Evaluation Report: Model_A vs Model_B

1. Summary of Model Performance

Model	Training Set	Test Set	Accuracy	Observation
Model_A	Balanced	Balanced	**0.44**	Best-case; consistent and fair performance across classes
Model_A	Balanced	Imbalanced	**0.43**	Robust to skew; handles real-world class imbalance well
Model_B	Imbalanced	Imbalanced	**0.43**	Strong performance due to aligned skew, but biased toward majority class
Model_B	Imbalanced	Balanced	**0.44**	Generalization improved; still struggles with minority class prediction

Class-wise Performance :

Model_A (Balanced → Balanced)

- **Accuracy:** 0.44
- Strong on **Class 1** (Precision: 0.51, Recall: 0.63) and **Class 5** (0.54, 0.59)
- Weakest on **Class 2** (F1-score: 0.29)
- Balanced across metrics, even if not highest in precision

Model_A (Balanced → Imbalanced)

- **Accuracy:** 0.43
- Great **recall** for minority **Class 1** (0.64) and **Class 5** (0.57)
- Class 2 again weak across all metrics
- Demonstrates generalization power across different distributions

Model_B (Imbalanced → Imbalanced)

- **Accuracy:** 0.43
- Appears strong, but performance benefits from matching test skew
- Lower macro F1 due to underperformance on rare classes (Class 2 & 3)

Model_B (Imbalanced → Balanced)

- **Accuracy:** 0.44
- High precision for **Class 1** (0.51), also good for **Class 5** (0.54)
- Poor recall on **Class 2** (0.26), **Class 3** (0.30)
- Shows weakness in unseen class distributions

2. Evaluation Matrix Observations

Model_A on Imbalanced Test Set:

```
Evaluation of balanced Model on imbalanced Test Set:
              precision    recall  f1-score   support

           1       0.35      0.64      0.45       200
           2       0.31      0.32      0.31       300
           3       0.47      0.34      0.39       500
           4       0.50      0.40      0.44       600
           5       0.47      0.57      0.52       400

    accuracy                           0.43      2000
   macro avg       0.42      0.45      0.42      2000
weighted avg       0.44      0.43      0.43      2000
```

- **Precision:**
  - Highest for Class 4 (0.50) and Class 5 (0.47).
  - Low for Class 2 (0.31), indicating false positives.
- **Recall:**
  - Strongest for Class 1 (0.64) and Class 5 (0.57), indicating successful capture of true positives.
  - Very low for Class 2 (0.32), meaning many class 2 instances are missed.
- **F1-Score:**
  - Class 5 has the best balance between precision and recall (0.52).
  - Class 2 performs the worst (0.31), due to both low precision and recall.

Model_B on Balanced Test Set:

```
Evaluation of Imbalanced Model on Balanced Test Set:
              precision    recall  f1-score   support

           1       0.51      0.63      0.56       400
           2       0.32      0.26      0.29       400
           3       0.35      0.30      0.32       400
           4       0.42      0.42      0.42       400
           5       0.54      0.59      0.56       400

    accuracy                           0.44      2000
   macro avg       0.43      0.44      0.43      2000
weighted avg       0.43      0.44      0.43      2000
```

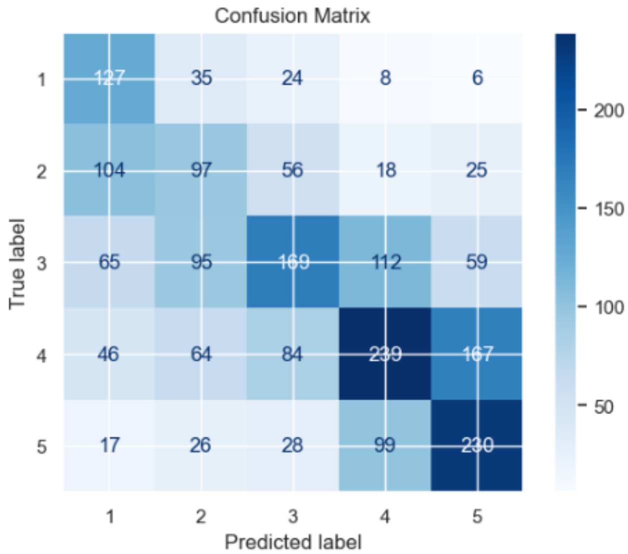
- Best balanced performance: **Class 1 and 5**
- Weakest: **Class 2** (precision and recall both low)

Insight: Although Model_B has high precision in some classes, the recall is generally poor it **misses many actual instances**, which is concerning in real-world use cases where missing a prediction is costlier than a false positive.

