JAIN UNIVERSITY

Language Detection Using NLP & ML

Mini Project –
Advanced Machine Learning (23CSE514)
B.Tech CSE – AI

Athul S Nair 23BTRCA019

Introduction

- **Language detection = identifying the language of any text**
- **Used in translation systems, NLP pipelines, chatbots, search engines**
- **This project detects 17 different languages using Machine Learning Model**
- **Uses TF-IDF + Logistic Regression for high accuracy**

Prior Study (Literature Review)

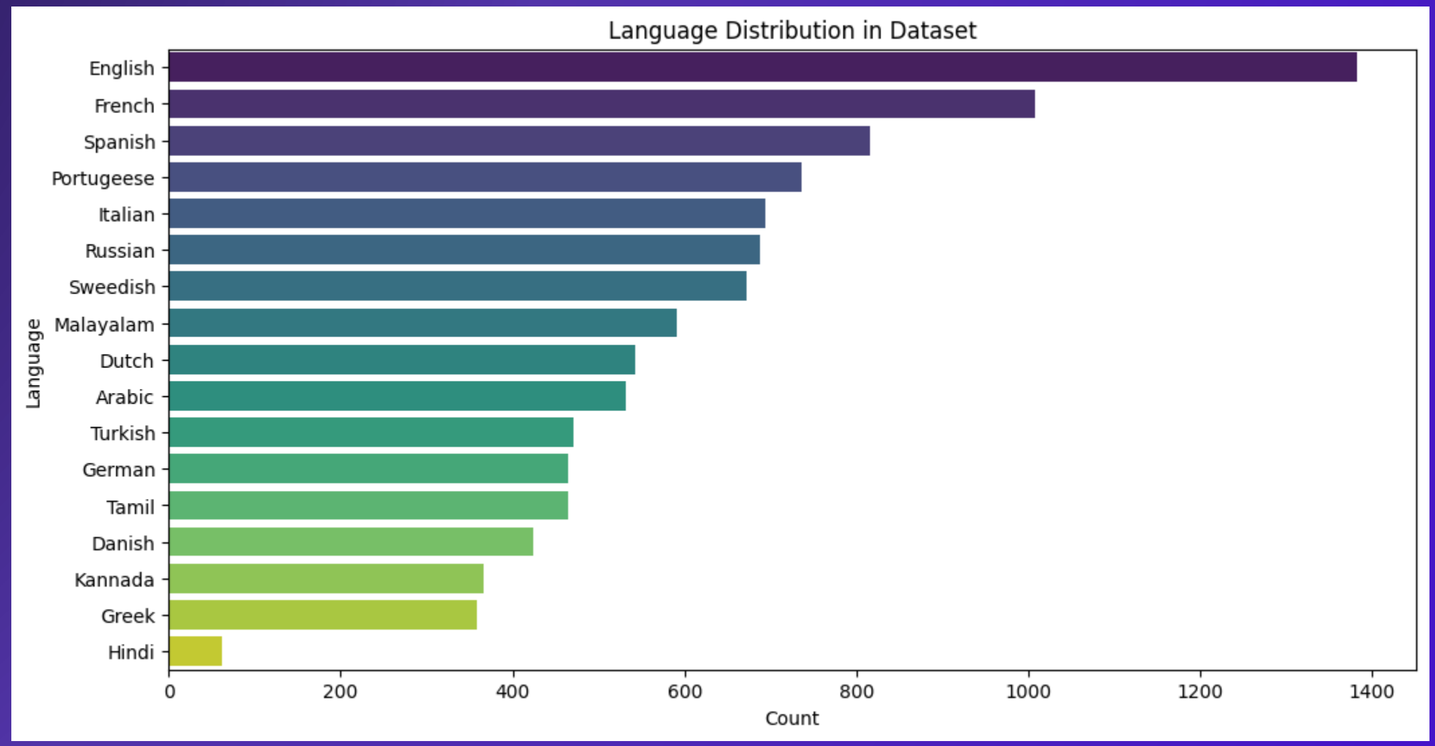
- Traditional rule-based approaches: dictionaries, stopwords (not accurate)
- N-gram models by Cavnar & Trenkle (1994) → high accuracy
- ML models used: Naïve Bayes, Logistic Regression, SVM
- Deep learning exists but heavy for small datasets
- TF-IDF + LR remains most practical and effective for language identification

