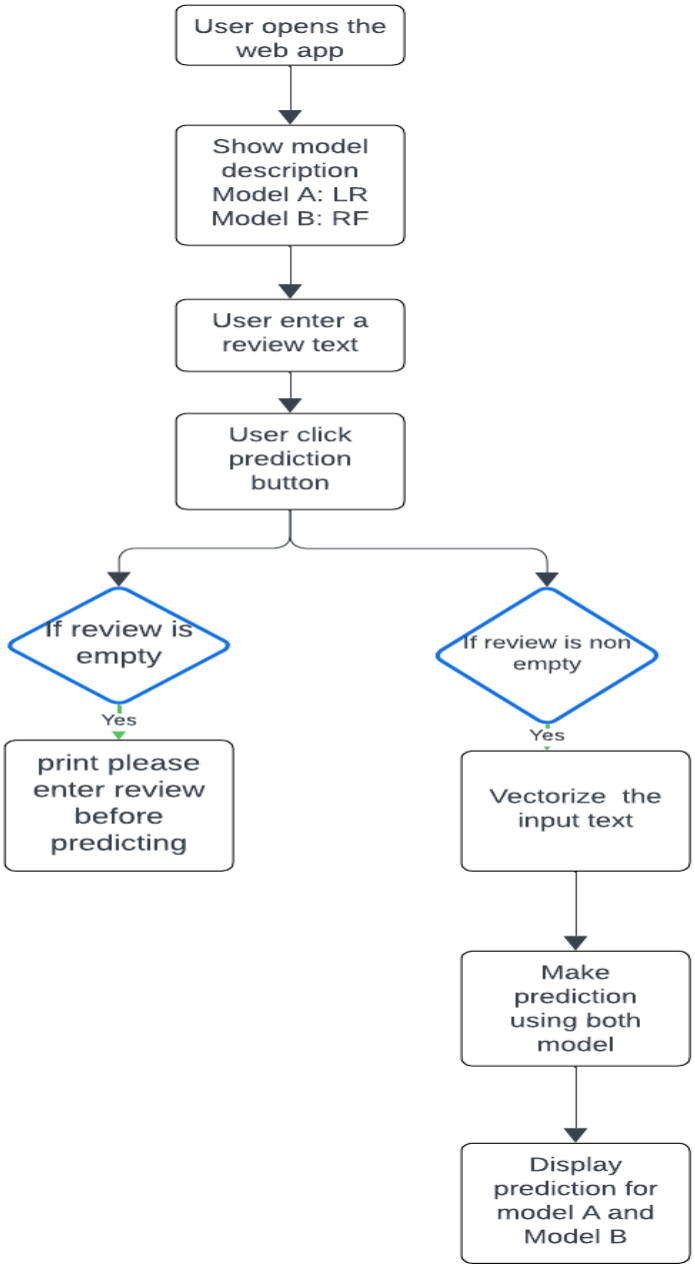
AUTOMATED REVIEW SYSTEM

FLOW CHART



Code:

```
import streamlit as st
import pickle
# ----------------------------
# 1. Load Models and Vectorizers
# ----------------------------
@st.cache_resource
def load_assets():
    with open("balance.pkl", "rb") as f:
        model_A = pickle.load(f)
    with open("vecbalance.pkl", "rb") as f:
        vectorizer_A = pickle.load(f)
    with open("random_forest_model.pkl", "rb") as f:
        model_B = pickle.load(f)
    with open("vectorizer.pkl", "rb") as f:
        vectorizer_B = pickle.load(f)
    return model_A, vectorizer_A, model_B, vectorizer_B
model_A, vectorizer_A, model_B, vectorizer_B = load_assets()

# ----------------------------
# 2. Streamlit UI Layout
# ----------------------------

st.title("AUTOMATED REVIEW SYSTEM")
st.subheader("Compare Predictions from Two Specialized Models")
st.markdown("""
- **Model A**: Trained on Logistic regression
- **Model B**: Trained on Random Forest
""")

# ----------------------------
```

```
# 3. User Input
# ---------------------------
review = st.text_area("📝 Enter a product review to predict its rating:", height=150)


if st.button("📊 Predict Rating"):
    if not review.strip():
        st.warning("Please enter a review before predicting.")
    else:
        # Vectorize using each model's respective vectorizer
        vec_review_A = vectorizer_A.transform([review])
        vec_review_B = vectorizer_B.transform([review])

        # Get predictions
        pred_A = model_A.predict(vec_review_A)[0]
        pred_B = model_B.predict(vec_review_B)[0]

        # ---------------------------
        # 4. Display Results
        # ---------------------------
        col1, col2 = st.columns(2)
        with col1:
            st.header("Model A")
            st.success(f"🔸 Predicted Rating: **{pred_A}**")
        with col2:
            st.header("Model B")
            st.info(f"🔹 Predicted Rating: **{pred_B}**")
```

