Unreliable Biomarker Panel Using Random Numbers

Zainab Nazari

Unreliable result is obtained when biomarker panel is generated first, using all the available data, and then tested by cross-validation.

This approach is implemented incorrectly, the result is not "credible".

When results are derived from an approach that does not properly account for overfitting and validation, performance metrics such as accuracy, sensitivity, specificity, and area under the curve (AUC) can be misleading. These metrics may suggest that the biomarker panel is highly effective when, in reality, it may not perform well in practice.

This following code generates a synthetic dataset of GeneID features for 40 participants, with 2000 random protein features and binary labels indicating disease status. It uses XGBoost to identify the top 15 most important features, based on feature importance scores. Then, it applies stratified 5-fold cross-validation to evaluate the model's performance on these top 15 features. Predicted probabilities from the cross-validation are used to compute the ROC curve and the Area Under the Curve (AUC).

The impressive AUC score of 0.94 may initially appear promising; however, it is crucial to recognize that this result was derived from a synthetic random dataset. This highlights the importance of using appropriate methodologies in our analyses. Achieving meaningful results requires careful consideration of the data and the techniques employed, rather than relying on potentially misleading metrics that do not reflect real-world performance.

In [1]:
```
import pandas as pd
import numpy as np
from xgboost import XGBClassifier
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.metrics import roc_auc_score, make_scorer
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import cross_val_predict
import seaborn as sns

# Step 1: Generate Random Data
np.random.seed(42)

# Generate 100 participants (rows) with 2000 random geneID features
GeneID_data = np.random.rand(40, 2000)
disease_status = np.random.choice([0, 1], size=40)  # Binary labels indicati

# Create a DataFrame with random GeneID data
columns = [f'GeneID_{i+1}' for i in range(2000)]
df = pd.DataFrame(GeneID_data, columns=columns)
df['Disease'] = disease_status
df
```

Out[1]:

	GeneID_1	GeneID_2	GeneID_3	GeneID_4	GeneID_5	GeneID_6	GeneID_7	GeneID_8	Ge
0	0.374540	0.950714	0.731994	0.598658	0.156019	0.155995	0.058084	0.866176	0.
1	0.261706	0.246979	0.906255	0.249546	0.271950	0.759398	0.449740	0.776711	0.0
2	0.571996	0.805432	0.760161	0.153900	0.149249	0.268174	0.361075	0.408456	0.6
3	0.648257	0.172386	0.872395	0.613116	0.157204	0.962338	0.518365	0.072898	0.6
