CODE

1. import pandas as pd

2. import numpy as np

3. import os

4. from datetime import datetime

5. import matplotlib.pyplot as plt

6. import seaborn as sns

7. from sklearn.preprocessing import LabelEncoder, StandardScaler

8. from sklearn.model\_selection import train\_test\_split, GridSearchCV

9. from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

10. from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, StackingClassifier

11. from sklearn.linear\_model import LogisticRegression

12. from sklearn.svm import SVC

13. from sklearn.neural\_network import MLPClassifier

14. from sklearn.decomposition import PCA

15. from imblearn.over\_sampling import SMOTE

16. import joblib

17. import warnings

18. warnings.filterwarnings('ignore')

19.

20. # Set up styling for visualizations

21. sns.set(style="whitegrid")

22. plt.rcParams['figure.figsize'] = (12, 8)

23.

24. # Create output directory for results

25. timestamp = datetime.now().strftime('%Y%m%d\_%H%M%S')

26. output\_dir = f'analysis\_results\_{timestamp}'

27. os.makedirs(output\_dir, exist\_ok=True)

28. print(f"Created output directory: {output\_dir}")

29.

30. print("="\*80)

31. print("ENHANCED COUNTRY & QUESTION LEVEL BALANCED CLASSIFICATION ANALYSIS")

32. print("="\*80)

33.

34. # STEP 1: Load Dataset

35. print("\n[1] LOADING DATASET...")

36. # Replace with your actual file path

37. df = pd.read\_csv(r"C:\Users\anand\Downloads\MathE dataset.csv", encoding='ISO-8859-1')

38. df = df.drop(columns=[col for col in df.columns if 'Unnamed' in col])

39.

40. print(f"Dataset shape: {df.shape}")

41. print("\nClass distribution (Type of Answer):")

42. print(df['Type of Answer'].value\_counts())

43.

44. # Save initial class distribution

45. plt.figure(figsize=(10, 6))

46. sns.countplot(x='Type of Answer', data=df, palette='viridis')

47. plt.title('Initial Class Distribution', fontsize=15)

48. plt.savefig(f'{output\_dir}/01\_initial\_class\_distribution.png', bbox\_inches='tight')

49. plt.close()

50.

51. # STEP 2: Data Preprocessing

52. print("\n[2] DATA PREPROCESSING...")

53.

54. # Convert categorical columns

55. label\_cols = ['Topic', 'Subtopic', 'Question ID', 'Question Level', 'Student Country']

56. encoders = {}

57.

58. for col in label\_cols:

59.     if col in df.columns:

60.         le = LabelEncoder()

61.         df[col] = le.fit\_transform(df[col].astype(str))

62.         encoders[col] = le

63.         print(f"Encoded {col} - {len(le.classes\_)} unique values")

64.

65. # ENHANCED FEATURE ENGINEERING

66. print("\nCreating ENHANCED features...")

67.

68. # Basic performance features

69. student\_avg = df.groupby('Student ID')['Type of Answer'].mean()

70. df['Student\_Avg\_Performance'] = df['Student ID'].map(student\_avg)

71.

72. question\_avg = df.groupby('Question ID')['Type of Answer'].mean()

73. df['Question\_Difficulty'] = df['Question ID'].map(question\_avg)

74.

75. country\_avg = df.groupby('Student Country')['Type of Answer'].mean()

76. df['Country\_Avg\_Performance'] = df['Student Country'].map(country\_avg)

77.

78. topic\_avg = df.groupby('Topic')['Type of Answer'].mean()

79. df['Topic\_Avg\_Performance'] = df['Topic'].map(topic\_avg)

80.

81. subtopic\_avg = df.groupby('Subtopic')['Type of Answer'].mean()

82. df['Subtopic\_Avg\_Performance'] = df['Subtopic'].map(subtopic\_avg)

83.

84. # Advanced features

85. # Per-student performance by topic

86. student\_topic\_perf = df.groupby(['Student ID', 'Topic'])['Type of Answer'].mean().reset\_index()

87. student\_topic\_dict = dict(zip(zip(student\_topic\_perf['Student ID'], student\_topic\_perf['Topic']),

88.                               student\_topic\_perf['Type of Answer']))

89. df['Student\_Topic\_Performance'] = df.apply(lambda x: student\_topic\_dict.get((x['Student ID'], x['Topic']),

90.                                                                              df['Student\_Avg\_Performance'].mean()), axis=1)

91.