Problem Statement

- Need automatic language identification from user input text
- Multilingual datasets contain multiple scripts (Latin, Arabic, Devanagari, Dravidian)
- Dataset imbalance leads to bias in predictions
- Goal: Build a robust, accurate, script-independent language detection model

Dataset

- Dataset: Language Detection.csv
- Contains 17 languages
- Around 10,267 samples originally
- Highly imbalanced:
  - English > 1000 samples
  - Hindi ~ 63 samples
- Balancing required to avoid bias



Methodology

- Data Cleaning:
  - Remove nulls
  - Remove duplicates
  - Convert text to lowercase
- Feature extraction: Character-level TF-IDF (2–4 n-grams)
- Balancing: Random Oversampling
- Train-test split: 80% training, 20% testing
- Model: Logistic Regression (multiclass)

How does the model detect a language?

Step 1 → Input comes from user
Example:
“hello"

Step 2 → Convert text into character n-grams
The sentence becomes many pieces like:
For the word "hello":

- **2-gram (bigram): he, el, ll, lo**
- **3-gram (trigram): hel, ell, llo**
- **4-gram: hell, ello**

So, an n-gram helps the model learn the patterns of characters that are unique to each language.

How does the model detect a language?

Step 3 → TF-IDF gives weights to each n-gram
TF-IDF assigns:
High weight → if a character pattern is important/unique
Low weight → if it is common in many languages
This converts text → **numeric vector**.

Step 4 → Logistic Regression uses learned patterns
During training, the model saw thousands of samples per language and learned:
Spanish texts = have patterns like "¿", "ó", "está", "que"
Hindi = “क”, “न”, “ि” etc.
Malayalam = “ൽ”, “ക”, “ാൻ”
Arabic = “”کا“ ,”مر“ ,”ال
English = “the”, “ing”, “er”
It learns **character fingerprints**.

How does the model detect a language?

Step 5 → Model predicts the language
Example output:
Spanish: **0.98**
Portuguese: 0.01
Italian: 0.01

So, the predicted language = English.

Code Implementation (Training)

- TF-IDF vectorizer for character n-grams
- RandomOverSampler() for balancing
- LogisticRegression(max_iter=2000) for stable training
- Save model using joblib
- Prediction function defined for inference

Code Implementation (Training)

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from imblearn.over_sampling import RandomOverSampler

df = pd.read_csv("/content/Language Detection.csv")

df = df.drop_duplicates(subset=["Text"])

df = df.dropna()

print("Dataset size before balancing:", len(df))

X = df["Text"]
y = df["Language"]

vectorizer = TfidfVectorizer(analyzer="char", ngram_range=(2,4))
X_vec = vectorizer.fit_transform(X)

oversample = RandomOverSampler()
X_bal, y_bal = oversample.fit_resample(X_vec, y)

print("Dataset size after balancing:", len(y_bal))

X_train, X_test, y_train, y_test = train_test_split(
    X_bal, y_bal, test_size=0.2, random_state=42
)
```

```python
model = LogisticRegression(max_iter=2000)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))

def predict_language(text: str) -> str:
    x = vectorizer.transform([text])
    return model.predict(x)[0]

def interactive_console():
    print("\n=== Interactive Language Detection (type 'q' to quit) ===")
    while True:
        user_text = input("\nEnter text: ")
        if user_text.strip().lower() in ["q", "quit", "exit"]:
            print("Exiting...")
            break
        if not user_text.strip():
            print("Please enter some text.")
            continue

        lang = predict_language(user_text)
        print("Predicted Language:", lang)

try:
    import ipywidgets as widgets
```

```python
try:
    import ipywidgets as widgets
    from IPython.display import display, Markdown