UI Design Decisions

1. Clear Hierarchical Structure

- **st.title("AUTOMATED REVIEW SYSTEM")**: Creates a bold, top-level title to clearly introduce the purpose of the app.
- **st.subheader("Compare Predictions from Two Specialized Models")**: Provides context, letting users know they are comparing two models.

2. Simple, Focused Input Area

- **st.text_area()**: Used for multiline input, suitable for reviews that vary in length. A height=150 allows comfortable visibility without being overwhelming.

3. Model Descriptions

- **st.markdown()**: Clarifies the models being used (Logistic Regression and Random Forest) and sets the expectation that two predictions will be shown.

4. User Feedback

- **st.warning()**: Gives immediate feedback if the user submits without entering a review — enhances usability.
- **st.button(" Predict Rating")**: A call-to-action button with an emoji improves clarity and engagement.

5. Side-by-Side Result Comparison

- **st.columns(2)**: Displays predictions from Model A and Model B side-by-side, enabling intuitive comparison.
- **st.success() and st.info()**: Different message styles and emojis are used to visually distinguish the models and results.

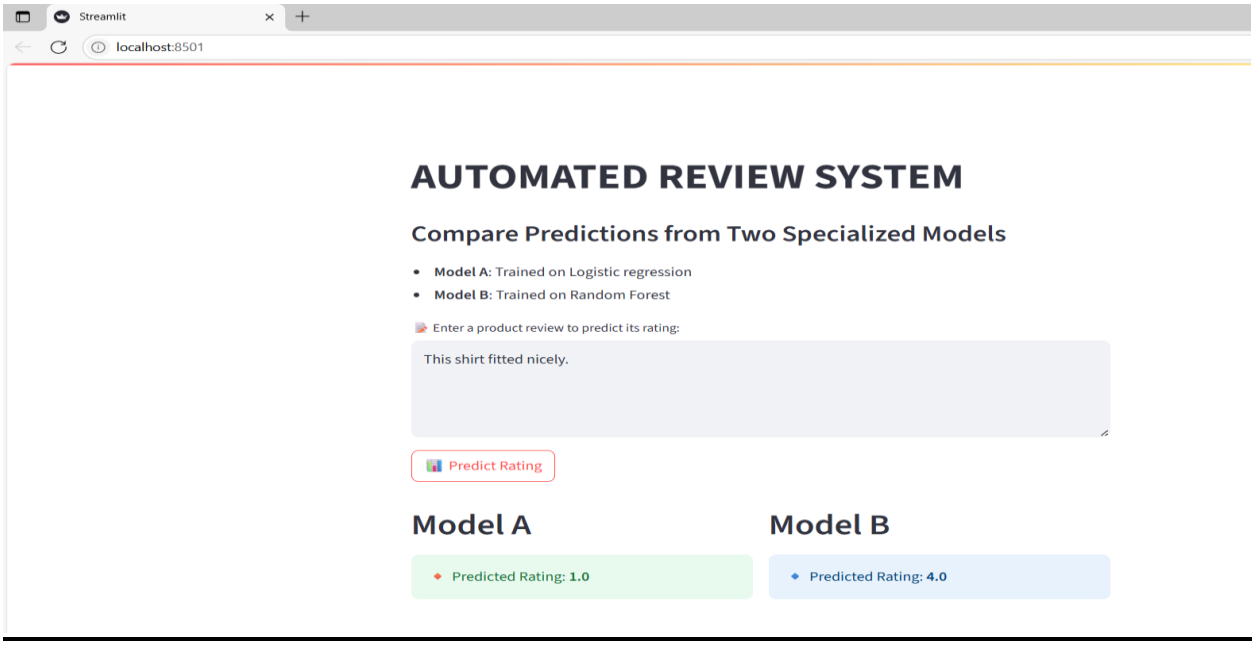
6. Cache Usage

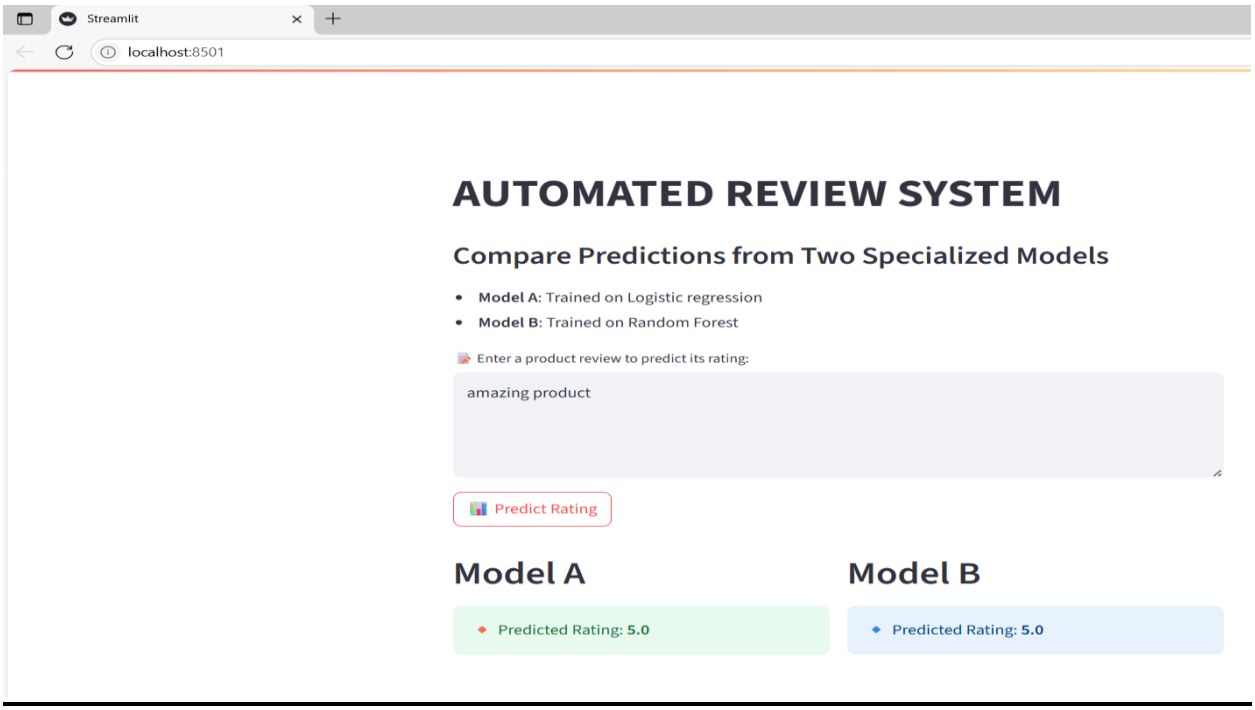
- **@st.cache_resource**: Ensures models and vectorizers are loaded once and reused, optimizing app performance during multiple interactions.