92. # Per-student performance by question level

93. student\_level\_perf = df.groupby(['Student ID', 'Question Level'])['Type of Answer'].mean().reset\_index()

94. student\_level\_dict = dict(zip(zip(student\_level\_perf['Student ID'], student\_level\_perf['Question Level']),

95.                               student\_level\_perf['Type of Answer']))

96. df['Student\_Level\_Performance'] = df.apply(lambda x: student\_level\_dict.get((x['Student ID'], x['Question Level']),

97.                                                                              df['Student\_Avg\_Performance'].mean()), axis=1)

98.

99. # Country performance by topic

100. country\_topic\_perf = df.groupby(['Student Country', 'Topic'])['Type of Answer'].mean().reset\_index()

101. country\_topic\_dict = dict(zip(zip(country\_topic\_perf['Student Country'], country\_topic\_perf['Topic']),

102.                               country\_topic\_perf['Type of Answer']))

103. df['Country\_Topic\_Performance'] = df.apply(lambda x: country\_topic\_dict.get((x['Student Country'], x['Topic']),

104.                                                                              df['Country\_Avg\_Performance'].mean()), axis=1)

105.

106. # Country performance by question level

107. country\_level\_perf = df.groupby(['Student Country', 'Question Level'])['Type of Answer'].mean().reset\_index()

108. country\_level\_dict = dict(zip(zip(country\_level\_perf['Student Country'], country\_level\_perf['Question Level']),

109.                               country\_level\_perf['Type of Answer']))

110. df['Country\_Level\_Performance'] = df.apply(lambda x: country\_level\_dict.get((x['Student Country'], x['Question Level']),

111.                                                                              df['Country\_Avg\_Performance'].mean()), axis=1)

112.

113. # Interaction features

114. df['Difficulty\_Level\_Interaction'] = df['Question\_Difficulty'] \* df['Question Level']

115. df['Student\_Country\_Interaction'] = df['Student\_Avg\_Performance'] \* df['Country\_Avg\_Performance']

116. df['Topic\_Subtopic\_Interaction'] = df['Topic\_Avg\_Performance'] \* df['Subtopic\_Avg\_Performance']

117.

118. # Check for missing values and handle them

119. missing\_values = df.isnull().sum()

120. columns\_with\_missing = missing\_values[missing\_values > 0]

121. if len(columns\_with\_missing) > 0:

122.     for col in columns\_with\_missing.index:

123.         if df[col].dtype in ['float64', 'int64']:

124.             df[col].fillna(df[col].mean(), inplace=True)

125.         else:

126.             df[col].fillna(df[col].mode()[0], inplace=True)

127.

128. # STEP 3: Split data for initial evaluation

129. X = df.select\_dtypes(include=['number'])

130. X = X.drop(['Type of Answer', 'Student ID'], axis=1, errors='ignore')

131. y = df['Type of Answer']

132.

133. # Standardize features

134. scaler = StandardScaler()

135. X\_scaled = scaler.fit\_transform(X)

136. X\_scaled\_df = pd.DataFrame(X\_scaled, columns=X.columns)

137.

138. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled\_df, y, test\_size=0.2, random\_state=42, stratify=y)

139.

140. # STEP 4: Initial model evaluation (before SMOTE)

141. print("\n[3] EVALUATING MODELS BEFORE BALANCING...")

142.

143. # Random Forest with optimized parameters

144. rf = RandomForestClassifier(n\_estimators=200, max\_depth=15, min\_samples\_split=2,

145.                           min\_samples\_leaf=1, random\_state=42, class\_weight='balanced')

146. rf.fit(X\_train, y\_train)

147. rf\_pred = rf.predict(X\_test)

148. rf\_accuracy = accuracy\_score(y\_test, rf\_pred)

149. print(f"Random Forest Accuracy (Before Balancing): {rf\_accuracy:.4f}")

150. print(classification\_report(y\_test, rf\_pred))

151.

