Model Development Phase Template

Date	18 June 2024
Team ID	SWTID1749713922
Project Title	Early Prediction for Chronic Kidney Disease Detection: A Progressive Approach to Health Management
Maximum Marks	4 Marks

Initial Model Training Code, Model Validation and Evaluation Report

The model validation and evaluation report will include classification reports, accuracy, and confusion matrices for multiple models, presented through respective screenshots.

Initial Model Training Code:

```
def run_healthcare_ml_pipeline_hyp(X_train, X_test, y_train, y_test):
    """
    Complete healthcare ML pipeline with enhanced hyperparameter tuning and overfitting detection
    """
    print(f"\n{'='*80}")
    print("HEALTHCARE ML ANALYSIS PIPELINE")
    print("Kidney Disease Detection - Enhanced with Hyperparameter Tuning")
    print(f"{'='*80}")

    # Convert target to binary if needed
    if hasattr(y_train, 'dtype') and y_train.dtype == 'object':
        unique_classes = np.unique(y_train)
        if len(unique_classes) == 2:
            y_train_processed = (y_train == unique_classes[0]).astype(int)
            y_test_processed = (y_test == unique_classes[0]).astype(int)
        else:
            from sklearn.preprocessing import LabelEncoder
            le = LabelEncoder()
            y_train_processed = le.fit_transform(y_train)
            y_test_processed = le.transform(y_test)
    else:
        y_train_processed = y_train
        y_test_processed = y_test
```

```
print(f"Dataset Info:")
print(f"   Training samples: {X_train.shape[0]}")
print(f"   Test samples: {X_test.shape[0]}")
print(f"   Features: {X_train.shape[1]}")
print(f"   Classes: {len(np.unique(y_train_processed))}")
print(f"   Class distribution: {dict(zip(*np.unique(y_train_processed, return_counts=True)))}")

# Create hyperparameter-tuned models
models, tuning_summary = create_tuned_models_dict(X_train, y_train_processed)
print(f"\n Testing {len(models)} different classification algorithms...")
print(f"    {len(tuning_summary)} models have been hyperparameter-tuned")
print(f"    {len(models) - len(tuning_summary)} baseline models included")

# Print hyperparameter tuning summary
if tuning_summary:
    print_hyperparameter_tuning_summary(tuning_summary)

# Run comprehensive analysis with enhanced overfitting detection
results_list, cv_results, overfitting_summary = detect_overfitting_comprehensive_enhanced(
    X_train, X_test, y_train_processed, y_test_processed, models
)

# Convert results list to dictionary for easier access
results_dict = {}
for result in results_list:
    model_name = result['Model']
    results_dict[model_name] = result
```

```
print(f"\n{'='*80}")
print("INDIVIDUAL MODEL ANALYSIS WITH VISUALIZATIONS")
print(f"{'='*80}")

# Dictionary to store trained models for plotting
trained_models = {}

# Individual model analysis with plotting
for model_name, model_results in results_dict.items():
    if model_results is None:
        continue
    print(f"\n ANALYZING: {model_name}")
    print("-" * 60)

    try:
        # Get model instance
        if model_name not in models:
            print(f" Model {model_name} not found in models dictionary, skipping...")
            continue

        model = models[model_name]

        # Train the model
        model.fit(X_train, y_train_processed)
        trained_models[model_name] = model

        # Make predictions
        y_pred = model.predict(X_test)
```

```python
# Get prediction probabilities if available
if hasattr(model, 'predict_proba'):
    y_pred_proba = model.predict_proba(X_test)
    if len(np.unique(y_test_processed)) == 2:
        y_pred_proba = y_pred_proba[:, 1]
    else:
        y_pred_proba = y_pred_proba.max(axis=1)
elif hasattr(model, 'decision_function'):
    y_pred_proba = model.decision_function(X_test)
    # Normalize decision function scores to [0,1] for binary classification
    if len(np.unique(y_test_processed)) == 2:
        y_pred_proba = (y_pred_proba - y_pred_proba.min()) / (y_pred_proba.max() - y_pred_proba.min())
else:
    y_pred_proba = None

print(f" Model Performance:")
# Access metrics from model_results dictionary
print(f"   Test Accuracy: {model_results.get('Test Accuracy', 0):.4f}")
print(f"   Precision: {model_results.get('Precision', 0):.4f}")
print(f"   Recall: {model_results.get('Recall', 0):.4f}")
print(f"   F1 Score: {model_results.get('F1 Score', 0):.4f}")
print(f"   ROC AUC: {model_results.get('ROC AUC', 0):.4f}")

# Display hyperparameter tuning results if available
if model_name in tuning_summary:
    tuning_info = tuning_summary[model_name]
    print(f"\n Hyperparameter Tuning Results:")
    print(f"   Best Recall Score (CV): {tuning_info['results']['best_score']:.4f}")
    print(f"   Overfitting Risk: {tuning_info['results']['overfitting_risk']}")
    print(f"   Overfitting Gap: {tuning_info['results']['overfitting_gap']:.4f}")
```

```python
    print(f"   Search Method: {tuning_info['results']['search_type']}")

    # Show key hyperparameters
    key_params = list(tuning_info['params'].items())[:3]  # Show first 3 params
    if key_params:
        print(f"   Key Tuned Parameters:")
        for param, value in key_params:
            print(f"      • {param}: {value}")

# Plot confusion matrix
print(f"\n Generating Confusion Matrix...")