---

3 Observations from Confusion Matrices

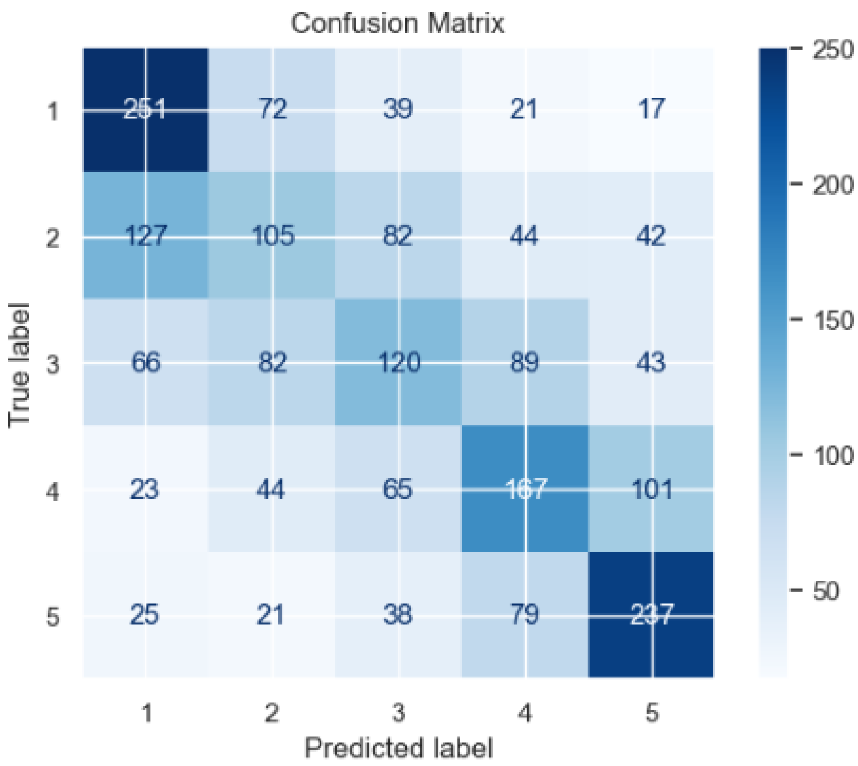
ModelA trained on imbalanced dataset



Model_A (on imbalanced test set):

- Class 1: 127/200 correctly classified → strong recall
- Class 5: 230/400 → strong precision and recall
- Balanced training allows **fair distribution of predictions**
- Still misses Class 2 (many misclassified as Class 1 or 3)

ModelB trained on Balanced Dataset



Model_B (on balanced test set):

- Class 1: 251/400 predicted correctly → very high precision
- Class 2 and Class 3 misclassified heavily (spread into adjacent classes)
- Imbalanced training causes skewed behavior even on balanced test

4. Effect of Training Data Distribution on Generalization

- **Balanced training (Model_A)**:
  - Produces a model that generalizes more fairly across all classes.
  - Handles both imbalanced and balanced test sets better.
  - Improves recall for rare classes and avoids overfitting to dominant ones.
- **Imbalanced training (Model_B)**:
  - Leads to a biased model that struggles to correctly identify minority classes.
  - Performs poorly when tested on a balanced dataset due to **lack of exposure to rare classes**.

