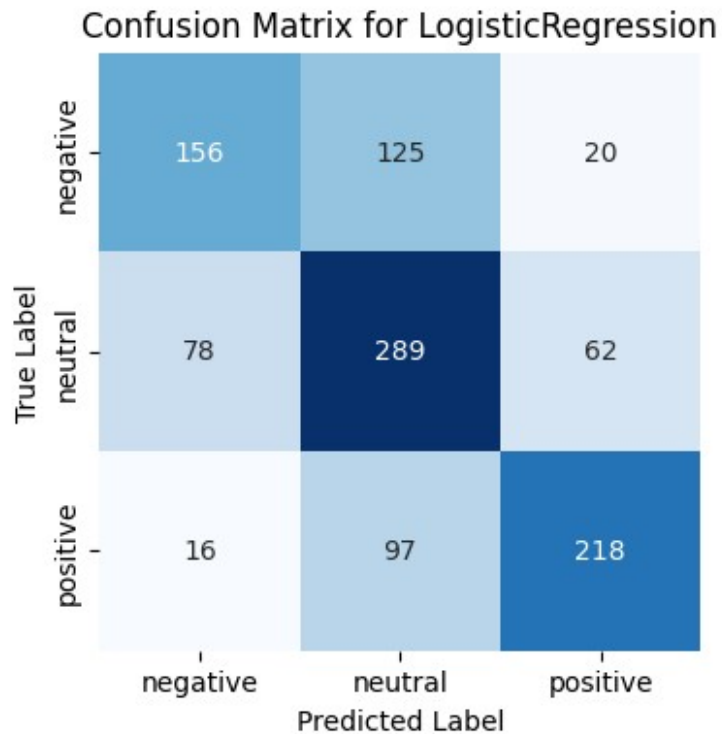
Model Name:
 RandomForestClassifier
 Accuracy: 0.6635
 Precision: 0.6729
 Recall: 0.6635
 F1 Score: 0.6609

	precision	recall	f1-score	support
negative	0.68	0.51	0.58	301
neutral	0.61	0.76	0.67	429
positive	0.75	0.68	0.71	331
accuracy			0.66	1061
macro avg	0.68	0.65	0.66	1061
weighted avg	0.67	0.66	0.66	1061

```
Best hyperparameters found by GridSearchCV:  
{'max_depth': None, 'n_estimators': 200}
```

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: 0.6249
Precision: 0.6324
Recall: 0.6249
F1 Score: 0.6248

	precision	recall	f1-score	support
negative	0.62	0.52	0.57	301
neutral	0.57	0.67	0.61	429
positive	0.73	0.66	0.69	331
accuracy			0.62	1061
macro avg	0.64	0.62	0.62	1061
weighted avg	0.63	0.62	0.62	1061

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

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)

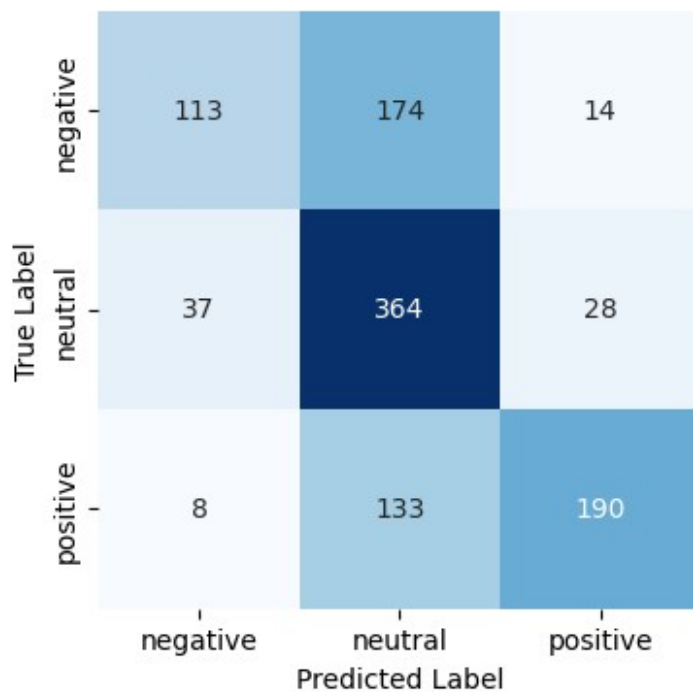
    # Evaluate the model
    evaluation_results = evaluate_model('DecisionTreeClassifier', y_test,
                                        y_pred)

