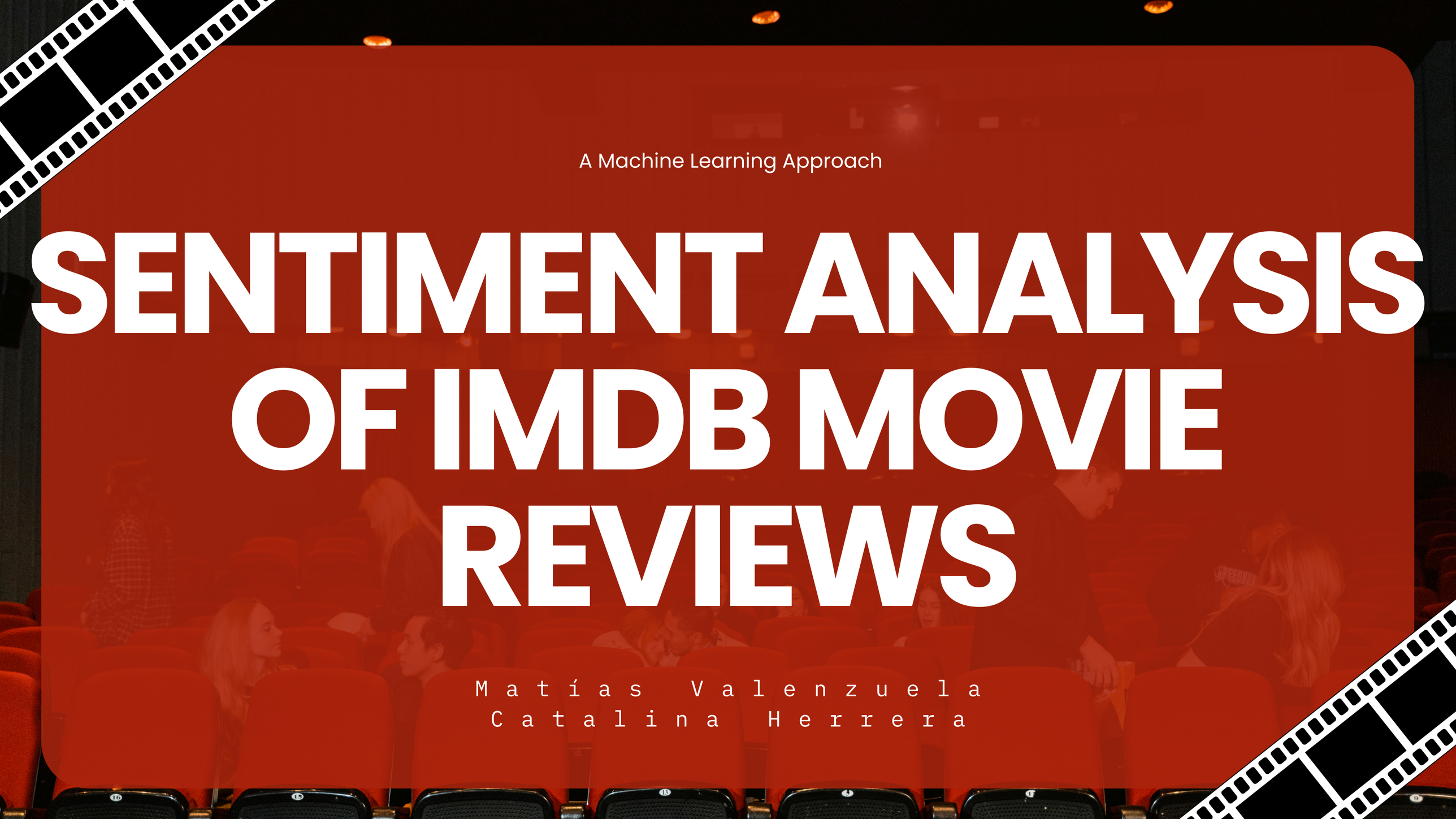


A Machine Learning Approach

SENTIMENT ANALYSIS OF IMDB MOVIE REVIEWS

Matías Valenzuela
Catalina Herrera

THE GOAL

OBJECTIVE

Automatically classify a movie review as **Positive** or **Negative**.

WHY?

This has applications in customer feedback analysis, brand monitoring, and content recommendation.