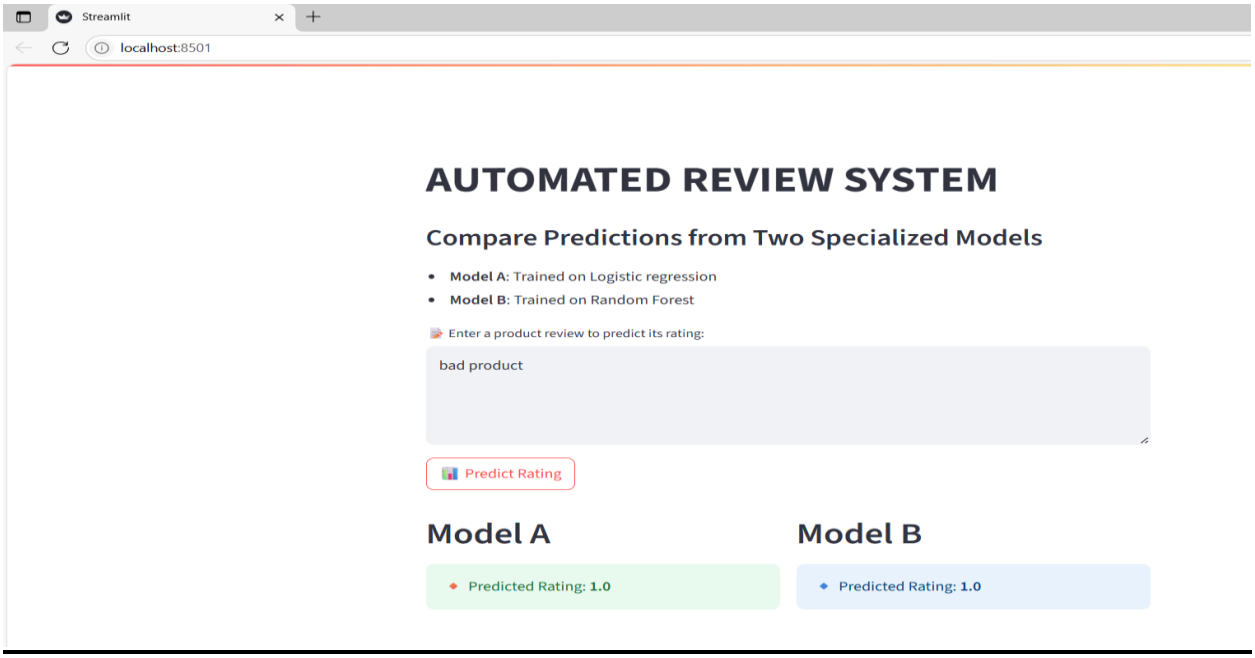
User Flow

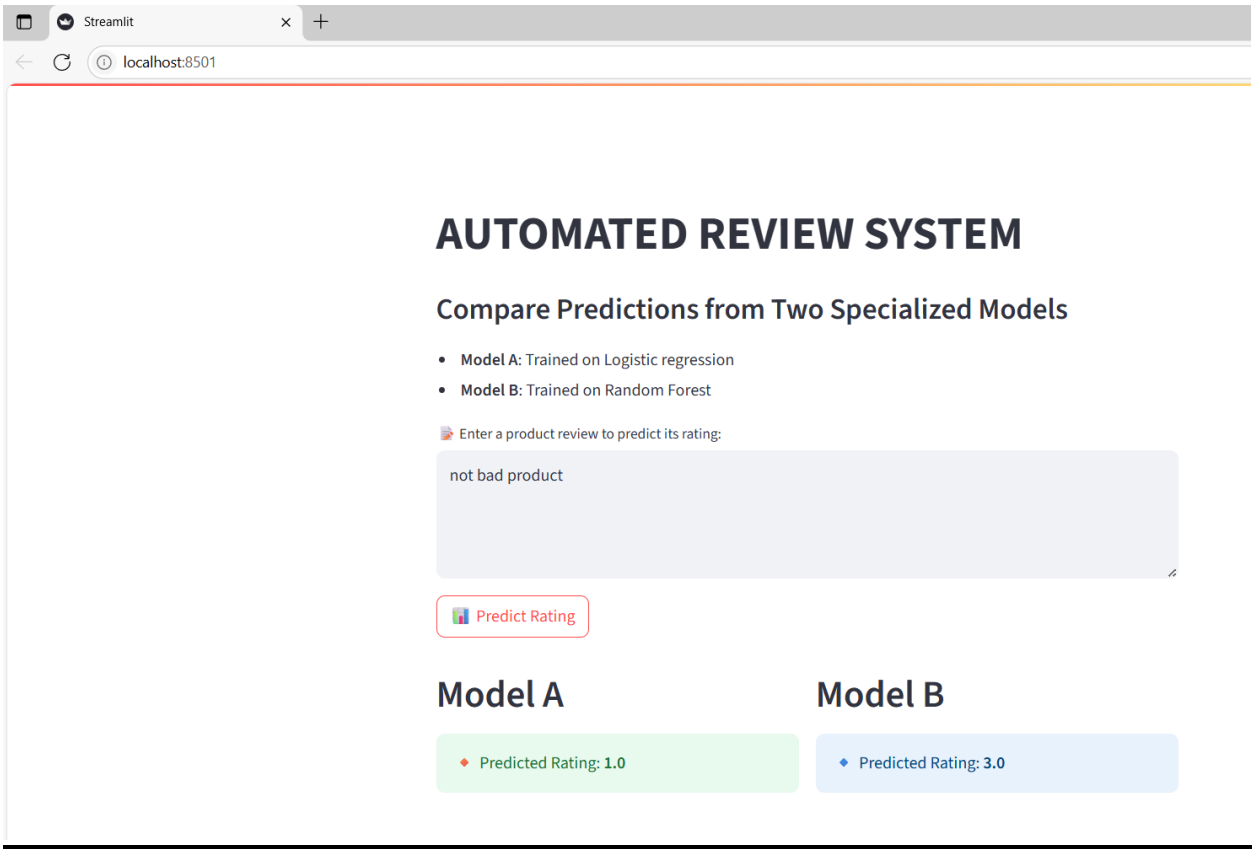
- **User lands on the page**:
    - Sees a title and subheading that set the context.
    - Reads the markdown description to understand the models involved.
- **User enters a product review**:
    - Types/pastes a review into the text area.
- **User clicks " Predict Rating"**:
    - If no input is provided, a warning appears asking for input.
    - If valid input is present, the review is vectorized using each model's corresponding vectorizer.
- **Models make predictions**:
    - Logistic Regression (Model A) and Random Forest (Model B) predict independently.
- **User sees results**:
    - Two columns appear side by side.
    - Each shows the predicted rating with visual styling (green success and blue info box).