    text_box = widgets.Textarea(
        value="Type some text here...",
        placeholder="Enter text to detect language",
        description="Input:",
        layout=widgets.Layout(width="100%", height="120px")
    )

    button = widgets.Button(
        description="Detect Language",
        button_style="",  # 'success', 'info', 'warning', 'danger' or ''
        tooltip="Click to detect language"
    )

    output = widgets.Output()

    def on_button_click(b):
        with output:
            output.clear_output()
            user_text = text_box.value.strip()
            if not user_text:
                print("Please enter some text.")
                return
            lang = predict_language(user_text)
            display(Markdown(f"**Predicted Language:** `{lang}`"))

    button.on_click(on_button_click)

    display(Markdown("## Language Detection"))
    display(text_box, button, output)

except ImportError:
    print("ipywidgets not available – only console mode can be used.")
```

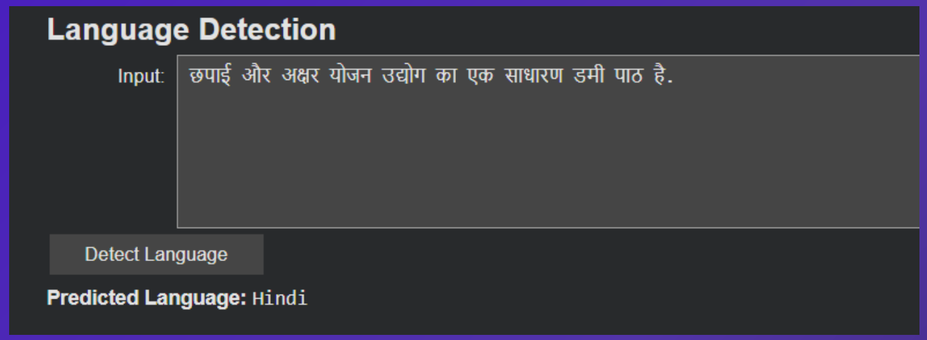
Model Evaluation

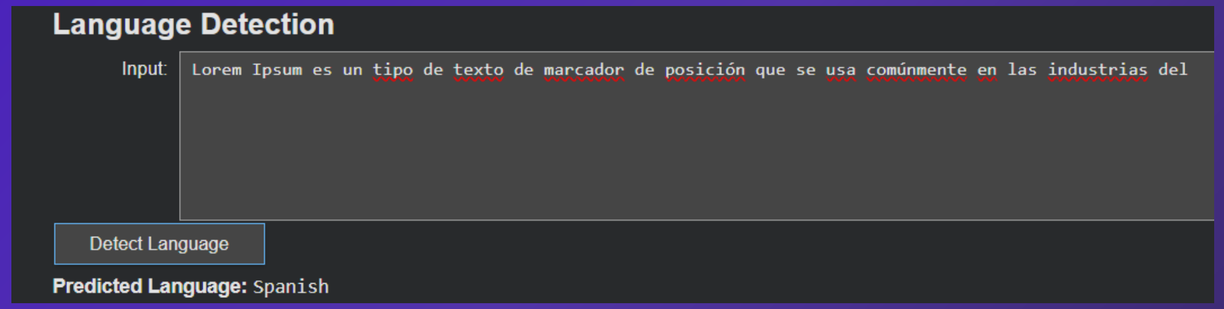
```
Dataset size before balancing: 10267
Dataset size after balancing: 23494
Accuracy: 0.9970206426899341
```

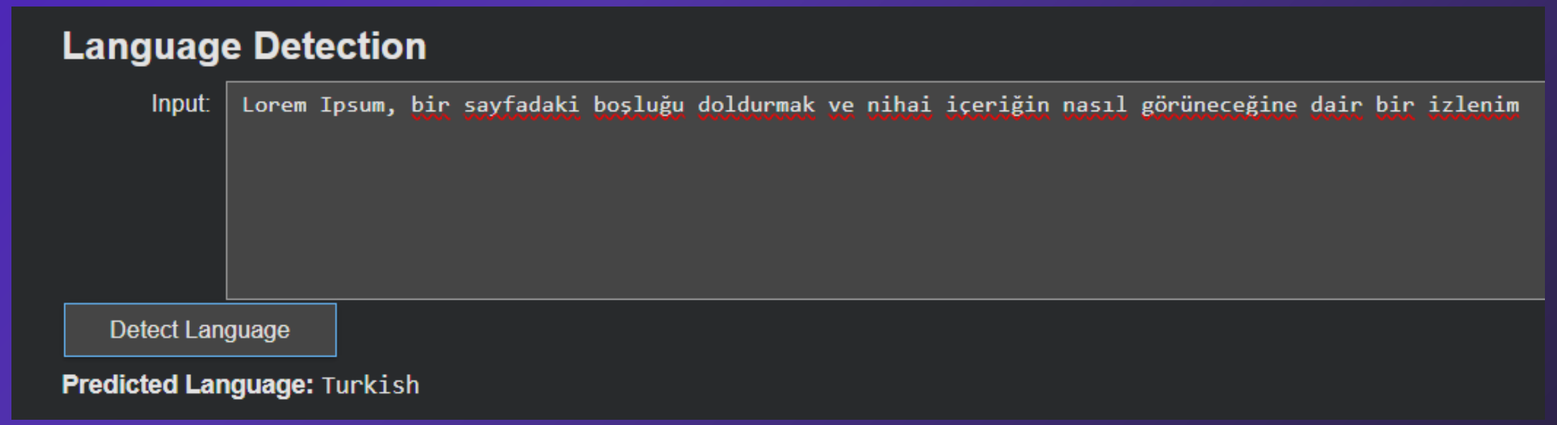
Classification Report:

	precision	recall	f1-score	support
Arabic	1.00	1.00	1.00	248
Danish	1.00	1.00	1.00	264
Dutch	1.00	0.99	1.00	297
English	0.99	1.00	0.99	250
French	1.00	1.00	1.00	308
German	0.99	1.00	0.99	291
Greek	1.00	1.00	1.00	266
Hindi	1.00	1.00	1.00	278
Italian	0.99	0.99	0.99	279
Kannada	1.00	1.00	1.00	267
Malayalam	1.00	1.00	1.00	271
Portugeese	0.99	0.99	0.99	291
Russian	1.00	1.00	1.00	288
Spanish	0.99	0.99	0.99	282
Sweedish	1.00	1.00	1.00	281
Tamil	1.00	1.00	1.00	259
Turkish	1.00	1.00	1.00	279
accuracy			1.00	4699
macro avg	1.00	1.00	1.00	4699
weighted avg	1.00	1.00	1.00	4699

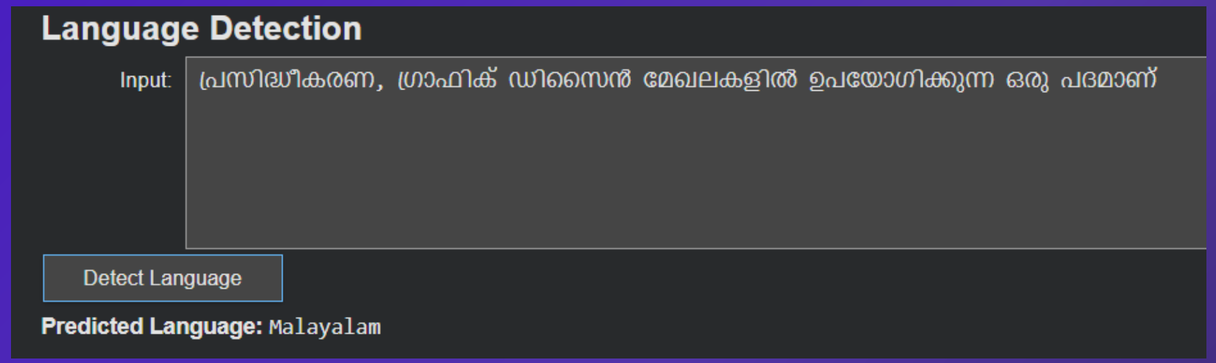
Code Implementation: Output