4	0.720268	0.687283	0.095754	0.922572	0.568472	0.363726	0.756539	0.257365	0.6
5	0.373641	0.332912	0.176154	0.607267	0.476624	0.865701	0.032110	0.643868	0.7
6	0.654306	0.080033	0.242330	0.773679	0.528686	0.927969	0.428751	0.869812	0.4
7	0.073175	0.089478	0.651974	0.486941	0.790415	0.916271	0.388367	0.596410	0.5
8	0.004402	0.000330	0.472263	0.029294	0.974533	0.222065	0.472665	0.061138	0.5
9	0.645691	0.402418	0.471909	0.716725	0.876306	0.272219	0.342077	0.930739	0.5
10	0.729998	0.184512	0.346640	0.663281	0.482089	0.738571	0.961208	0.116547	0.7
11	0.764636	0.006516	0.477050	0.793904	0.630045	0.474063	0.644642	0.906946	0.5
12	0.685528	0.467571	0.055707	0.918510	0.980329	0.431043	0.613661	0.450299	0.3
13	0.060570	0.203208	0.491263	0.368150	0.410912	0.781447	0.014679	0.660492	0.
14	0.745880	0.262071	0.654146	0.699199	0.345118	0.474919	0.403767	0.947349	0.3
15	0.638145	0.459292	0.964499	0.218978	0.587856	0.700210	0.825564	0.406971	0.6
16	0.026658	0.007961	0.224375	0.799321	0.904748	0.391768	0.563199	0.706053	0.9
17	0.533400	0.536756	0.759721	0.817031	0.337195	0.801562	0.814209	0.234126	0.
18	0.916592	0.950214	0.004410	0.317075	0.837378	0.536445	0.807509	0.853323	0.4
19	0.269987	0.663679	0.371357	0.218953	0.408629	0.763270	0.475793	0.299864	0.8
20	0.298912	0.094818	0.126359	0.180671	0.203653	0.242262	0.255460	0.455716	0.5
21	0.946699	0.896413	0.235497	0.262760	0.981438	0.236561	0.953479	0.649754	0.0
22	0.583290	0.887597	0.762901	0.867406	0.059423	0.318126	0.318712	0.657045	0.4
23	0.407564	0.431417	0.197118	0.009551	0.349359	0.148414	0.421727	0.833825	0.
24	0.686706	0.412564	0.280342	0.615857	0.740269	0.254585	0.299400	0.640294	0.9
25	0.847237	0.494517	0.195466	0.736642	0.418678	0.594627	0.107265	0.631584	0.3
26	0.056704	0.947704	0.659054	0.249654	0.533536	0.635371	0.226659	0.093540	0.2
27	0.007563	0.663538	0.913460	0.354768	0.314057	0.626030	0.328552	0.402418	0.
28	0.470901	0.984670	0.727657	0.845849	0.439675	0.904777	0.732151	0.746128	0.0
29	0.635574	0.939913	0.991109	0.570274	0.273630	0.914983	0.844195	0.084611	0.9
30	0.741555	0.881102	0.463180	0.289179	0.318847	0.696948	0.567558	0.486494	0.2
31	0.929964	0.937747	0.558168	0.663302	0.349172	0.607159	0.302748	0.746028	0.3
32	0.157366	0.020781	0.327086	0.824316	0.015380	0.109434	0.806855	0.305852	0.9
33	0.429136	0.049549	0.201128	0.019235	0.254792	0.739565	0.193319	0.528056	0.7
34	0.645401	0.433148	0.134349	0.005923	0.029677	0.867516	0.457827	0.657427	0.9
35	0.042661	0.828505	0.249308	0.283937	0.226245	0.840084	0.978902	0.755187	0.8
36	0.477577	0.977820	0.816840	0.797894	0.583713	0.461150	0.894611	0.804557	0.5

