🔧 Notebook 3: Nominal Models Final Project - Ordinal vs Nominal Sentiment Analysis

Atharv Chaudhary

Purpose: Train and evaluate NOMINAL classification models.

Models:

1. Multinomial Naive Bayes
2. Logistic Regression (Multinomial)

Input: `amazon_electronics_cleaned.csv`

Output: `nominal_results.csv`, confusion matrices

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dr
```

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score,
    mean_absolute_error,
    f1_score,
    classification_report,
    confusion_matrix
)

# Settings
RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)
plt.style.use('seaborn-v0_8-whitegrid')

print("✅ Libraries imported")
```

```
✅ Libraries imported
```

Step 1: Load Data

```
# Load cleaned data
df = pd.read_csv('/content/drive/MyDrive/Fall 2025/Foundations of Artificia
print(f"✅ Loaded {len(df):,} reviews")

# Show distribution
print("\n📊 Rating Distribution:")
print(df['rating'].value_counts().sort_index())
```

```
✅ Loaded 49,960 reviews

📊 Rating Distribution:
rating
1     2835
2     2161
3     3964
4    10103
5    30897
Name: count, dtype: int64
```

Step 2: Feature Extraction (TF-IDF)

```
# ======================================================================
# TF-IDF FEATURE EXTRACTION
# ======================================================================

print("=" * 70)
print("TF-IDF FEATURE EXTRACTION")
print("=" * 70)

# Configuration
MAX_FEATURES = 5000
NGRAM_RANGE = (1, 2)  # Unigrams and bigrams

vectorizer = TfidfVectorizer(
    max_features=MAX_FEATURES,
    stop_words='english',
    ngram_range=NGRAM_RANGE,
    min_df=5,
    max_df=0.95
)

print(f"\nSettings:")
print(f"   Max features: {MAX_FEATURES}")
print(f"   N-gram range: {NGRAM_RANGE}")

```

```
# Transform
X = vectorizer.fit_transform(df['text'])
y = df['rating'].values

print(f"\n✅ TF-IDF matrix: {X.shape}")
```

```
======================================================================
TF-IDF FEATURE EXTRACTION
======================================================================

Settings:
   Max features: 5000
   N-gram range: (1, 2)

✅ TF-IDF matrix: (49960, 5000)
```

Step 3: Train/Test Split

```
# ======================================================================
# TRAIN/TEST SPLIT
# ======================================================================

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=RANDOM_STATE,
    stratify=y
)

print(f"✅ Training set: {X_train.shape[0]:,} samples")
print(f"✅ Test set: {X_test.shape[0]:,} samples")
```

```
✅ Training set: 39,968 samples
✅ Test set: 9,992 samples
```

Step 4: Helper Functions

```
# ======================================================================
# HELPER FUNCTIONS
# ======================================================================

def evaluate_model(y_true, y_pred, model_name):
```

```
    """Evaluate model and return metrics."""
    accuracy = accuracy_score(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    f1_macro = f1_score(y_true, y_pred, average='macro')
    f1_weighted = f1_score(y_true, y_pred, average='weighted')

    print(f"\n{'='*55}")
    print(f"📊 {model_name}")
    print(f"{'='*55}")
    print(f"Accuracy:      {accuracy:.4f} ({accuracy*100:.2f}%)")
    print(f"MAE:           {mae:.4f}")
    print(f"F1 (macro):    {f1_macro:.4f}")
    print(f"F1 (weighted): {f1_weighted:.4f}")

    return {
        'model': model_name,
        'encoding': 'Nominal',
        'accuracy': accuracy,
        'mae': mae,
        'f1_macro': f1_macro,
        'f1_weighted': f1_weighted
    }


def calculate_error_rates(y_true, y_pred):
    """Calculate adjacent and severe error rates."""
    errors = y_true != y_pred
    if errors.sum() == 0:
        return 0.0, 0.0

    error_distances = np.abs(y_true[errors] - y_pred[errors])
    adjacent = (error_distances == 1).sum() / errors.sum()
    severe = (error_distances >= 2).sum() / errors.sum()

    return adjacent, severe


print("✅ Helper functions defined")
```

```
✅ Helper functions defined
```