152. # Gradient Boosting

153. gb = GradientBoostingClassifier(n\_estimators=200, learning\_rate=0.1, max\_depth=5, random\_state=42)

154. gb.fit(X\_train, y\_train)

155. gb\_pred = gb.predict(X\_test)

156. gb\_accuracy = accuracy\_score(y\_test, gb\_pred)

157. print(f"Gradient Boosting Accuracy (Before Balancing): {gb\_accuracy:.4f}")

158. print(classification\_report(y\_test, gb\_pred))

159.

160. # Neural Network

161. nn = MLPClassifier(hidden\_layer\_sizes=(100, 50, 25), max\_iter=300, activation='relu',

162.                    solver='adam', random\_state=42, early\_stopping=True)

163. nn.fit(X\_train, y\_train)

164. nn\_pred = nn.predict(X\_test)

165. nn\_accuracy = accuracy\_score(y\_test, nn\_pred)

166. print(f"Neural Network Accuracy (Before Balancing): {nn\_accuracy:.4f}")

167. print(classification\_report(y\_test, nn\_pred))

168.

169. # Stacking Ensemble

170. base\_models = [

171.     ('rf', RandomForestClassifier(n\_estimators=200, max\_depth=15, random\_state=42)),

172.     ('gb', GradientBoostingClassifier(n\_estimators=200, learning\_rate=0.1, max\_depth=5, random\_state=42)),

173.     ('nn', MLPClassifier(hidden\_layer\_sizes=(50, 25), max\_iter=300, early\_stopping=True, random\_state=42))

174. ]

175. stacking = StackingClassifier(

176.     estimators=base\_models,

177.     final\_estimator=LogisticRegression(C=10, max\_iter=1000),

178.     cv=5

179. )

180. stacking.fit(X\_train, y\_train)

181. stacking\_pred = stacking.predict(X\_test)

182. stacking\_accuracy = accuracy\_score(y\_test, stacking\_pred)

183. print(f"Stacking Ensemble Accuracy (Before Balancing): {stacking\_accuracy:.4f}")

184. print(classification\_report(y\_test, stacking\_pred))

185.

186. # STEP 5: Enhanced SMOTE-like resampling by country - ALL BALANCED TO 3001

187. print("\n[4] APPLYING ENHANCED UNIFORM BALANCING (ALL TO 3001)...")

188.

189. MAX\_TARGET\_COUNT = 3001  # Setting all countries to have 3001 samples per class

190. resampled\_data = []

191. country\_level\_stats = []

192.

193. # Process each country

194. for country in df['Student Country'].unique():

195.     country\_name = encoders['Student Country'].inverse\_transform([country])[0] if 'Student Country' in encoders else f"Country\_{country}"

196.     print(f"\nProcessing country: {country\_name}")

197.

198.     # Filter data for this country

199.     country\_df = df[df['Student Country'] == country]

200.

201.     # Get class counts

202.     class\_counts = country\_df['Type of Answer'].value\_counts().to\_dict()

203.     class\_0\_before = class\_counts.get(0, 0)

204.     class\_1\_before = class\_counts.get(1, 0)

205.

206.     # Set target count to the maximum value 3001

207.     target\_count = MAX\_TARGET\_COUNT

208.     print(f"Class 0 Before: {class\_0\_before}, Class 1 Before: {class\_1\_before}")

209.     print(f"Target count for all classes: {target\_count}")

210.

211.     # Create synthetic samples

212.     resampled\_country\_df = pd.DataFrame()  # Start fresh

213.

214.     # For Class 0

215.     if class\_0\_before > 0:

216.         class\_0\_df = country\_df[country\_df['Type of Answer'] == 0]

217.         synthetic\_needed = target\_count

218.         # Use SMOTE-like approach for small datasets

219.         if class\_0\_before < 5:

220.             # Create copies with small variations

221.             synthetic\_samples = pd.DataFrame()

222.             for \_ in range(synthetic\_needed):

223.                 sample = class\_0\_df.sample(1, replace=True, random\_state=np.random.randint(1000))

224.                 # Add small random noise to numeric columns

225.                 for col in sample.select\_dtypes(include=['number']).columns:

226.                     if col != 'Type of Answer' and col != 'Student ID':

227.                         noise = np.random.normal(0, 0.05, 1)[0]

228.                         sample[col] = sample[col] + noise

229.                 synthetic\_samples = pd.concat([synthetic\_samples, sample])

230.         else:

231.             # If we have enough samples, use sampling with replacement

232.             synthetic\_samples = class\_0\_df.sample(synthetic\_needed, replace=True, random\_state=42)

233.