plot_confusion_matrix(y_test_processed, y_pred, model_name)

# Plot ROC curve (only for binary classification)
if len(np.unique(y_test_processed)) == 2 and y_pred_proba is not None:
    print(f" Generating ROC Curve...")
    plot_roc_curve(y_test_processed, y_pred_proba, model_name)

    print(f" Generating Precision-Recall Curve...")
    plot_precision_recall_curve(y_test_processed, y_pred_proba, model_name)

# Plot feature importance (if available)
if hasattr(model, 'feature_importances_'):
    print(f" Generating Feature Importance Plot...")
    plot_feature_importance(model, X_train, model_name)
elif hasattr(model, 'coef_') and model.coef_.ndim == 1:
    print(f" Generating Feature Coefficients Plot...")
    # Handle linear model coefficients
```

```
            plt.figure(figsize=(10, 6))
            if hasattr(X_train, 'columns'):
                coef_series = pd.Series(np.abs(model.coef_), index=X_train.columns)
            else:
                coef_series = pd.Series(np.abs(model.coef_), index=[f'Feature_{i}' for i in range(len(model.coef_))])
            coef_series.sort_values(ascending=False).head(10).plot(kind='bar')
            plt.title(f'Top 10 Feature Coefficients (Absolute) - {model_name}')
            plt.xticks(rotation=45, ha='right')
            plt.tight_layout()
            plt.show()

        # Validation curve analysis for selected models with hyperparameters
        print(f" Generating Validation Curve Analysis...")
        if model_name == 'Random Forest':
            validation_curve_analysis_enhanced(
                X_train, y_train_processed, model,
                'n_estimators', [10, 50, 100, 200, 300]
            )
        elif model_name == 'XGBoost':
            validation_curve_analysis_enhanced(
                X_train, y_train_processed, model,
                'max_depth', [3, 4, 5, 6, 7, 8]
            )
        elif model_name == 'LightGBM':
            validation_curve_analysis_enhanced(
                X_train, y_train_processed, model,
                'num_leaves', [10, 20, 30, 40, 50]
            )
        elif 'SVM' in model_name:
            validation_curve_analysis_enhanced(
                X_train, y_train_processed, model,
```

```
                'C', [0.1, 1, 10, 100, 1000]
            )
        elif 'Logistic Regression' in model_name:
            validation_curve_analysis_enhanced(
                X_train, y_train_processed, model,
                'C', [0.01, 0.1, 1, 10, 100]
            )

        print(f"Completed analysis for {model_name}\n")

    except Exception as e:
        print(f" Error analyzing {model_name}: {str(e)}")
        continue

    # Print overfitting summary
risk_groups = print_overfitting_summary(overfitting_summary)

# Convert results list to DataFrame for healthcare model selection
results_df = pd.DataFrame(results_list)

# Healthcare-specific model selection
best_model_name, ranked_models = healthcare_model_selection_algorithm(results_df)

# Generate comprehensive model comparison plots
print(f"\n{'='*80}")
print(" COMPREHENSIVE MODEL COMPARISON VISUALIZATIONS")
print(f"{'='*80}")
```

```python
    print(" Generating Top Models Comparison...")
    plot_model_comparison(results_df)

    # Enhanced hyperparameter tuning summary visualization
    if tuning_summary:
        print(f"\n Generating Hyperparameter Tuning Summary Visualization...")