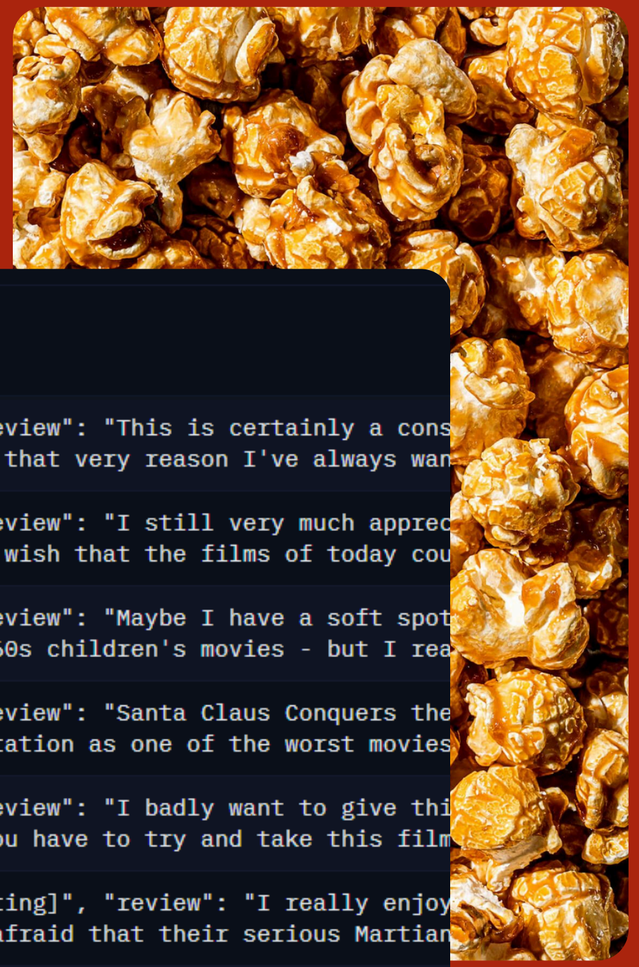
OUR TASK

Task: Binary Text Classification

- Dataset: IMDB Movie Reviews
- Algorithm: Logistic Regression

THE DATASET: IMDB REVIEWS

A COLLECTION OF 50,000 MOVIE REVIEWS.



Perfectly balanced:

- 25,000 for training (12.5k positive, 12.5k negative)
- 25,000 for testing (12.5k positive, 12.5k negative)

We created an additional **validation set** (5,000 reviews) from the training data for fine-tuning our model.

movie_id string	**reviews** dict
tt0058548	{ "rating": "4", "review": "This is certainly a cons ever made - and for that very reason I've always wan
tt0058548	{ "rating": "4", "review": "I still very much apprec and sense of fun. I wish that the films of today cou
tt0058548	{ "rating": "4", "review": "Maybe I have a soft spot conceived, silly 1960s children's movies - but I rea
tt0058548	{ "rating": "5", "review": "Santa Claus Conquers the 60's. It has a reputation as one of the worst movies
tt0058548	{ "rating": "5", "review": "I badly want to give thi abysmal. However, you have to try and take this film
tt0058548	{ "rating": "[No Rating]", "review": "I really enjoy Martian adults are afraid that their serious Martian
tt0058548	{ "rating": "2", "review": "Since plan nine failed, become obsessed with American TV, and apparently in

OUR METHODOLOGY: THE PIPELINE

```python
def load_imdb_dataset():  1 usage  MathiCL +1
    """
    Load IMDB dataset using HuggingFace `datasets`.
    Returns:
        train_texts, train_labels, test_texts, test_labels
    """
    print("Loading IMDB dataset from HuggingFace...")
    dataset = load_dataset("imdb") # dataset de imdb

    #  extrae los textos y las etiquetas (0 para "negativo",
    #  1 para "positivo") para los conjuntos de
    #  entrenamiento y de prueba
    train_texts = dataset["train"]["text"]
    train_labels = dataset["train"]["label"]

    test_texts = dataset["test"]["text"]
    test_labels = dataset["test"]["label"]

    print(f"Train size: {len(train_texts)}")
    print(f"Test size : {len(test_texts)}")
    print()

    # los devuelve para que otras partes del código puedan usarlos
    return train_texts, train_labels, test_texts, test_labels
```

A 4-STEP PROCESS:

1. **Load Data:** Import the IMDB dataset.

2. **Vectorize Text:** Convert words into numbers using TF-IDF. This tells the model how important a word is to a review.

3. **Train Model:** Teach a Logistic Regression model to distinguish between positive and negative reviews.

4. **Evaluate:** Test the model on data it has never seen before.