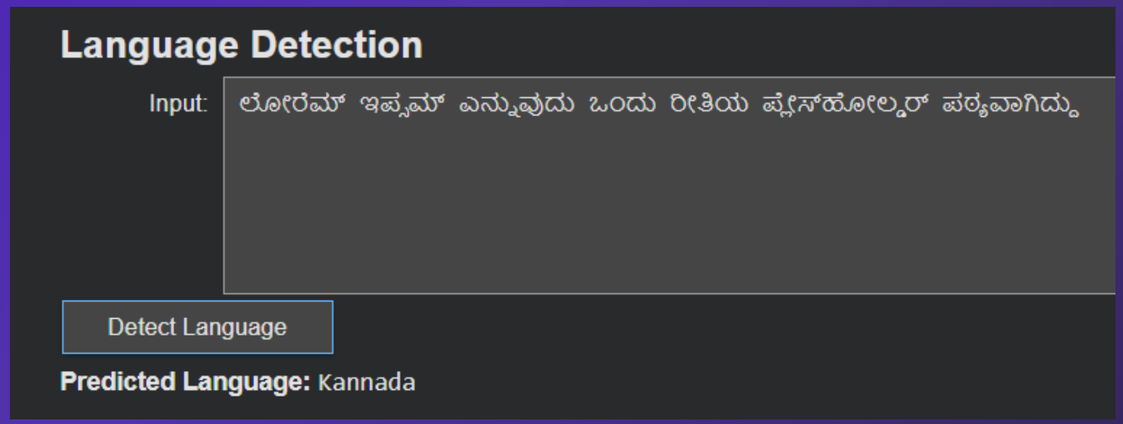






Code Implementation: Output

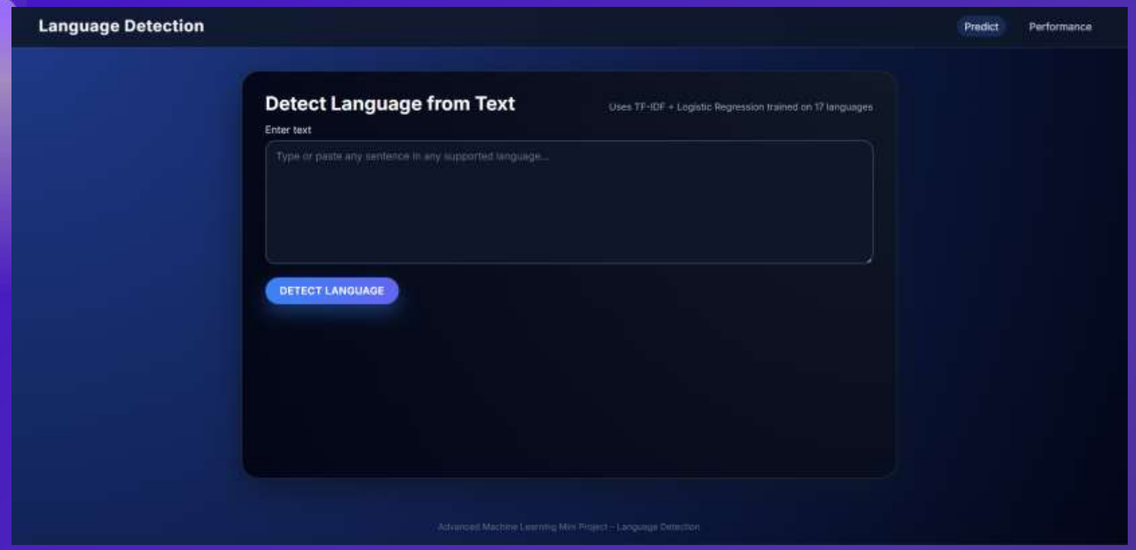


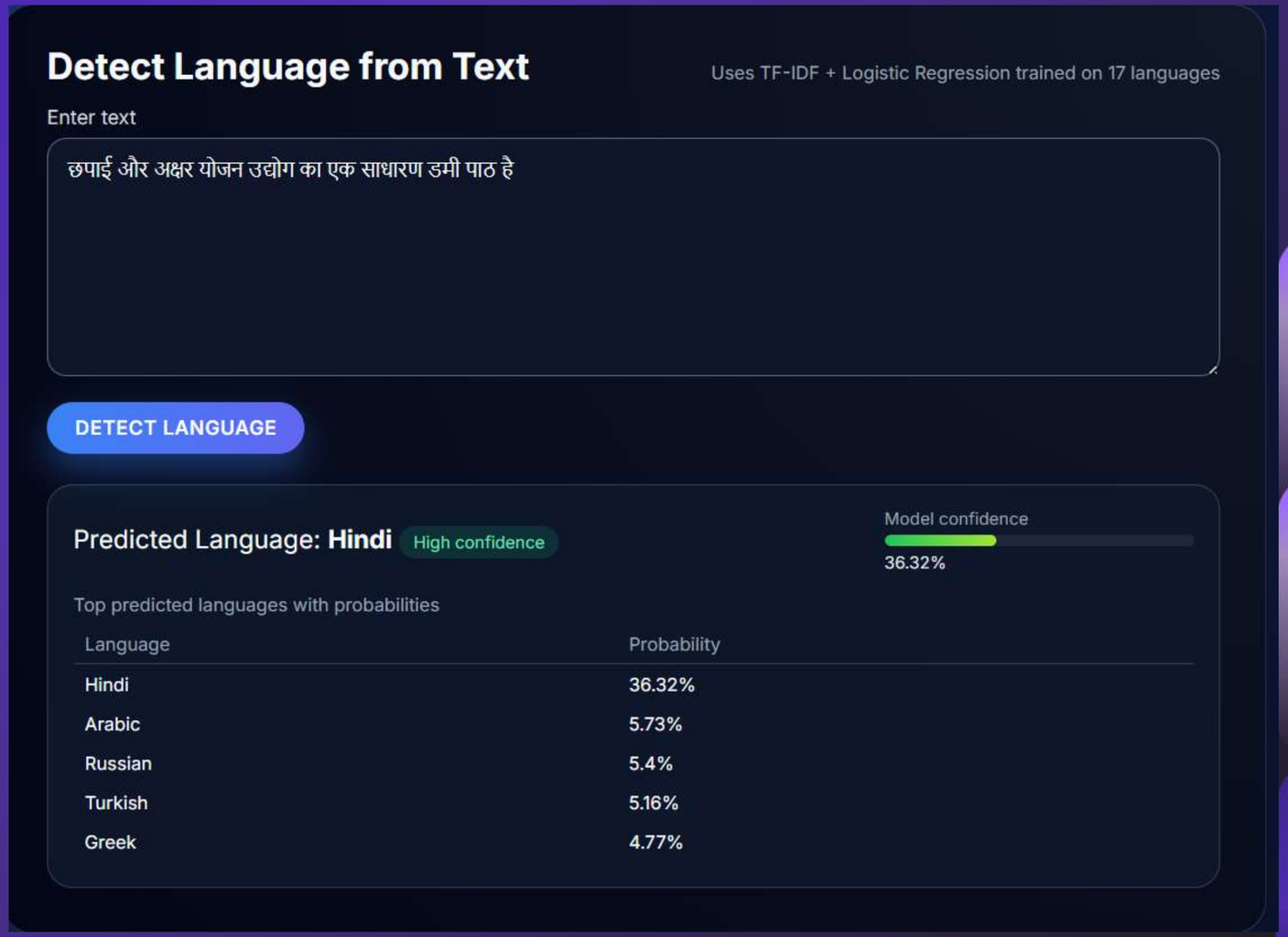


Code Implementation (Flask App)

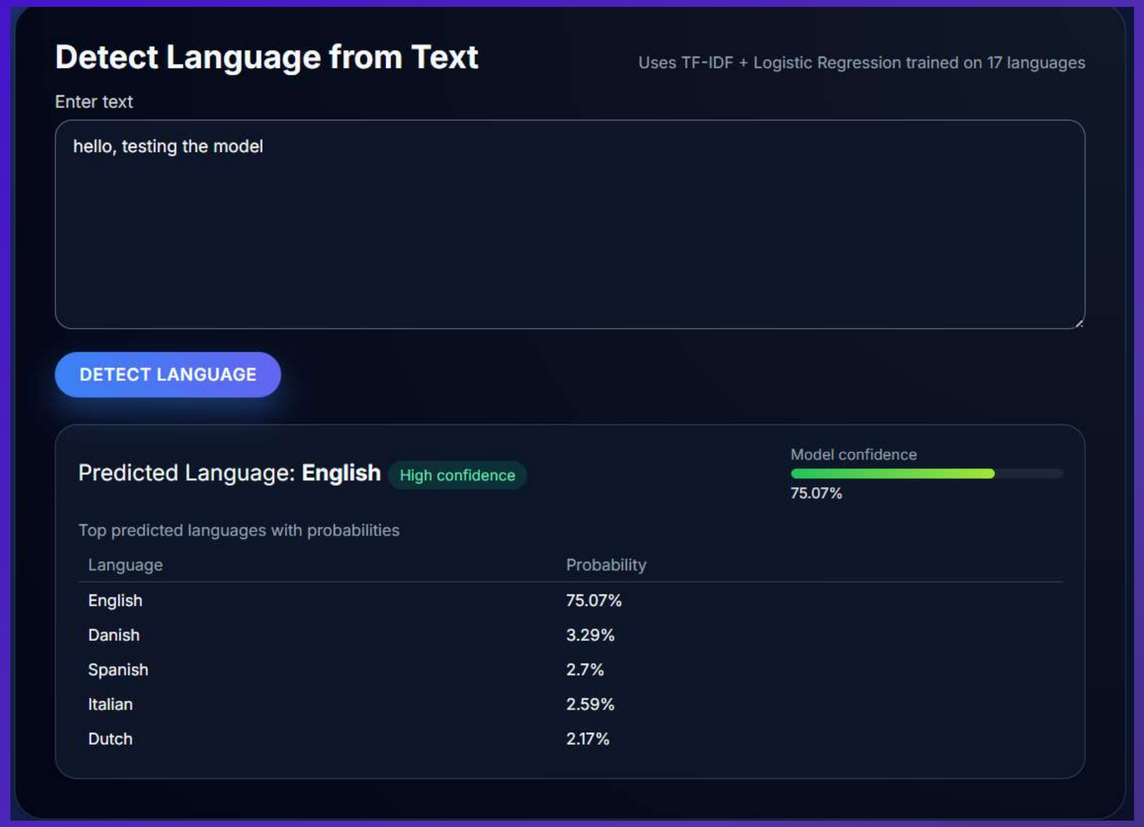
- Load saved lang_model.pkl and vectorizer.pkl
- Flask routes:
  - / → detect language
  - /performance → show model accuracy & F1-scores
- UI: text input box → predicted language
- Confidence score displayed using predict_proba()
- Modern UI with HTML/CSS

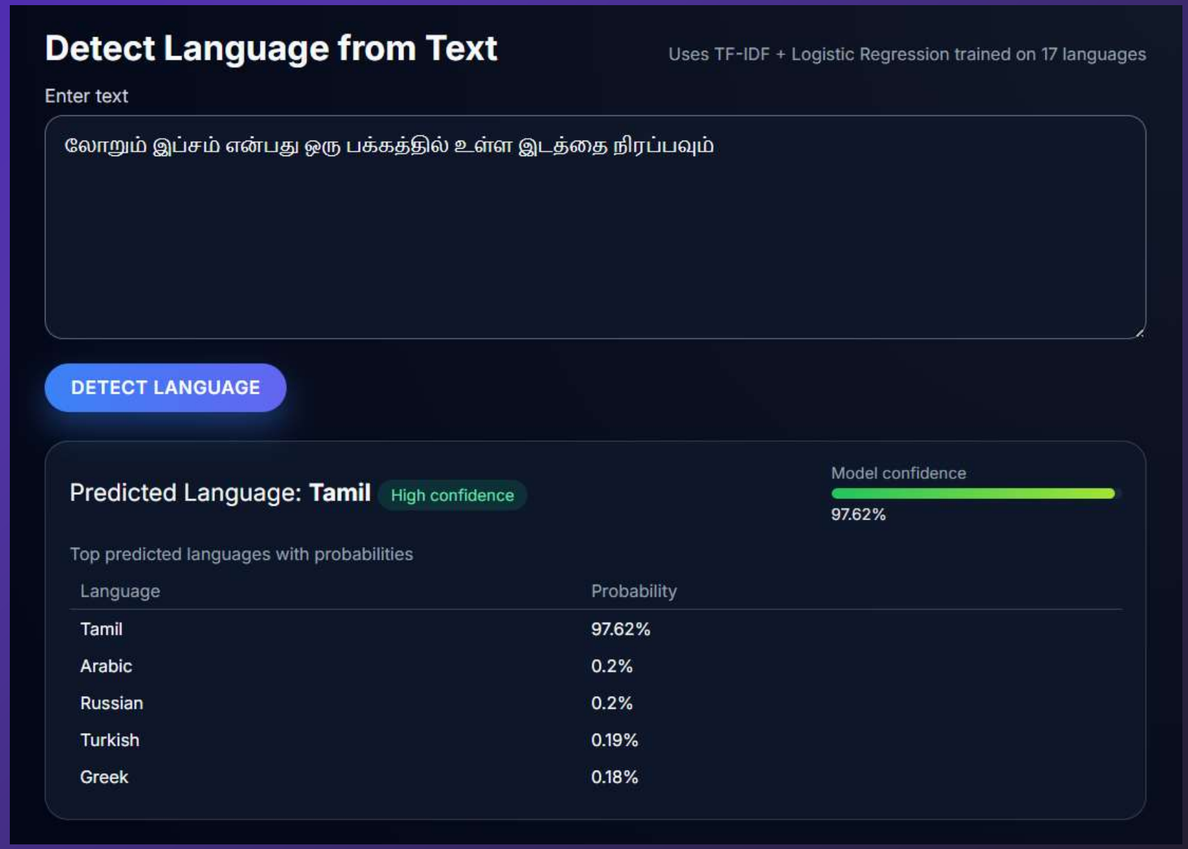
Code Implementation (Flask App)





Code Implementation (Flask App)





Code Implementation (Flask App)

