```
!pip install scikit-learn matplotlib seaborn wordcloud clean-text joblib -q

print("Setup complete!")
```

```
Setup complete!
```

```
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import (
    train_test_split,
    cross_val_score,
    HalvingGridSearchCV
)
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    ConfusionMatrixDisplay,
    roc_auc_score,
    roc_curve,
    average_precision_score,
    precision_recall_curve
)

import joblib
from cleantext import clean

# import warnings
# warnings.filterwarnings("ignore")

sns.set_theme(style="whitegrid")
```

```
WARNING:root:Since the GPL-licensed package `unidecode` is not installed,
```

```
def clean_text(text):
    """
    Minimal cleaning for IMDB dataset:
    - Remove HTML tags
    - Remove line breaks
    - Remove extra spaces
    """
    # Remove HTML tags
    text = re.sub(r'<.*?>', ' ', text)

    # Remove line breaks and tabs
    text = text.replace("\n", " ").replace("\r", " ").replace("\t", " ")
```

```
    # Normalize multiple spaces to single
    text = re.sub(r'\s+', ' ', text).strip()

    return text

# Load data
url_sent = "https://raw.githubusercontent.com/laxmimerit/All-CSV-ML-Data-Files-
df_sent = pd.read_csv(url_sent)
print(df_sent.head())
print("\nClass distribution:\n", df_sent['sentiment'].value_counts(normalize=Tr

df_sent['clean_text'] = df_sent['review'].apply(clean_text)
```

```
                                              review sentiment
0  One of the other reviewers has mentioned that ...  positive
1  A wonderful little production. <br /><br />The...  positive
2  I thought this was a wonderful way to spend ti...  positive
3  Basically there's a family where a little boy ...  negative
4  Petter Mattei's "Love in the Time of Money" is...  positive

Class distribution:
 sentiment
positive    0.5
negative    0.5
Name: proportion, dtype: float64
```

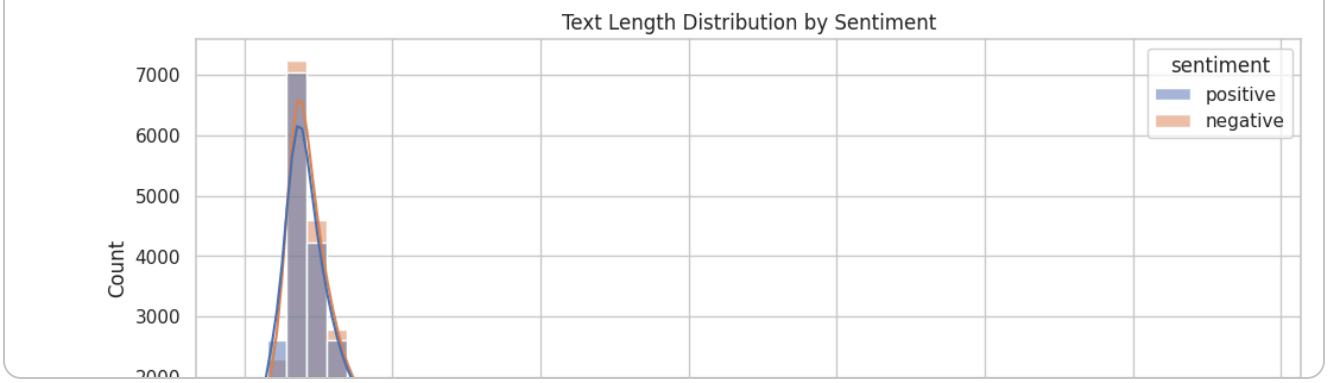
```
df_sent['clean_text'].head()
```

```
0    One of the other reviewers has mentioned that ...
1    A wonderful little production. The filming tec...
2    I thought this was a wonderful way to spend ti...
3    Basically there's a family where a little boy ...
4    Petter Mattei's "Love in the Time of Money" is...
Name: clean_text, dtype: str
```