    # 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: 0.6287
Precision: 0.6777
Recall: 0.6287
F1 Score: 0.6178

	precision	recall	f1-score	support
negative	0.72	0.38	0.49	301
neutral	0.54	0.85	0.66	429
positive	0.82	0.57	0.67	331
accuracy			0.63	1061
macro avg	0.69	0.60	0.61	1061
weighted avg	0.68	0.63	0.62	1061

Best hyperparameters found by GridSearchCV:
{'criterion': 'gini', 'max_depth': 30}

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)

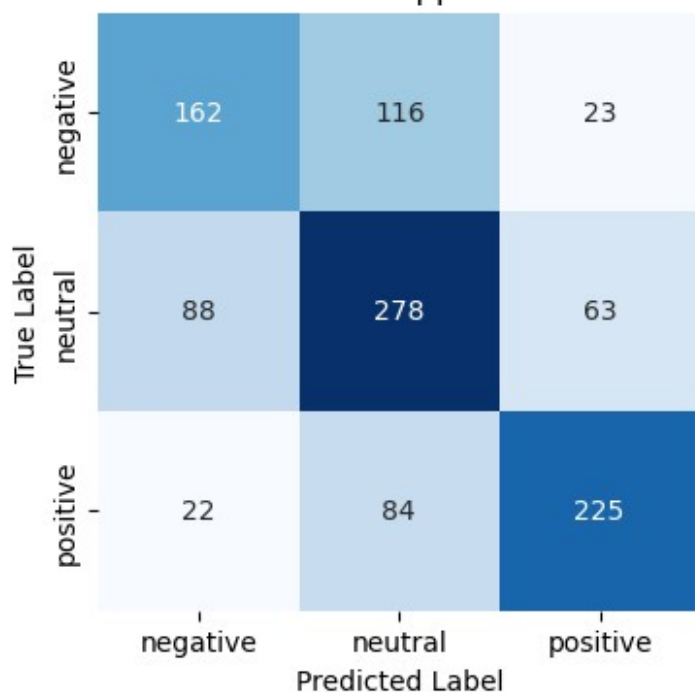
# Evaluate the model
evaluation_results = evaluate_model('Support Vector Machine', y_test,
y_pred)

# 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: 0.6268
Precision: 0.6298
Recall: 0.6268
F1 Score: 0.6269

	precision	recall	f1-score	support
negative	0.60	0.54	0.57	301
neutral	0.58	0.65	0.61	429
positive	0.72	0.68	0.70	331
accuracy			0.63	1061
macro avg	0.63	0.62	0.63	1061
weighted avg	0.63	0.63	0.63	1061

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

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)

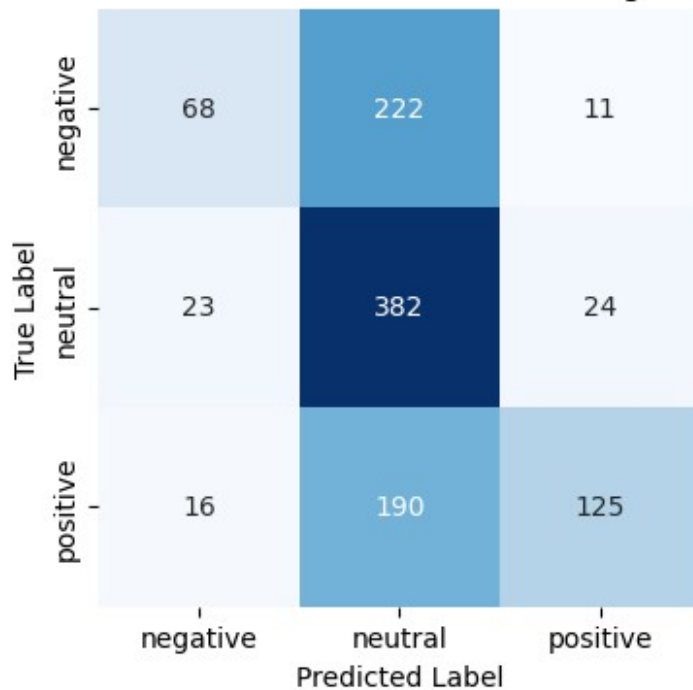
# Evaluate the model
evaluation_results = evaluate_model('K-Nearest Neighbors', y_test,
y_pred)

# 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_)

```

Confusion Matrix for K-Nearest Neighbors



Model Name:
K-Nearest Neighbors
Accuracy: 0.5419
Precision: 0.6185
Recall: 0.5419
F1 Score: 0.5060

	precision	recall	f1-score	support
negative	0.64	0.23	0.33	301
neutral	0.48	0.89	0.62	429
positive	0.78	0.38	0.51	331
accuracy			0.54	1061
macro avg	0.63	0.50	0.49	1061
weighted avg	0.62	0.54	0.51	1061

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