Step 5: Model 1 - Multinomial Naive Bayes

```
# ==========================================================================
# MODEL 1: MULTINOMIAL NAIVE BAYES
# ==========================================================================
```

```python

print("\n" + "="*70)
print("🔧 MODEL 1: Multinomial Naive Bayes (Nominal)")
print("="*70)
print("\nTreats classes as UNORDERED categories.")
print("Formula: P(Y=k|x) ∝ P(Y=k) × Π P(xⱼ|Y=k)")

# Train
nb_model = MultinomialNB(alpha=1.0)  # Laplace smoothing
nb_model.fit(X_train, y_train)

# Predict
nb_pred = nb_model.predict(X_test)

# Evaluate
nb_results = evaluate_model(y_test, nb_pred, "Naive Bayes")

# Error analysis
nb_adjacent, nb_severe = calculate_error_rates(y_test, nb_pred)
nb_results['adjacent_error'] = nb_adjacent
nb_results['severe_error'] = nb_severe

print(f"\nError Analysis:")
print(f"   Adjacent Error Rate (±1): {nb_adjacent:.2%}")
```

```
======================================================================
🔧 MODEL 1: Multinomial Naive Bayes (Nominal)
======================================================================

Treats classes as UNORDERED categories.
Formula: P(Y=k|x) ∝ P(Y=k) × Π P(xⱼ|Y=k)

=======================================================
📊 Naive Bayes
=======================================================
Accuracy:      0.6312 (63.12%)
MAE:           0.6651
F1 (macro):    0.2321
F1 (weighted): 0.5129

Error Analysis:
   Adjacent Error Rate (±1): 55.63%
   Severe Error Rate (±2+):  44.37%
```

```python
# Classification report
print("\n📋 Classification Report - Naive Bayes:")
print(classification_report(y_test, nb_pred, digits=4))
```

```
📋 Classification Report - Naive Bayes:
              precision    recall  f1-score   support

           1     0.6236    0.1958    0.2980       567
           2     0.0000    0.0000    0.0000       432
           3     0.3750    0.0038    0.0075       793
           4     0.3797    0.0445    0.0797      2021
           5     0.6378    0.9877    0.7751      6179

    accuracy                         0.6312      9992
   macro avg     0.4032    0.2464    0.2321      9992
weighted avg     0.5364    0.6312    0.5129      9992
```

Step 6: Model 2 - Logistic Regression (Multinomial)

```python
# ============================================================================
# MODEL 2: LOGISTIC REGRESSION (MULTINOMIAL)
# ============================================================================

print("\n" + "="*70)
print("🔧 MODEL 2: Logistic Regression (Nominal - Multinomial)")
print("="*70)
print("\nUses softmax, treats classes as UNORDERED.")
print("Formula: P(Y=k|x) = exp(wₖᵀx + bₖ) / Σ exp(wⱼᵀx + bⱼ)")

# Train
lr_model = LogisticRegression(
    multi_class='multinomial',
    solver='lbfgs',
    max_iter=1000,
    random_state=RANDOM_STATE,
    n_jobs=-1
)
lr_model.fit(X_train, y_train)

# Predict
lr_pred = lr_model.predict(X_test)

# Evaluate
lr_results = evaluate_model(y_test, lr_pred, "Logistic Regression")

# Error analysis
lr_adjacent, lr_severe = calculate_error_rates(y_test, lr_pred)
lr_results['adjacent_error'] = lr_adjacent
lr_results['severe_error'] = lr_severe

print(f"\nError Analysis:")
print(f"   Adjacent Error Rate (±1): {lr_adjacent:.2%}")
print(f"   Severe Error Rate (±2+):  {lr_severe:.2%}")
```

```
======================================================================
🔧 MODEL 2: Logistic Regression (Nominal - Multinomial)
======================================================================

Uses softmax, treats classes as UNORDERED.
Formula: P(Y=k|x) = exp(wₖᵀx + bₖ) / Σ exp(wⱼᵀx + bⱼ)

============================================================
📊 Logistic Regression
============================================================
Accuracy:      0.6595 (65.95%)
MAE:           0.5337
```

```
F1 (macro):    0.3793
F1 (weighted): 0.6064

Error Analysis:
   Adjacent Error Rate (±1): 65.17%
   Severe Error Rate (±2+):  34.83%
```

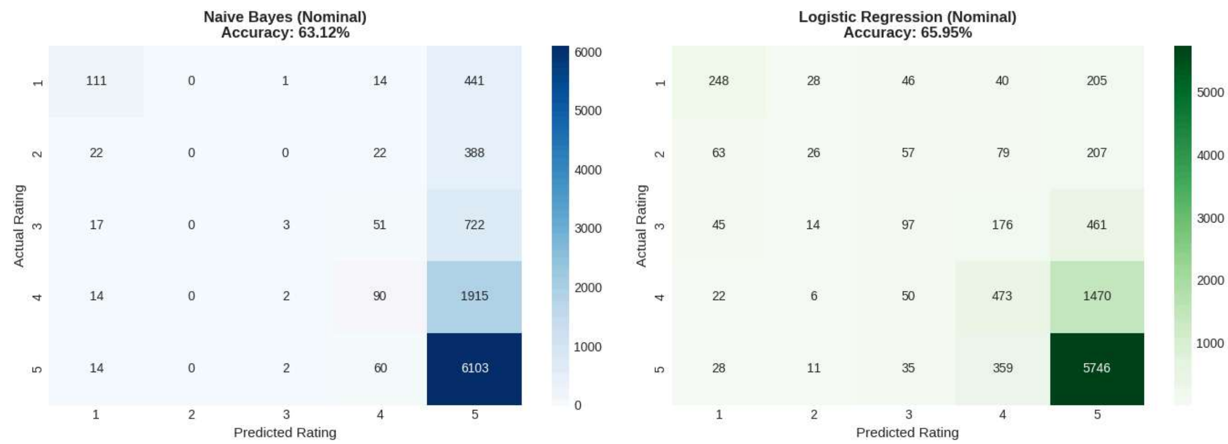
```
# Classification report
print("\n📋 Classification Report - Logistic Regression:")
print(classification_report(y_test, lr_pred, digits=4))
```

```
📋 Classification Report - Logistic Regression:
              precision    recall  f1-score   support

           1     0.6108    0.4374    0.5098       567
           2     0.3059    0.0602    0.1006       432
           3     0.3404    0.1223    0.1800       793
           4     0.4197    0.2340    0.3005      2021
           5     0.7103    0.9299    0.8054      6179

    accuracy                         0.6595      9992
   macro avg     0.4774    0.3568    0.3793      9992
weighted avg     0.5991    0.6595    0.6064      9992
```