234.         resampled\_country\_df = pd.concat([resampled\_country\_df, synthetic\_samples])

235.

236.     # For Class 1

237.     if class\_1\_before > 0:

238.         class\_1\_df = country\_df[country\_df['Type of Answer'] == 1]

239.         synthetic\_needed = target\_count

240.         # Use SMOTE-like approach for small datasets

241.         if class\_1\_before < 5:

242.             # Create copies with small variations

243.             synthetic\_samples = pd.DataFrame()

244.             for \_ in range(synthetic\_needed):

245.                 sample = class\_1\_df.sample(1, replace=True, random\_state=np.random.randint(1000))

246.                 # Add small random noise to numeric columns

247.                 for col in sample.select\_dtypes(include=['number']).columns:

248.                     if col != 'Type of Answer' and col != 'Student ID':

249.                         noise = np.random.normal(0, 0.05, 1)[0]

250.                         sample[col] = sample[col] + noise

251.                 synthetic\_samples = pd.concat([synthetic\_samples, sample])

252.         else:

253.             # If we have enough samples, use sampling with replacement

254.             synthetic\_samples = class\_1\_df.sample(synthetic\_needed, replace=True, random\_state=42)

255.

256.         resampled\_country\_df = pd.concat([resampled\_country\_df, synthetic\_samples])

257.

258.     # Add to resampled data

259.     resampled\_data.append(resampled\_country\_df)

260.

261.     # Calculate stats

262.     class\_counts\_after = resampled\_country\_df['Type of Answer'].value\_counts().to\_dict()

263.     class\_0\_after = class\_counts\_after.get(0, 0)

264.     class\_1\_after = class\_counts\_after.get(1, 0)

265.

266.     synthetic\_samples = len(resampled\_country\_df) - len(country\_df)

267.

268.     # Record statistics

269.     country\_level\_stats.append({

270.         'Country': country\_name,

271.         'Class\_0\_Before': class\_0\_before,

272.         'Class\_1\_Before': class\_1\_before,

273.         'Class\_0\_After': class\_0\_after,

274.         'Class\_1\_After': class\_1\_after,

275.         'Synthetic\_Samples': synthetic\_samples

276.     })

277.

278.     print(f"After resampling - Class 0: {class\_0\_after}, Class 1: {class\_1\_after}, Synthetic: {synthetic\_samples}")

279.

280. # Combine all resampled data

281. df\_resampled = pd.concat(resampled\_data, ignore\_index=True)

282. print(f"\nCombined resampled dataset shape: {df\_resampled.shape}")

283. print("Final class distribution after resampling:")

284. print(df\_resampled['Type of Answer'].value\_counts())

285.

286. # Save resampled dataset and stats

287. df\_resampled.to\_csv(f'{output\_dir}/MathE\_uniform\_balanced\_resampled.csv', index=False)

288. stats\_df = pd.DataFrame(country\_level\_stats)

289. stats\_df.to\_csv(f'{output\_dir}/country\_uniform\_balance\_stats.csv', index=False)

290.

291. # Display statistics in a nice table

292. print("\nResampling Statistics by Country:")

293. print(stats\_df[['Country', 'Class\_0\_Before', 'Class\_1\_Before', 'Class\_0\_After', 'Class\_1\_After', 'Synthetic\_Samples']])

294.

295. # STEP 6: Train models on enhanced balanced data with hyperparameter tuning

296. print("\n[5] TRAINING OPTIMIZED MODELS ON BALANCED DATA...")

297.

298. X\_resampled = df\_resampled.select\_dtypes(include=['number'])

299. X\_resampled = X\_resampled.drop(['Type of Answer', 'Student ID'], axis=1, errors='ignore')

300. y\_resampled = df\_resampled['Type of Answer']

301.

302. # Standardize features

303. scaler\_resampled = StandardScaler()

304. X\_resampled\_scaled = scaler\_resampled.fit\_transform(X\_resampled)

305. X\_resampled\_scaled\_df = pd.DataFrame(X\_resampled\_scaled, columns=X\_resampled.columns)

306.