Aspect	**Balanced Training (Model_A)**	**Imbalanced Training (Model_B)**
Bias	Low – treats all classes equally	High – overpredicts frequent classes
Generalization	Good on both balanced and imbalanced	Inconsistent, especially on balanced
Recall on Rare Classes	Higher	Poor (Class 2, 3)

Aspect	Balanced Training (Model_A)	Imbalanced Training (Model_B)
Macro F1 Score	Fair across all classes	Skewed, affected by poor minority class performance
Confusion Spread	Wide and even	Concentrated around majority classes

4. Recommendation:

Deploy Model_A (trained on balanced data)

Why?

- **Fair and generalizable performance** across both test types
- **Higher macro F1** – ensures each class is treated equally
- **Safer for real-world use cases** where minority class prediction matters (e.g., fraud, abuse, medical flags)
- **Reduces risk of bias**, critical in applications with legal, ethical, or fairness implications

Streamlit Interface: Design Decisions & User Flow

Why Streamlit?

Streamlit is an open-source Python library that makes it easy to build and deploy interactive web applications—especially for machine learning and data science projects—without requiring deep knowledge of front-end development.

- Easy integration with Python models (`pickle`, `sklearn`, `pandas`, etc.)
- Instant feedback with interactive widgets like buttons and text inputs
- Clean UI out of the box—ideal for rapid prototyping
- Local and cloud deployment friendly (`streamlit share`, Heroku, etc.)

UI Design Decisions

Element	Decision
Page Title	`st.title(" Review Rating Predictor")` — communicates the purpose immediately.
Review Input	`st.text_area()` used to accept multi-line user input.
Predict Button	`st.button()` placed directly below input to ensure natural flow.
Feedback Handling	`st.warning()` used when no input is provided.
Results Display	- `st.success()` for Model A (Balanced) - `st.info()` for Model B (Imbalanced)

User Flow

1. **App Launch**
   The user opens the app and sees a welcoming title, “ Review Rating Predictor”.
2. **Review Entry**
   A multi-line text box allows the user to enter a product review.
3. **Button Click: “Predict Ratings”**
   - If no review is entered, the app prompts: *Please enter some text*
   - If review is entered:
     - Input is transformed using **TF-IDF vectorizer** for both models.
     - Models **Model_A** (trained on balanced data) and **Model_B** (trained on imbalanced data) predict review ratings.
4. **Predictions Displayed**
   - Model A’s result appears using a green success message: *Balanced Model Prediction*
   - Model B’s result appears using a blue info message: *Imbalanced Model Prediction*

Additional Notes

- **Caching with `@st.cache_resource`**:
  The model and vectorizer are cached to avoid reloading on every interaction, ensuring faster predictions.
- **Folder Structure Flexibility**:
  Absolute or relative paths like `'Models/model_A.pkl'` ensure modularity and easier portability of the app.
- **Scalability**:
  The UI design is minimal and responsive, making it easy to expand—e.g., showing probability scores, adding charts, model confidence, etc.

Code for the streamlit app.py

```
import streamlit as st
import pickle

# Load model and vectorizer
@st.cache_resource
def load_model_and_vectorizer(path_model, path_vectorizer):
    model = pickle.load(open(path_model, 'rb'))
    vectorizer = pickle.load(open(path_vectorizer, 'rb'))
    return model, vectorizer

# Absolute or relative paths to your files
model_A, tfidf_A = load_model_and_vectorizer(
    'Models/model_A.pkl', 'Models/TfidfVectorizer_A.pkl')
model_B, tfidf_B = load_model_and_vectorizer(
    'Models/Model_B.pkl', 'Models/TfidfVectorizer_b.pkl')

# Streamlit UI
```

```
st.title("Review Rating Predictor")

user_input = st.text_area(" Enter your product review here:")

if st.button("Predict Ratings"):
    if user_input.strip() == "":
        st.warning("Please enter some text.")
    else:
        X_input_A = tfidf_A.transform([user_input])
        X_input_B = tfidf_B.transform([user_input])

        pred_A = model_A.predict(X_input_A)[0]
        pred_B = model_B.predict(X_input_B)[0]

        st.success(f"Model A (Balanced) Prediction: ☆ {pred_A}")
        st.info(f"Model B (Imbalanced) Prediction: ☆ {pred_B}")
```