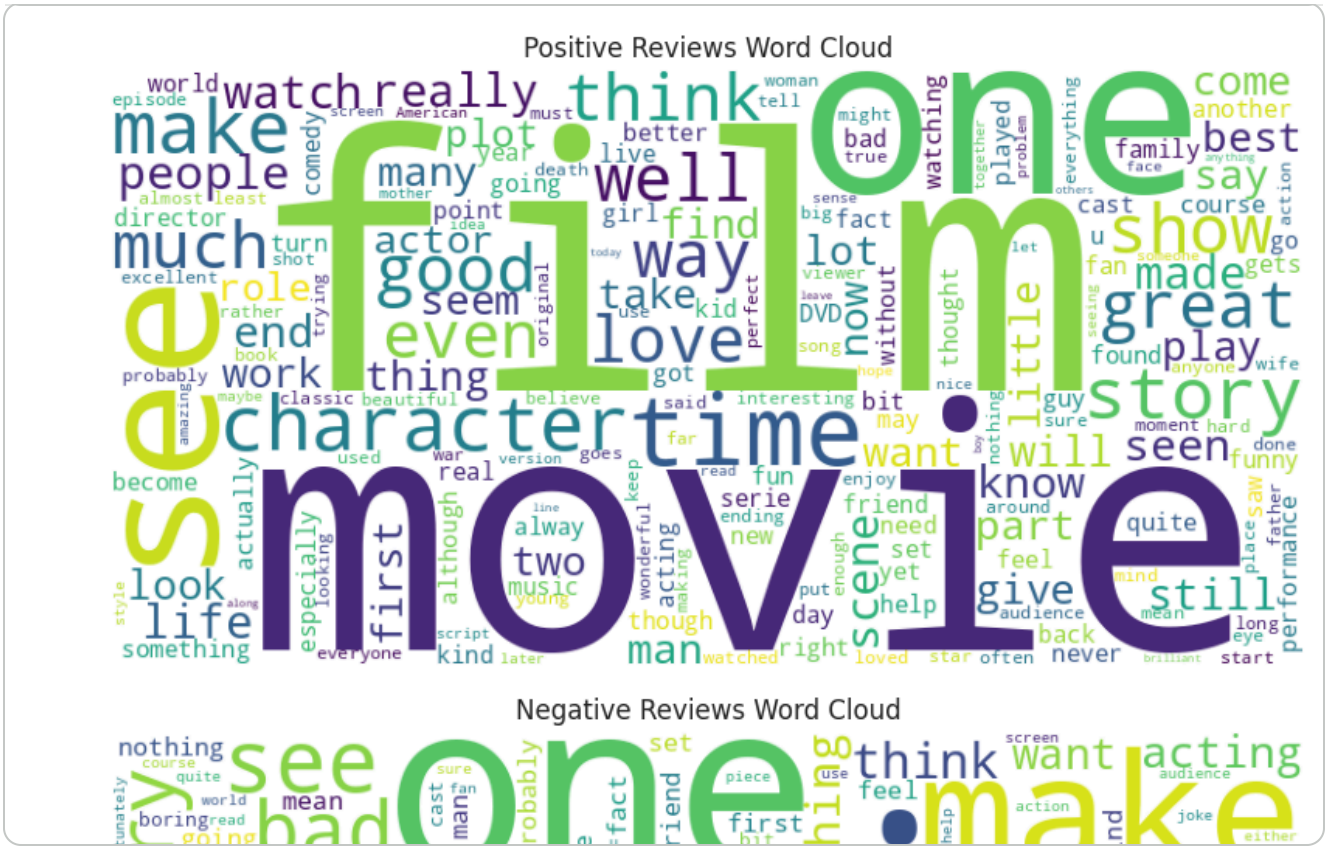
```
# EDA & Visualizations
```

```
# Class balance
plt.figure(figsize=(6,4))
sns.countplot(data=df_sent, x='sentiment')
plt.title("Sentiment Class Distribution")
plt.show()
```


```
# Text length
df_sent['text_length'] = df_sent['clean_text'].str.len()
plt.figure(figsize=(12,5))
sns.histplot(df_sent, x='text_length', hue='sentiment', bins=50, kde=True)
plt.title("Text Length Distribution by Sentiment")
plt.show()
```




```
# Wordclouds
def generate_wordcloud(texts, title):
    wc = WordCloud(width=800, height=400, background_color='white').generate('
    plt.figure(figsize=(10,5))
    plt.imshow(wc)
    plt.axis('off')
    plt.title(title)
    plt.show()

pos_texts = df_sent[df_sent['sentiment']=='positive']['clean_text']
neg_texts = df_sent[df_sent['sentiment']=='negative']['clean_text']
generate_wordcloud(pos_texts, "Positive Reviews Word Cloud")
generate_wordcloud(neg_texts, "Negative Reviews Word Cloud")
```