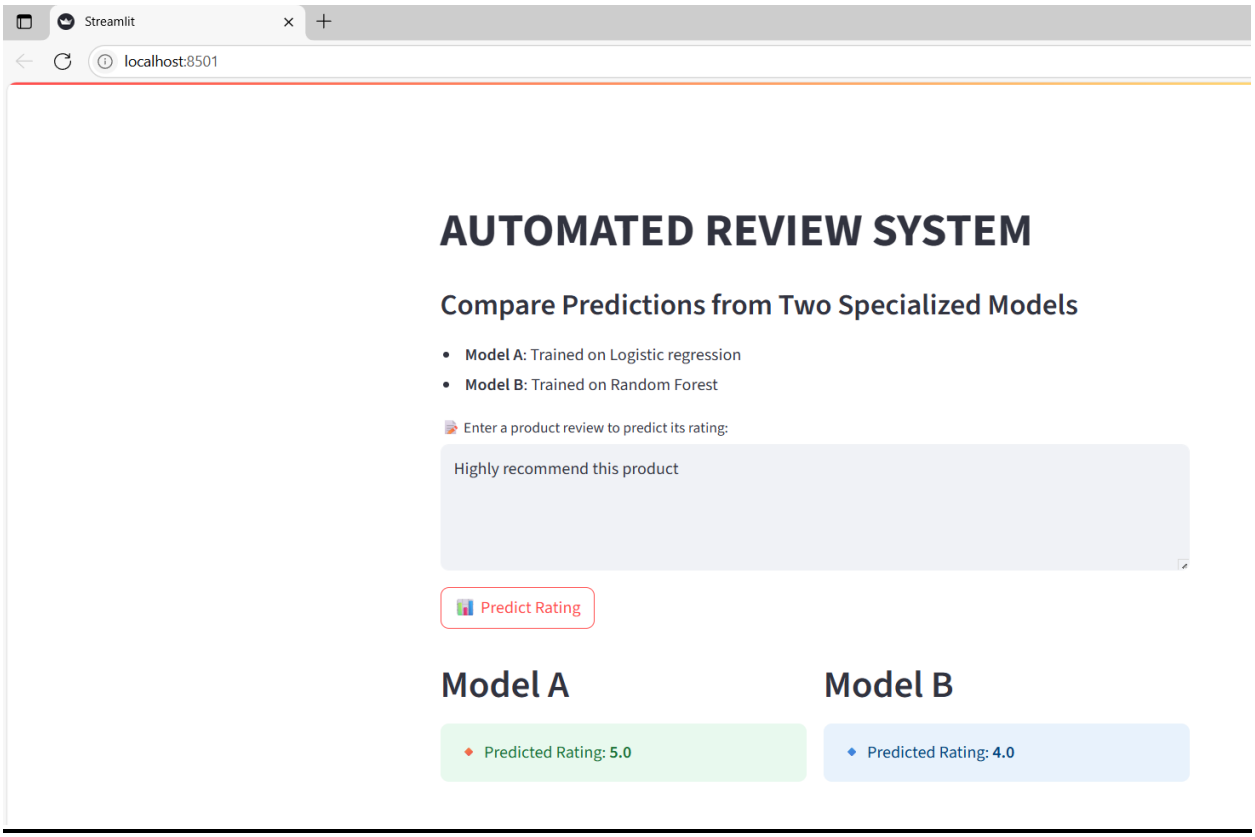
Samples of prediction:

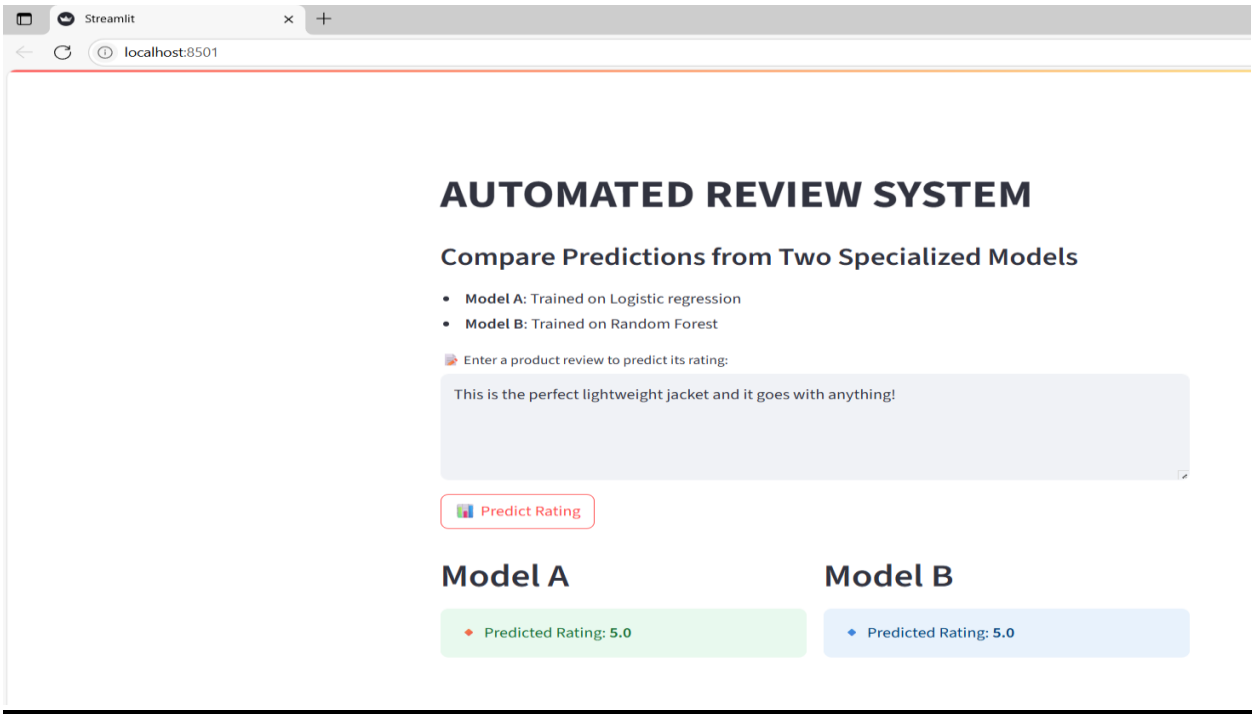





Streamlit
localhost:8501
AUTOMATED REVIEW SYSTEM
Compare Predictions from Two Specialized Models
Model A: Trained on Logistic regression
Model B: Trained on Random Forest
Enter a product review to predict its rating:
bad product
Predict Rating
Model A
Predicted Rating: 1.0
Model B
Predicted Rating: 1.0