Model Performance

Evaluation on a held-out test split of the Language Detection dataset.

Accuracy	Macro F1-score	Weighted F1-score	Macro Precision / Recall
0.999	1.0	1.0	1.0 / 1.0

Per-class metrics

Language	Precision	Recall	F1-score	Support
Arabic	1.0	1.0	1.0	106
Danish	1.0	1.0	1.0	85
Dutch	1.0	1.0	1.0	108
English	1.0	1.0	1.0	277
French	1.0	1.0	1.0	201
German	1.0	1.0	1.0	93
Greek	1.0	1.0	1.0	72
Hindi	1.0	1.0	1.0	12
Italian	0.99	1.0	0.99	139
Kannada	1.0	1.0	1.0	73
Malayalam	1.0	1.0	1.0	118
Portugeese	1.0	0.99	1.0	147
Russian	1.0	1.0	1.0	138
Spanish	1.0	0.99	1.0	163
Sweedish	1.0	1.0	1.0	135
Tamil	1.0	1.0	1.0	93
Turkish	1.0	1.0	1.0	94

Results

- Accuracy: 99.9%
- Macro F1-score: 1.0
- Weighted F1-score: 1.0
- Perfect per-language results
- Balanced dataset fixed model bias
- Flask app predicts accurately in real-time

Summary

- Complete NLP + ML pipeline implemented
- Very high performance achieved
- Character-level n-grams ideal for language detection
- Balanced dataset ensured fairness
- Deployed using Flask for real-world usage

References

- Cavnar & Trenkle (1994) – N-gram text categorization
- Manning et al. – *Introduction to Information Retrieval*
- scikit-learn documentation
- imbalanced-learn documentation
- Flask documentation

Thank You