        plot_hyperparameter_tuning_summary(tuning_summary)

    # Final recommendations with hyperparameter considerations
    print(f"\n{'='*80}")
    print(" FINAL HEALTHCARE RECOMMENDATIONS")
    print(f"{'='*80}")

    print(f" RECOMMENDED MODEL: {best_model_name}")
    best_stats = ranked_models.iloc[0]
    print(f"   Healthcare Score: {best_stats['Healthcare_Score']:.4f}")
    print(f"   Recall (Sensitivity): {best_stats['Recall']:.4f}")
    print(f"   Precision: {best_stats['Precision']:.4f}")
    print(f"   F1 Score: {best_stats['F1 Score']:.4f}")
    print(f"   Overfitting Risk: {best_stats['Overfitting Risk']}")

    # Show hyperparameter tuning info for best model if available
    if best_model_name in tuning_summary:
        best_tuning = tuning_summary[best_model_name]
        print(f"    Hyperparameter Optimization:")
        print(f"      • Tuning Method: {best_tuning['results']['search_type']}")
        print(f"      • CV Recall Score: {best_tuning['results']['best_score']:.4f}")
        print(f"      • Overfitting Gap: {best_tuning['results']['overfitting_gap']:.4f}")
```

```python
    print(f"\n TOP 3 SAFE MODELS FOR HEALTHCARE:")
    safe_models = ranked_models[ranked_models['Overfitting Risk'].isin(['LOW', 'MEDIUM'])].head(3)
    for i, (_, row) in enumerate(safe_models.iterrows(), 1):
        risk_indicator = "" if row['Overfitting Risk'] == 'LOW' else ""
        tuning_indicator = "" if row['Model'] in tuning_summary else ""
        print(f"   {i}. {tuning_indicator} {row['Model']} (Score: {row['Healthcare_Score']:.4f}, Risk: {risk_indicator}{row['Overfitting Risk']})")

    # Models to avoid with hyperparameter tuning context
    avoid_models = ranked_models[ranked_models['Overfitting Risk'].isin(['CRITICAL', 'HIGH'])]['Model'].tolist()
    if avoid_models:
        print(f"\n MODELS TO AVOID IN HEALTHCARE:")
        for model in avoid_models[:5]:  # Show top 5 to avoid
            if model in tuning_summary:
                gap = tuning_summary[model]['results']['overfitting_gap']
                print(f"     {model} (Overfitting Gap: {gap:.4f})")
            else:
                print(f"     {model}")

    # Hyperparameter tuning insights
    if tuning_summary:
        print(f"\n HYPERPARAMETER TUNING INSIGHTS:")

        # Count tuned models by risk level
        tuned_risks = {}
        for model_name, info in tuning_summary.items():
            risk = info['results']['overfitting_risk']
            tuned_risks[risk] = tuned_risks.get(risk, 0) + 1
```

```python
    print(f"    Tuned Models by Risk Level:")
    risk_order = ['LOW', 'MEDIUM', 'HIGH', 'CRITICAL']
    for risk in risk_order:
        if risk in tuned_risks:
            icon = {'LOW': '🟢', 'MEDIUM': '🟡', 'HIGH': '🟠', 'CRITICAL': '🔴'}.get(risk, '⚪')
            print(f"      {icon} {risk}: {tuned_risks[risk]} models")

    # Best tuning results
    best_tuned_recall = max(tuning_summary.items(), key=lambda x: x[1]['results']['best_score'])
    lowest_overfitting_tuned = min(tuning_summary.items(), key=lambda x: x[1]['results']['overfitting_gap'])

    print(f"    Best Tuned Recall: {best_tuned_recall[0]} ({best_tuned_recall[1]['results']['best_score']:.4f})")
    print(f"    Lowest Overfitting (Tuned): {lowest_overfitting_tuned[0]} (Gap: {lowest_overfitting_tuned[1]['results']['overfitting_gap']:.4f})")

# Additional comprehensive analysis plots
print(f"\n GENERATING ADDITIONAL ANALYSIS PLOTS...")