307. X\_train\_resampled, X\_test\_resampled, y\_train\_resampled, y\_test\_resampled = train\_test\_split(

308.     X\_resampled\_scaled\_df, y\_resampled, test\_size=0.2, random\_state=42, stratify=y\_resampled

309. )

310.

311. # Create PCA features to capture additional patterns

312. pca = PCA(n\_components=5)

313. pca\_features = pca.fit\_transform(X\_train\_resampled)

314. X\_train\_pca = np.hstack((X\_train\_resampled, pca\_features))

315. pca\_features\_test = pca.transform(X\_test\_resampled)

316. X\_test\_pca = np.hstack((X\_test\_resampled, pca\_features\_test))

317.

318. # Optimized Random Forest

319. rf\_optimized = RandomForestClassifier(

320.     n\_estimators=500,

321.     max\_depth=20,

322.     min\_samples\_split=2,

323.     min\_samples\_leaf=1,

324.     bootstrap=True,

325.     max\_features='sqrt',

326.     random\_state=42

327. )

328. rf\_optimized.fit(X\_train\_pca, y\_train\_resampled)

329. rf\_optimized\_pred = rf\_optimized.predict(X\_test\_pca)

330. rf\_optimized\_accuracy = accuracy\_score(y\_test\_resampled, rf\_optimized\_pred)

331.

332. print(f"Optimized Random Forest Accuracy (After Balancing): {rf\_optimized\_accuracy:.4f}")

333. print(classification\_report(y\_test\_resampled, rf\_optimized\_pred))

334.

335. # Optimized Gradient Boosting

336. gb\_optimized = GradientBoostingClassifier(

337.     n\_estimators=300,

338.     learning\_rate=0.05,

339.     max\_depth=8,

340.     min\_samples\_split=4,

341.     min\_samples\_leaf=2,

342.     subsample=0.8,

343.     random\_state=42

344. )

345. gb\_optimized.fit(X\_train\_pca, y\_train\_resampled)

346. gb\_optimized\_pred = gb\_optimized.predict(X\_test\_pca)

347. gb\_optimized\_accuracy = accuracy\_score(y\_test\_resampled, gb\_optimized\_pred)

348.

349. print(f"Optimized Gradient Boosting Accuracy (After Balancing): {gb\_optimized\_accuracy:.4f}")

350. print(classification\_report(y\_test\_resampled, gb\_optimized\_pred))

351.

352. # Neural Network with optimizations

353. nn\_optimized = MLPClassifier(

354.     hidden\_layer\_sizes=(200, 100, 50),

355.     activation='relu',

356.     solver='adam',

357.     alpha=0.0001,

358.     batch\_size=128,

359.     learning\_rate='adaptive',

360.     max\_iter=500,

361.     early\_stopping=True,

362.     random\_state=42

363. )

364. nn\_optimized.fit(X\_train\_pca, y\_train\_resampled)

365. nn\_optimized\_pred = nn\_optimized.predict(X\_test\_pca)

366. nn\_optimized\_accuracy = accuracy\_score(y\_test\_resampled, nn\_optimized\_pred)

367.

368. print(f"Optimized Neural Network Accuracy (After Balancing): {nn\_optimized\_accuracy:.4f}")

369. print(classification\_report(y\_test\_resampled, nn\_optimized\_pred))

370.

371. # Enhanced Stacking Ensemble with optimized base models

372. base\_models\_optimized = [

373.     ('rf', RandomForestClassifier(n\_estimators=500, max\_depth=20, random\_state=42)),

374.     ('gb', GradientBoostingClassifier(n\_estimators=300, learning\_rate=0.05, max\_depth=8, random\_state=42)),

375.     ('nn', MLPClassifier(hidden\_layer\_sizes=(200, 100, 50), max\_iter=500, early\_stopping=True, random\_state=42)),

376.     ('svm', SVC(C=10, gamma='scale', probability=True, random\_state=42))

377. ]

378.

379. stacking\_optimized = StackingClassifier(

380.     estimators=base\_models\_optimized,

381.     final\_estimator=LogisticRegression(C=100, solver='liblinear', max\_iter=2000),

382.     cv=5

383. )

384. stacking\_optimized.fit(X\_train\_pca, y\_train\_resampled)

385. stacking\_optimized\_pred = stacking\_optimized.predict(X\_test\_pca)

386. stacking\_optimized\_accuracy = accuracy\_score(y\_test\_resampled, stacking\_optimized\_pred)

387.