	GeneID_1	GeneID_2	GeneID_3	GeneID_4	GeneID_5	GeneID_6	GeneID_7	GeneID_8	Ge
37	0.698683	0.725310	0.292684	0.806458	0.946241	0.369851	0.835349	0.772040	0.5
38	0.841912	0.009050	0.799747	0.688040	0.379678	0.791586	0.126662	0.203319	0.5
39	0.120140	0.568022	0.055043	0.588410	0.342462	0.669410	0.782042	0.791399	0.0

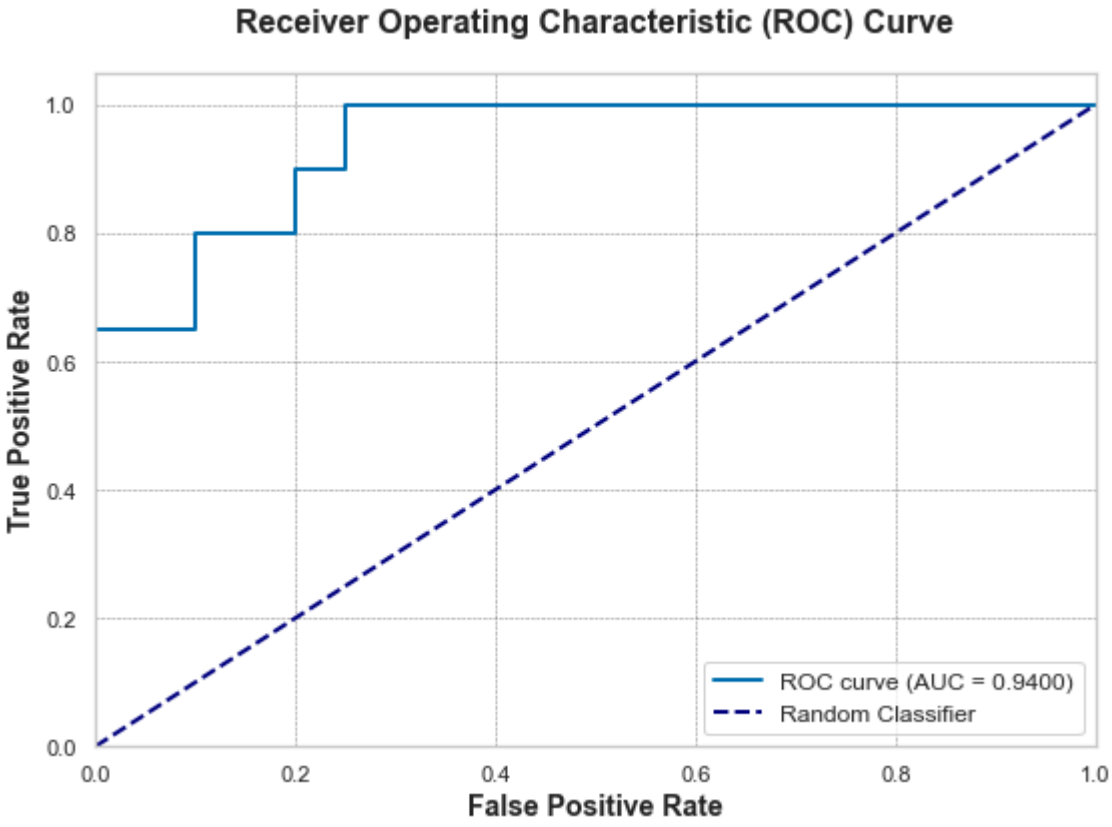
40 rows × 2001 columns

In [2]:
```
# Step 2: Define Features and Labels
X = df.drop(columns=['Disease'])
y = df['Disease']

# Step 3: Initialize XGBoost Classifier and fit with all features
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='auc', randor
xgb_model.fit(X, y)

# Step 4: Identify the top 15 best features by feature importance
feature_importances = pd.Series(xgb_model.feature_importances_, index=X.colu
top_15_features = feature_importances.nlargest(15).index  # Select top 15 fe

# Select only the top 15 features for the dataset
X_top15 = X[top_15_features]

# Step 5: Cross-validation with the top 15 features
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
auc_scores = cross_val_score(XGBClassifier(use_label_encoder=False, eval_met
                             X_top15, y, cv=cv, scoring=make_scorer(roc_auc_

# Output the average AUC score
average_auc = np.mean(auc_scores)
print("Selected top 15 features:", list(top_15_features))
print(f"Cross-validated AUC with top 15 features: {average_auc:.4f}")
```