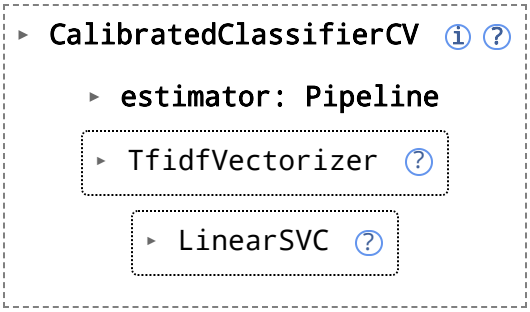

```
# Model Pipeline + Training (TF-IDF + LinearSVC)
```

```
X = df_sent['clean_text']
y = df_sent['sentiment'].map({'positive': 1, 'negative': 0})

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randor

pipe_sent = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=30000, ngram_range=(1,2), min_df=5,
    ('clf', LinearSVC(C=0.8, dual=False, class_weight='balanced', max_iter=200(
])

# Cross-validation baseline
cv_scores = cross_val_score(pipe_sent, X_train, y_train, cv=5, scoring='f1')
print(f"CV F1: {cv_scores.mean():.4f} ± {cv_scores.std():.4f}")

# Hyperparameter tuning (efficient HalvingGridSearchCV)
param_grid = {
    'tfidf__ngram_range': [(1,1), (1,2)],
    'tfidf__max_features': [20000, 30000],
    'clf__C': [0.5, 0.8, 1.2]
}

search = HalvingGridSearchCV(pipe_sent, param_grid, cv=3, factor=2, scoring='f
search.fit(X_train, y_train)
print("Best params:", search.best_params_)

best_pipe = search.best_estimator_
```

```
CV F1: 0.9069 ± 0.0021
Best params: {'clf__C': 0.5, 'tfidf__max_features': 30000, 'tfidf__ngram_
```

```
# Calibration, Evaluation & Visualizations
```

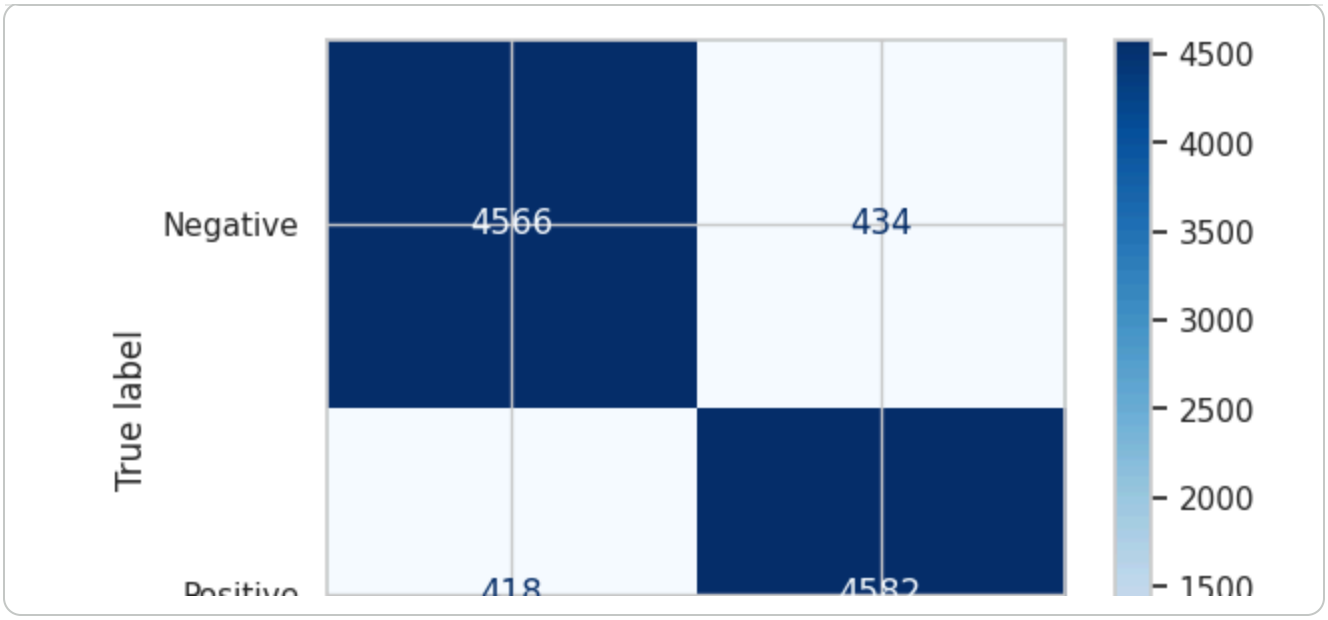
```
# Calibrate (isotonic for better probability calibration)
calibrated = CalibratedClassifierCV(best_pipe, method='isotonic')
calibrated.fit(X_train, y_train)
```