388. print(f"Enhanced Stacking Ensemble Accuracy (After Balancing): {stacking\_optimized\_accuracy:.4f}")

389. print(classification\_report(y\_test\_resampled, stacking\_optimized\_pred))

390.

391. # STEP 7: Feature importance analysis

392. feature\_importance = pd.DataFrame({

393.     'Feature': X\_resampled.columns,

394.     'Importance': rf\_optimized.feature\_importances\_[:len(X\_resampled.columns)]

395. }).sort\_values('Importance', ascending=False)

396.

397. print("\nTop 10 Most Important Features:")

398. print(feature\_importance.head(10))

399.

400. # Visualize feature importance

401. plt.figure(figsize=(12, 8))

402. sns.barplot(x='Importance', y='Feature', data=feature\_importance.head(10), palette='viridis')

403. plt.title('Feature Importance After Balancing', fontsize=16)

404. plt.tight\_layout()

405. plt.savefig(f'{output\_dir}/feature\_importance.png', bbox\_inches='tight')

406. plt.close()

407.

408. # STEP 8: Compare results before and after balancing

409. comparison = pd.DataFrame({

410.     'Model': ['Random Forest', 'Gradient Boosting', 'Neural Network', 'Stacking Ensemble'],

411.     'Accuracy Before Balancing': [rf\_accuracy, gb\_accuracy, nn\_accuracy, stacking\_accuracy],

412.     'Accuracy After Balancing': [rf\_optimized\_accuracy, gb\_optimized\_accuracy, nn\_optimized\_accuracy, stacking\_optimized\_accuracy],

413.     'Improvement': [rf\_optimized\_accuracy - rf\_accuracy,

414.                    gb\_optimized\_accuracy - gb\_accuracy,

415.                    nn\_optimized\_accuracy - nn\_accuracy,

416.                    stacking\_optimized\_accuracy - stacking\_accuracy]

417. })

418.

419. print("\nAccuracy Comparison:")

420. print(comparison)

421.

422. # Save models

423. joblib.dump(rf\_optimized, f'{output\_dir}/optimized\_random\_forest\_model.pkl')

424. joblib.dump(gb\_optimized, f'{output\_dir}/optimized\_gradient\_boosting\_model.pkl')

425. joblib.dump(nn\_optimized, f'{output\_dir}/optimized\_neural\_network\_model.pkl')

426. joblib.dump(stacking\_optimized, f'{output\_dir}/optimized\_stacking\_ensemble\_model.pkl')

427. joblib.dump(scaler\_resampled, f'{output\_dir}/feature\_scaler.pkl')

428. joblib.dump(pca, f'{output\_dir}/pca\_transformer.pkl')

429.

430. # Create summary visualization for presentation

431. plt.figure(figsize=(12, 8))

432. comparison\_melted = pd.melt(comparison, id\_vars=['Model'],

433.                             value\_vars=['Accuracy Before Balancing', 'Accuracy After Balancing'])

434. sns.barplot(x='Model', y='value', hue='variable', data=comparison\_melted, palette='Set2')

435. plt.title('Model Accuracy Before and After Balancing & Optimization', fontsize=16)

436. plt.ylabel('Accuracy')

437. plt.ylim(0.6, 1.0)  # Set y-axis to focus on the accuracy range

438. plt.tight\_layout()

439. plt.savefig(f'{output\_dir}/accuracy\_comparison.png', bbox\_inches='tight')

440. plt.close()

441.

442. # Create visualization of country statistics

443. plt.figure(figsize=(15, 10))

444. stats\_plot = stats\_df.sort\_values('Class\_0\_Before', ascending=False)

445. x = np.arange(len(stats\_plot))

446. width = 0.2

447.

448. fig, ax = plt.subplots(figsize=(15, 8))

449. ax.bar(x - width\*1.5, stats\_plot['Class\_0\_Before'], width, label='Class 0 Before', color='#1f77b4')

450. ax.bar(x - width/2, stats\_plot['Class\_1\_Before'], width, label='Class 1 Before', color='#ff7f0e')

451. ax.bar(x + width/2, stats\_plot['Class\_0\_After'], width, label='Class 0 After', color='#2ca02c')

452. ax.bar(x + width\*1.5, stats\_plot['Class\_1\_After'], width, label='Class 1 After', color='#d62728')

453.