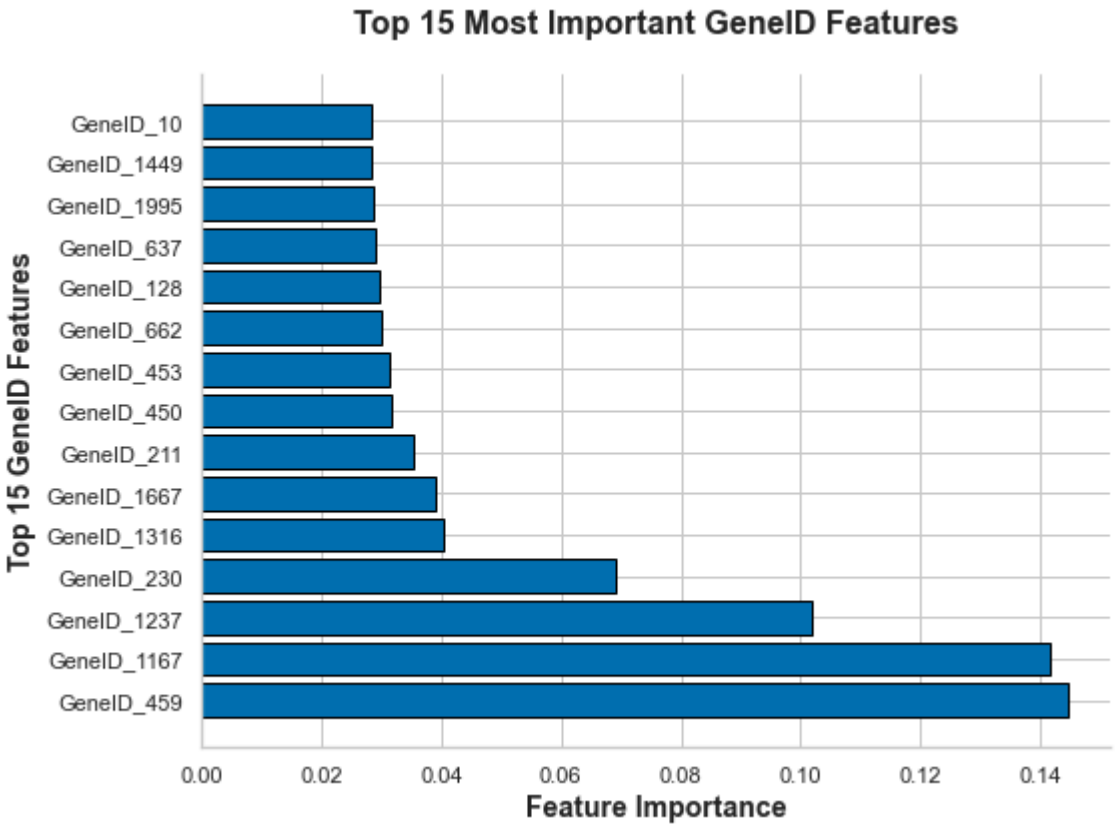
```
Selected top 15 features: ['GeneID_459', 'GeneID_1167', 'GeneID_1237', 'Gen
eID_230', 'GeneID_1316', 'GeneID_1667', 'GeneID_211', 'GeneID_450', 'GeneID
_453', 'GeneID_662', 'GeneID_128', 'GeneID_637', 'GeneID_1995', 'GeneID_144
9', 'GeneID_10']
Cross-validated AUC with top 15 features: 0.8500
```

In [3]:
```
# Initialize XGBoost model with top 15 selected features
xgb_model_top15 = XGBClassifier(use_label_encoder=False, eval_metric='auc',

# Get cross-validated probabilities for ROC curve calculation
y_pred_proba = cross_val_predict(xgb_model_top15, X_top15, y, cv=cv, method=

# Calculate ROC curve and AUC
fpr, tpr, _ = roc_curve(y, y_pred_proba)
roc_auc = auc(fpr, tpr)

# Set Seaborn style for professional plots
sns.set(style="whitegrid")

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='#0072B2', lw=2, label=f'ROC curve (AUC = {roc_auc:
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label="Random (

# Set axis limits and labels
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize=14, fontweight='bold', fontname=
plt.ylabel('True Positive Rate', fontsize=14, fontweight='bold', fontname='A
```

```
plt.title('Receiver Operating Characteristic (ROC) Curve', fontsize=16, font

# Add grid, legend, and layout adjustments
plt.grid(color='grey', linestyle='--', linewidth=0.5)
plt.legend(loc="lower right", fontsize=12)
plt.tight_layout()  # Ensure the layout fits well

# Display the plot
plt.show()
```



In [4]:
```
# Sort the top 15 features by their importance values in ascending order for
top_15_importances = feature_importances[top_15_features].sort_values(ascend

# Set Seaborn style for professional plots
sns.set(style="whitegrid")

# Plot
plt.figure(figsize=(8, 6))
plt.barh(top_15_importances.index, top_15_importances.values, color="#0072B2

# Customize plot appearance for scientific presentation
plt.xlabel('Feature Importance', fontsize=14, fontweight='bold', fontname='A
plt.ylabel('Top 15 GeneID Features', fontsize=14, fontweight='bold', fontnam
plt.title('Top 15 Most Important GeneID Features', fontsize=16, fontweight=

# Remove top and right spines for a cleaner look, and invert y-axis to show
sns.despine()
plt.gca().invert_yaxis()

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```


Top 15 Most Important GeneID Features
Top 15 GeneID Features
GeneID_10
GeneID_1449
GeneID_1995
GeneID_637
GeneID_128
GeneID_662
GeneID_453
GeneID_450
GeneID_211
GeneID_1667
GeneID_1316
GeneID_230
GeneID_1237
GeneID_1167
GeneID_459
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
Feature Importance