OUR METHODOLOGY: THE PIPELINE

```python
build_pipeline() -> Pipeline:  2 usages  MathiCL +1
"""
Build a scikit-learn Pipeline: TF-IDF vectorizer + Logistic Regression.
"""
#  Ensambla la máquina de aprendizaje, construye la linea de montaje
tfidf = TfidfVectorizer( # el traductor texto-números
    max_features=MAX_FEATURES,
    ngram_range=NGRAM_RANGE,
    stop_words="english",
)

lr = LogisticRegression() # CEREBRO del modelo, el clasificador

pipe = Pipeline(
    steps=[
        ("tfidf", tfidf), # primero el traductor,
        ("lr", lr),       # luego el clasificador
    ]
)

return pipe
```

A 4-STEP PROCESS:

1. **Load Data:** Import the IMDB dataset.

2. **Vectorize Text:** Convert words into numbers using TF-IDF. This tells the model how important a word is to a review.

3. **Train Model:** Teach a Logistic Regression model to distinguish between positive and negative reviews.

4. **Evaluate:** Test the model on data it has never seen before.

OUR METHODOLOGY: THE PIPELINE

```python
def evaluate_model(  3 usages  MathiCL +1
    name: str, model: Pipeline, X, y, save_confusion: bool = False, split_name: str = ""
):
    """
    Print standard classification metrics and optionally save confusion matrix.
    """
    # Le pone una nota al modelo
    y_pred = model.predict(X) # hace las predicciones

    # Cálculo de las métricas
    accuracy = accuracy_score(y, y_pred)
    precision = precision_score(y, y_pred)
    recall = recall_score(y, y_pred)
    f1 = f1_score(y, y_pred)

    # Impresión de un informe claro de las métricas
    print(f"===== {name} =====")
    print(f"Accuracy : {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall   : {recall:.4f}")
    print(f"F1-score : {f1:.4f}")
    print()
    print("Classification report:")
    print(classification_report(y, y_pred, target_names=["negative", "positive"]))
    print("-" * 60)

    if save_confusion:
        # llamará a la función plot_confusion_matrix
        # para guardar el gráfico
        cm = confusion_matrix(y, y_pred)
        plot_confusion_matrix(
            cm,
            classes=["negative", "positive"]
```

A 4-STEP PROCESS:

1. **Load Data:** Import the IMDB dataset.

2. **Vectorize Text:** Convert words into numbers using TF-IDF. This tells the model how important a word is to a review.

3. **Train Model:** Teach a Logistic Regression model to distinguish between positive and negative reviews.

4. **Evaluate:** Test the model on data it has never seen before.

STEP 1 - FEATURE EXTRACTION WITH TF-IDF

Problem: Machines don't understand words, they understand numbers.

Solution: TF-IDF (Term Frequency-Inverse Document Frequency)

- TF (Term Frequency): How often does a word appear in a review?
- IDF (Inverse Document Frequency): How rare is that word across all reviews?

Result: Important words get a high score. Common words like "the" or "a" get a low score.

```
RANDOM_STATE = 42
MAX_FEATURES = 20000          # número máximo de tokens en el vocabulario
NGRAM_RANGE = (1, 2)          # unigrams + bigrams
MODEL_DIR = os.path.join("models")
REPORTS_DIR = os.path.join("reports")
```

```
#  Ensambla la máquina de aprendizaje, construye la linea de montaje
tfidf = TfidfVectorizer( # el traductor texto-números
    max_features=MAX_FEATURES,
    ngram_range=NGRAM_RANGE,
    stop_words="english",
)
```

STEP 2 – MODEL TRAINING & OPTIMIZATION

Algorithm: Logistic Regression

A simple, fast, and effective linear model for classification.

Optimization: How do we find the best version of our model?

We used GridSearchCV.

- It's an automated process that tests different hyperparameter combinations to find the one with the best performance.
- Key parameters tuned: n-gram range, vocabulary size, and regularization strength (C).

```
lr = LogisticRegression() # CEREBRO del modelo, el clasificador
```

```
def run_grid_search(pipe: Pipeline, X_train, y_train) -> GridSearchCV:  1 usage  MathiCL +1
    """
    Run GridSearchCV over a small hyperparameter grid.
    """
    # Encontrar la mejor configuración para el modelo
    #  Funciona como un "ingeniero automático"
    print("Running GridSearchCV (this may take some minutes)...")

    param_grid = { # es el diccionario con todas las "piezas" a probar
        "tfidf__max_features": [10000, 20000],
        "tfidf__ngram_range": [(1, 1), (1, 2)],
        "lr__learning_rate": [0.01, 0.1],
        "lr__n_iterations": [1000, 2000],
    }

    grid = GridSearchCV( #3ejecuta GridSearchCV para que pruebe todas las
                # combinaciones posibles sobre los datos de entrenamiento
        pipe,
        param_grid=param_grid,
        cv=3,
        scoring="f1",
        n_jobs=-1,
        verbose=1,
    )

    grid.fit(X_train, y_train)

    # Se imprime la mejor combinación hallada y su puntuación
    print("Best parameters:")
    print("\n===== GridSearchCV results =====")
    print("Best params:", grid.best_params_)
    print(f"Best CV F1: {grid.best_score_:.4f}")
```