454. ax.set\_ylabel('Number of Samples')

455. ax.set\_title('Class Distribution by Country Before and After Uniform Balancing')

456. ax.set\_xticks(x)

457. ax.set\_xticklabels(stats\_plot['Country'], rotation=45, ha='right')

458. ax.legend()

459.

460. plt.tight\_layout()

461. plt.savefig(f'{output\_dir}/country\_class\_distribution.png', bbox\_inches='tight')

462. plt.close()

463.

464. print("\nEnhanced analysis complete! Results saved to:", output\_dir)

465.

OUTPUT

1. Created output directory: analysis\_results\_20250417\_222650

2. ================================================================================

3. ENHANCED COUNTRY & QUESTION LEVEL BALANCED CLASSIFICATION ANALYSIS

4. ================================================================================

5.

6. [1] LOADING DATASET...

7. Dataset shape: (9546, 8)

8.

9. Class distribution (Type of Answer):

10. Type of Answer

11. 0 5076

12. 1 4470

13. Name: count, dtype: int64

14.

15. [2] DATA PREPROCESSING...

16. Encoded Topic - 14 unique values

17. Encoded Subtopic - 24 unique values

18. Encoded Question ID - 833 unique values

19. Encoded Question Level - 2 unique values

20. Encoded Student Country - 8 unique values

21.

22. Creating ENHANCED features...

23.

24. [3] EVALUATING MODELS BEFORE BALANCING...

25. Random Forest Accuracy (Before Balancing): 0.7099

26. precision recall f1-score support

27.

28. 0 0.74 0.70 0.72 1016

29. 1 0.68 0.72 0.70 894

30.

31. accuracy 0.71 1910

32. macro avg 0.71 0.71 0.71 1910

33. weighted avg 0.71 0.71 0.71 1910

34.

35. Gradient Boosting Accuracy (Before Balancing): 0.7157

36. precision recall f1-score support

37.

38. 0 0.74 0.72 0.73 1016

39. 1 0.69 0.71 0.70 894

40.

41. accuracy 0.72 1910

42. macro avg 0.71 0.72 0.72 1910

43. weighted avg 0.72 0.72 0.72 1910

44.

45. Neural Network Accuracy (Before Balancing): 0.7450

46. precision recall f1-score support

47.

48. 0 0.77 0.74 0.75 1016

49. 1 0.72 0.75 0.73 894

50.

51. accuracy 0.75 1910

52. macro avg 0.74 0.75 0.74 1910

53. weighted avg 0.75 0.75 0.75 1910

54.

55. Stacking Ensemble Accuracy (Before Balancing): 0.7466

56. precision recall f1-score support

57.

58. 0 0.77 0.75 0.76 1016

59. 1 0.72 0.74 0.73 894

60.

61. accuracy 0.75 1910

62. macro avg 0.75 0.75 0.75 1910

63. weighted avg 0.75 0.75 0.75 1910

64.

65.

66. [4] APPLYING ENHANCED UNIFORM BALANCING (ALL TO 3001)...

67.

68. Processing country: Ireland

69. Class 0 Before: 162, Class 1 Before: 138

70. Target count for all classes: 3001

71. After resampling - Class 0: 3001, Class 1: 3001, Synthetic: 5702

72.

73. Processing country: Portugal

74. Class 0 Before: 3001, Class 1 Before: 2494

75. Target count for all classes: 3001

76. After resampling - Class 0: 3001, Class 1: 3001, Synthetic: 507

77.

78. Processing country: Italy

79. Class 0 Before: 752, Class 1 Before: 606

80. Target count for all classes: 3001

81. After resampling - Class 0: 3001, Class 1: 3001, Synthetic: 4644

82.

83. Processing country: Lithuania

84. Class 0 Before: 814, Class 1 Before: 629

85. Target count for all classes: 3001

86. After resampling - Class 0: 3001, Class 1: 3001, Synthetic: 4559

87.

88. Processing country: Spain

89. Class 0 Before: 16, Class 1 Before: 12

90. Target count for all classes: 3001

91. After resampling - Class 0: 3001, Class 1: 3001, Synthetic: 5974

92.