Streamlit
localhost:8501
AUTOMATED REVIEW SYSTEM
Compare Predictions from Two Specialized Models
Model A: Trained on Logistic regression
Model B: Trained on Random Forest
Enter a product review to predict its rating:
not bad product
Predict Rating
Model A
Predicted Rating: 1.0
Model B
Predicted Rating: 3.0


Streamlit
localhost:8501
AUTOMATED REVIEW SYSTEM
Compare Predictions from Two Specialized Models
Model A: Trained on Logistic regression
Model B: Trained on Random Forest
Enter a product review to predict its rating:
Highly recommend this product
Predict Rating
Model A
Predicted Rating: 5.0
Model B
Predicted Rating: 4.0


Streamlit
localhost:8501
AUTOMATED REVIEW SYSTEM
Compare Predictions from Two Specialized Models
Model A: Trained on Logistic regression
Model B: Trained on Random Forest
Enter a product review to predict its rating:
This is the perfect lightweight jacket and it goes with anything!
Predict Rating
Model A
Predicted Rating: 5.0
Model B
Predicted Rating: 5.0

Deep Learning Research: Text Classification

1. Deep Learning Models for Text Classification

Model	Description	Use Case Strength
RNN (Recurrent Neural Network)	Processes sequential data word-by-word, remembers previous tokens.	Short sequences, basic sentiment analysis
LSTM (Long Short-Term Memory)	Improved RNN with memory gates; handles long dependencies.	Context-aware sentiment, sequence tagging
GRU (Gated Recurrent Unit)	Simplified LSTM with comparable performance.	Resource-limited environments
CNN (for Text)	Detects local patterns (n-grams) via convolutions.	Sentence classification, spam detection
Bi-LSTM/GRU	Uses both forward and backward context.	Named Entity Recognition, intent classification
Transformer	Self-attention mechanism; models global dependencies.	Long sequences, QA systems, embeddings
BERT / RoBERTa / DistilBERT	Pretrained transformer-based models for text.	Fine-tuned for domain-specific tasks

2. Performance Comparison: Deep Learning vs Traditional ML

Criterion	Traditional ML	Deep Learning
Feature Engineering	Manual (TF-IDF, n-grams)	Automatic (word embeddings, context)
Performance on Large Data	Limited	Improves with more data
Context Understanding	Shallow	Deep context understanding (especially with Transformers)
Training Time	Fast	Slower, requires GPU
Interpretability	Easier	Harder (black-box)
Scalability	Moderate	Scales well with distributed computing
Handling Complex Language Patterns	Poor	Excellent (idioms, sarcasm, context)

3. When Deep Learning Performs Better:

- Large-scale text data (e.g., millions of samples)
- Context-sensitive tasks (e.g., question answering, sentiment shift)
- Domain-specific language modeling (medical, legal)
- Need for capturing long-term dependencies in text
- Deep learning models automatically learn complex features from raw text, eliminating the need for manual feature engineering.
- They understand word meaning in context using attention mechanisms, capturing deeper semantics than traditional ML.

4. Research Findings

1. Pretrained Embeddings Outperform Handcrafted Features

- Models using **pretrained embeddings** (e.g., GloVe, fastText, BERT) consistently outperform models using **TF-IDF** or **bag-of-words**, especially in capturing **semantic meaning**.
- Example: In sentiment classification, pretrained GloVe embeddings improved accuracy by 5–10% over TF-IDF.

2. Transformer Models Handle Long-Range Dependencies Better

- Unlike LSTM/GRU (which suffer from vanishing gradients over long texts), Transformer-based models (e.g., **BERT**, **RoBERTa**) maintain performance across **long sequences**.
- Result: On datasets like **SQuAD** or **Legal Texts**, transformers showed 15–20% better F1 scores.

3. Data Augmentation Boosts Deep Learning Performance

- Techniques like **back translation**, **synonym replacement**, and **EDA (Easy Data Augmentation)** improve DL model generalization.

4. Explainability Tools Are Maturing

- Tools like **LIME**, **SHAP**, **Integrated Gradients**, and **Attention Visualizations** provide insight into model predictions, which helps build trust in deep models.

5. Multilingual Deep Learning Outperforms Rule-Based Translation + Classification

- Models like **mBERT** and **XLM-RoBERTa** support over 100 languages and outperform pipelines involving translation followed by monolingual classification.