THE WINNING COMBINATION

GRIDSEARCHCV FOUND THE BEST SETTINGS FOR OUR MODEL:

- N-grams: ‘(1, 2)’ -> The model looks at both single words ("amazing") and pairs of words ("very good"). This captures more context.
- Vocabulary Size: ‘10,000' features -> The model focuses on the 10,000 most important words.
- lr__learning_rate: Controls the size of the model's learning steps. Small = slow and accurate; large = fast but risky.
- lr__n_iterations: Number of times the model reviews all the data to adjust its parameters.

```
param_grid = { # es el diccionario con todas las "piezas" a probar
    "tfidf__max_features": [10000, 20000],
    "tfidf__ngram_range": [(1, 1), (1, 2)],
    "lr__learning_rate": [0.01, 0.1],
    "lr__n_iterations": [1000, 2000],
}
```

```
===== GridSearchCV results =====
Best params: {'lr__learning_rate': 0.1, 'lr__n_iterations': 2000, 'tfidf__max_features': 10000, 'tfidf__ngram_range': (1, 2)}
Best CV F1: 0.7999
------------------------------------------------------------
```

THE RESULTS: HOW WELL DID IT DO?

We evaluated the final, optimized model on the 25,000 unseen test reviews.

Final Accuracy: 79.4%

Key Metrics:

- Precision: 78.1% (When it predicts "positive", it's right 78.1% of the time).

- Recall: 81.7% (It correctly identifies 88.1% of all actual "positive" reviews).

- F1-Score: 79.8% (A combined measure of Precision and Recall, showing a robust and balanced model).

```
=== Final evaluation on TEST set ===

===== Best model (test) =====

Accuracy  : 0.7942

Precision: 0.7811

Recall    : 0.8175

F1-score  : 0.7989
```

CONFUSION MATRIX (VISUALIZING THE RESULTS)

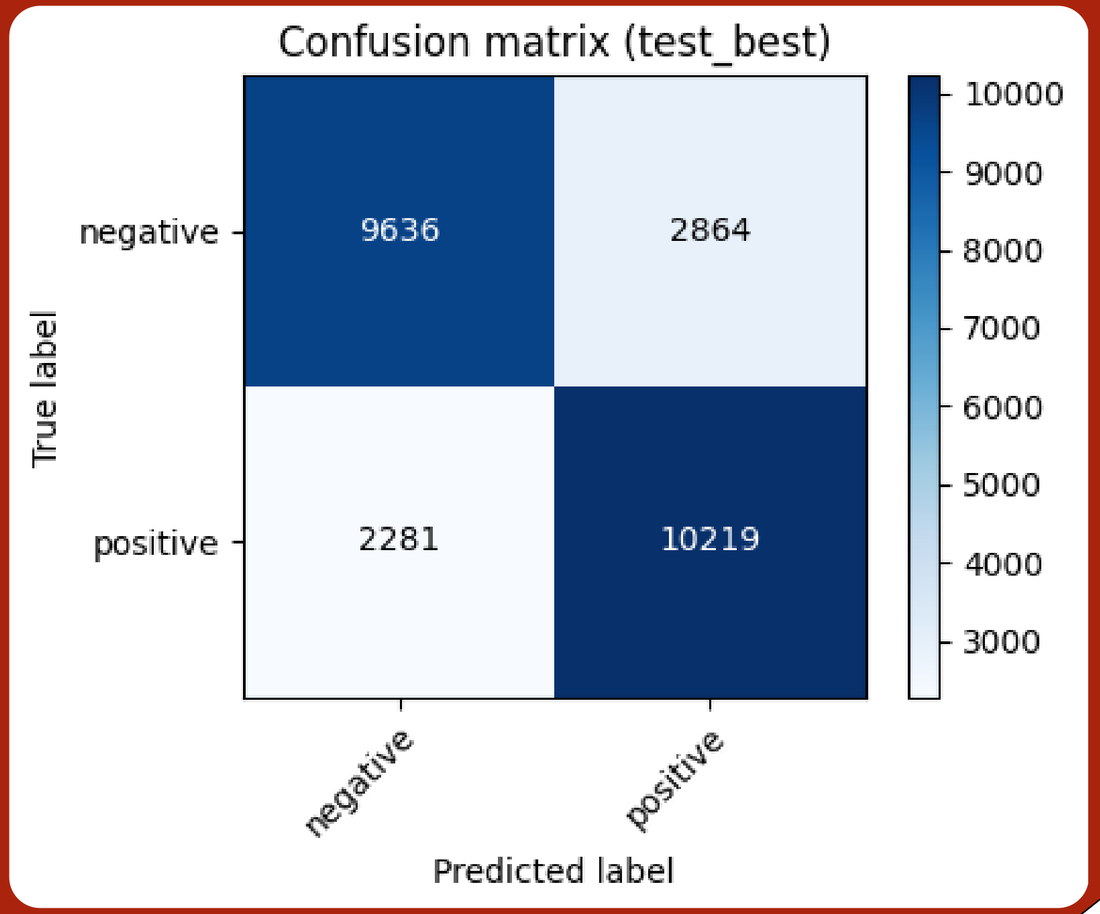
What it shows:

- High numbers on the diagonal (top-left to bottom-right) are **correct** predictions.

- Low numbers on the off-diagonal are **incorrect** predictions.

Our result:

The model is very good at correctly identifying both positive and negative reviews, with a similar number of errors for each class.



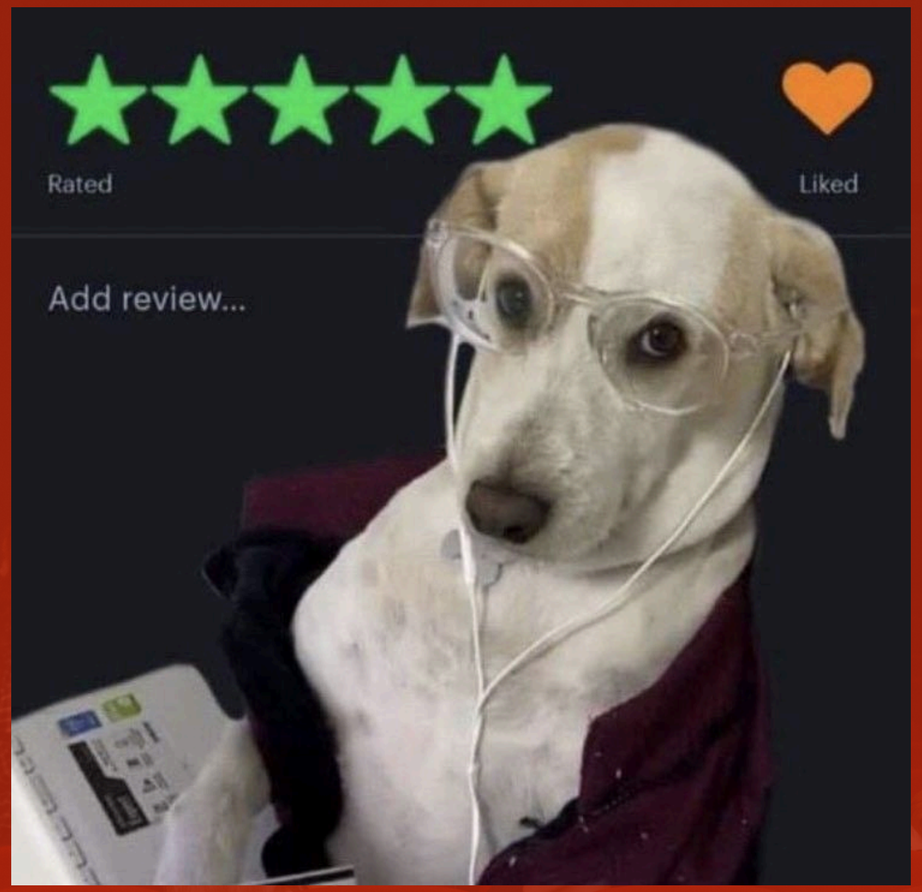
CONCLUSION

¡SUCCESS!

We built a highly effective sentiment analysis model with an 79.8% F1-score

Key Takeaways:

- Proper data splitting (train/validation/test) is essential.
- TF-IDF is a powerful technique for text feature extraction.
- Hyperparameter tuning with GridSearchCV significantly improves model performance.
- Logistic Regression, while simple, is a very strong baseline for text classification.



THANK YOU