93. Processing country: Russian Federation

94. Class 0 Before: 70, Class 1 Before: 37

95. Target count for all classes: 3001

96. After resampling - Class 0: 3001, Class 1: 3001, Synthetic: 5895

97.

98. Processing country: Romania

99. Class 0 Before: 25, Class 1 Before: 35

100. Target count for all classes: 3001

101. After resampling - Class 0: 3001, Class 1: 3001, Synthetic: 5942

102.

103. Processing country: Slovenia

104. Class 0 Before: 236, Class 1 Before: 519

105. Target count for all classes: 3001

106. After resampling - Class 0: 3001, Class 1: 3001, Synthetic: 5247

107.

108. Combined resampled dataset shape: (48016, 20)

109. Final class distribution after resampling:

110. Type of Answer

111. 0 24008

112. 1 24008

113. Name: count, dtype: int64

114.

115. Resampling Statistics by Country:

116. Country Class\_0\_Before Class\_1\_Before Class\_0\_After Class\_1\_After Synthetic\_Samples

117. 0 Ireland 162 138 3001 3001 5702

118. 1 Portugal 3001 2494 3001 3001 507

119. 2 Italy 752 606 3001 3001 4644

120. 3 Lithuania 814 629 3001 3001 4559

121. 4 Spain 16 12 3001 3001 5974

122. 5 Russian Federation 70 37 3001 3001 5895

123. 6 Romania 25 35 3001 3001 5942

124. 7 Slovenia 236 519 3001 3001 5247

125.

126. [5] TRAINING OPTIMIZED MODELS ON BALANCED DATA...

127. Optimized Random Forest Accuracy (After Balancing): 0.8687

128. precision recall f1-score support

129.

130. 0 0.88 0.85 0.87 4802

131. 1 0.86 0.89 0.87 4802

132.

133. accuracy 0.87 9604

134. macro avg 0.87 0.87 0.87 9604

135. weighted avg 0.87 0.87 0.87 9604

136.

137. Optimized Gradient Boosting Accuracy (After Balancing): 0.8662

138. precision recall f1-score support

139.

140. 0 0.88 0.85 0.86 4802

141. 1 0.85 0.88 0.87 4802

142.

143. accuracy 0.87 9604

144. macro avg 0.87 0.87 0.87 9604

145. weighted avg 0.87 0.87 0.87 9604

146.

147. Optimized Neural Network Accuracy (After Balancing): 0.8188

148. precision recall f1-score support

149.

150. 0 0.84 0.79 0.81 4802

151. 1 0.80 0.85 0.82 4802

152.

153. accuracy 0.82 9604

154. macro avg 0.82 0.82 0.82 9604

155. weighted avg 0.82 0.82 0.82 9604

156.

157. Enhanced Stacking Ensemble Accuracy (After Balancing): 0.8692

158. precision recall f1-score support

159.

160. 0 0.88 0.85 0.87 4802

161. 1 0.86 0.88 0.87 4802

162.

163. accuracy 0.87 9604

164. macro avg 0.87 0.87 0.87 9604

165. weighted avg 0.87 0.87 0.87 9604

166.

167.

168. Top 10 Most Important Features:

169. Feature Importance

170. 6 Question\_Difficulty 0.145970

171. 10 Student\_Topic\_Performance 0.070958

172. 14 Difficulty\_Level\_Interaction 0.069628

173. 1 Question ID 0.056159

174. 11 Student\_Level\_Performance 0.046332

175. 5 Student\_Avg\_Performance 0.029886

176. 15 Student\_Country\_Interaction 0.028864

177. 12 Country\_Topic\_Performance 0.016218

178. 16 Topic\_Subtopic\_Interaction 0.013392

179. 0 Student Country 0.012912

180.

181. Accuracy Comparison:

182. Model Accuracy Before Balancing Accuracy After Balancing Improvement

183. 0 Random Forest 0.709948 0.868701 0.158753

184. 1 Gradient Boosting 0.715707 0.866202 0.150495

185. 2 Neural Network 0.745026 0.818825 0.073799

186. 3 Stacking Ensemble 0.746597 0.869221 0.122624

187.

188. Enhanced analysis complete! Results saved to: analysis\_results\_20250417\_222650

189. PS D:\>

190.