Step 7: Confusion Matrices

```
# ==============================================================================
# CONFUSION MATRICES
# ==============================================================================

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Naive Bayes
cm_nb = confusion_matrix(y_test, nb_pred)
sns.heatmap(cm_nb, annot=True, fmt='d', cmap='Blues', ax=axes[0],
            xticklabels=[1, 2, 3, 4, 5], yticklabels=[1, 2, 3, 4, 5])
axes[0].set_xlabel('Predicted Rating', fontsize=11)
axes[0].set_ylabel('Actual Rating', fontsize=11)
axes[0].set_title(f'Naive Bayes (Nominal)\nAccuracy: {nb_results["accuracy
                  fontsize=12, fontweight='bold')

# Logistic Regression
cm_lr = confusion_matrix(y_test, lr_pred)
sns.heatmap(cm_lr, annot=True, fmt='d', cmap='Greens', ax=axes[1],
            xticklabels=[1, 2, 3, 4, 5], yticklabels=[1, 2, 3, 4, 5])
axes[1].set_xlabel('Predicted Rating', fontsize=11)
axes[1].set_ylabel('Actual Rating', fontsize=11)
axes[1].set_title(f'Logistic Regression (Nominal)\nAccuracy: {lr_results['
                  fontsize=12, fontweight='bold')
```

```

plt.tight_layout()
plt.savefig('confusion_matrices_nominal.png', dpi=150, bbox_inches='tight
plt.show()

print("\n✅ Saved: confusion_matrices_nominal.png")
```



✅ Saved: confusion_matrices_nominal.png

Step 8: Save Results

```
# ========================================================================
# SAVE RESULTS
# ========================================================================

# Combine results
nominal_results = pd.DataFrame([nb_results, lr_results])

print("\n" + "="*70)
print("📊 NOMINAL MODELS SUMMARY")
print("="*70)
print(nominal_results.to_string(index=False))

# Save to CSV
nominal_results.to_csv('nominal_results.csv', index=False)
print("\n✅ Saved: nominal_results.csv")
```

```
======================================================================
📊 NOMINAL MODELS SUMMARY
======================================================================
              model encoding  accuracy      mae  f1_macro  f1_weighted  adjacent
        Naive Bayes  Nominal  0.631205 0.665132  0.232055     0.512934         0.
Logistic Regression  Nominal  0.659528 0.533727  0.379251     0.606418         0.

✅ Saved: nominal_results.csv
```

```python
# Save predictions for later analysis
predictions_df = pd.DataFrame({
    'actual': y_test,
    'nb_pred': nb_pred,
    'lr_pred': lr_pred
})
predictions_df.to_csv('nominal_predictions.csv', index=False)
print("✅ Saved: nominal_predictions.csv")
```

```
✅ Saved: nominal_predictions.csv
```

```python
# Download files
try:
    from google.colab import files
    files.download('nominal_results.csv')
    files.download('confusion_matrices_nominal.png')
except:
    print("Files saved locally")
```

```python
import shutil
import os

drive_path = '/content/drive/MyDrive/Fall 2025/Foundations of Artificial Intelligence/Final Project/data'

# Create the directory if it doesn't exist
os.makedirs(drive_path, exist_ok=True)

files_to_save = [
    'nominal_results.csv',
    'confusion_matrices_nominal.png',
    'nominal_predictions.csv'
]

for file_name in files_to_save:
    source_path = os.path.join('/content', file_name)
    destination_path = os.path.join(drive_path, file_name)
    try:
```

```
            shutil.copy(source_path, destination_path)
            print(f"✅ Saved '{file_name}' to Drive: {destination_path}")
        except FileNotFoundError:
            print(f"⚠️ Error: '{file_name}' not found. Skipping.")
        except Exception as e:
            print(f"❌ Error saving '{file_name}' to Drive: {e}")
```

```
✅ Saved 'nominal_results.csv' to Drive: /content/drive/MyDrive/Fall 2025/Founda
✅ Saved 'confusion_matrices_nominal.png' to Drive: /content/drive/MyDrive/Fall
✅ Saved 'nominal_predictions.csv' to Drive: /content/drive/MyDrive/Fall 2025/Fo
```

✅ Summary