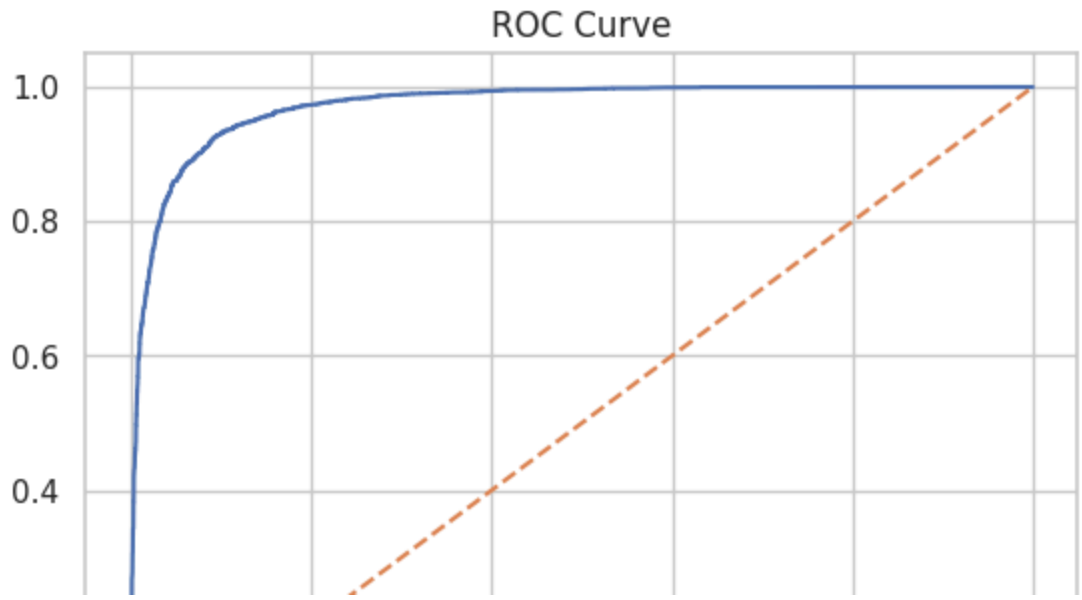

```
# Predict & metrics
y_pred = calibrated.predict(X_test)
y_proba = calibrated.predict_proba(X_test)[:, 1]

print(classification_report(y_test, y_pred, digits=4))
print(f"ROC-AUC: {roc_auc_score(y_test, y_proba):.4f}")
```

```
              precision    recall  f1-score   support

           0     0.9161    0.9132    0.9147      5000
           1     0.9135    0.9164    0.9149      5000

    accuracy                         0.9148     10000
   macro avg     0.9148    0.9148    0.9148     10000
weighted avg     0.9148    0.9148    0.9148     10000

ROC-AUC: 0.9728
```

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(cm, display_labels=['Negative', 'Positive'])
disp.plot(cmap='Blues')
plt.show()
```




```
# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_proba)
plt.plot(fpr, tpr, label=f'AUC = {roc_auc_score(y_test, y_proba):.4f}')
plt.plot([0,1],[0,1],'--')
plt.title("ROC Curve")
plt.legend()
plt.show()
```




```
# Save model
joblib.dump(calibrated, "sentiment_model_calibrated.joblib")
```

```
['sentiment_model_calibrated.joblib']
```

```
import gradio as gr
import joblib

model = joblib.load("/content/sentiment_model_calibrated.joblib")
```

```
def predict(text):
    cleaned = professional_clean(text)
    prob = model.predict_proba([])[0][1]
    label = "Positive" if prob > 0.5 else "Negative"
    return f"{label} ({prob:.1%})"

demo = gr.Interface(
    fn=predict,
    inputs=gr.Textbox(label="Enter review or sentence"),
    outputs=gr.Textbox(label="Prediction"),
    title="Movie Review Sentiment Checker",
    description="Type any sentence and see if it's predicted positive or nega
)

demo.launch()
```

```
It looks like you are running Gradio on a hosted Jupyter notebook, which

Colab notebook detected. To show errors in colab notebook, set debug=True
* Running on public URL: https://1761084ae5bb16c267.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upg
```