# Overfitting risk distribution plot
plt.figure(figsize=(15, 10))

# Risk distribution
plt.subplot(2, 3, 1)
risk_counts = ranked_models['Overfitting Risk'].value_counts()
colors = {'LOW': 'green', 'MEDIUM': 'orange', 'HIGH': 'red', 'CRITICAL': 'darkred'}
risk_colors = [colors.get(risk, 'gray') for risk in risk_counts.index]
plt.pie(risk_counts.values, labels=risk_counts.index, autopct='%1.1f%%', colors=risk_colors)
plt.title('Overfitting Risk Distribution')
```

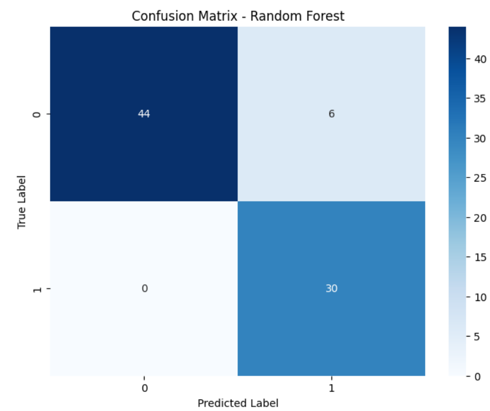
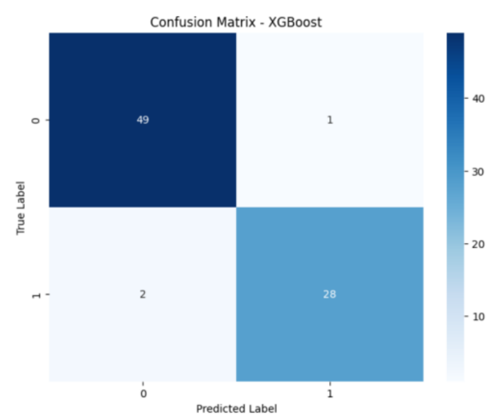
```python
# Healthcare scores distribution
plt.subplot(2, 3, 2)
plt.hist(ranked_models['Healthcare_Score'], bins=15, alpha=0.7, color='skyblue', edgecolor='black')
plt.xlabel('Healthcare Score')
plt.ylabel('Number of Models')
plt.title('Healthcare Scores Distribution')
plt.grid(True, alpha=0.3)

# Recall vs Precision scatter plot
plt.subplot(2, 3, 3)
colors_risk = ranked_models['Overfitting Risk'].map(colors)
scatter = plt.scatter(ranked_models['Recall'], ranked_models['Precision'],
                     c=colors_risk, alpha=0.7, s=60, edgecolors='black', linewidth=0.5)
plt.xlabel('Recall (Sensitivity)')
plt.ylabel('Precision')
plt.title('Recall vs Precision (Colored by Risk)')
plt.grid(True, alpha=0.3)

# F1 Score vs CV Stability
plt.subplot(2, 3, 4)
plt.scatter(ranked_models['F1 Score'], ranked_models['CV Std F1'],
           c=colors_risk, alpha=0.7, s=60, edgecolors='black', linewidth=0.5)
plt.xlabel('F1 Score')
plt.ylabel('CV Standard Deviation')
plt.title('Performance vs Stability (Colored by Risk)')
plt.grid(True, alpha=0.3)
```

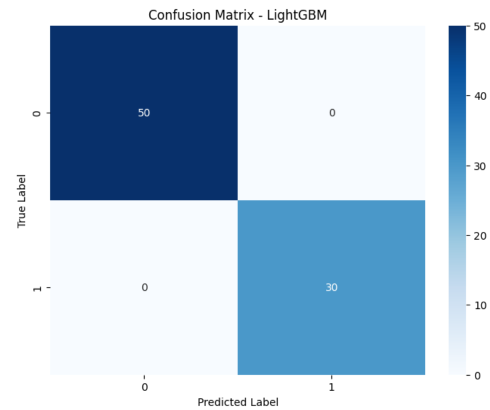
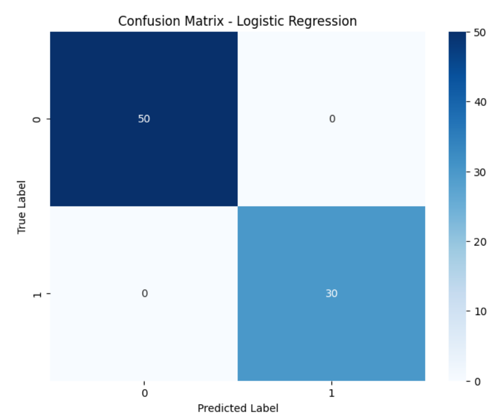
```
# Hyperparameter tuning comparison (if available)
if tuning_summary:
    plt.subplot(2, 3, 5)
    tuned_models_data = []
    tuned_scores = []
    tuned_gaps = []
    for model_name in ranked_models['Model']:
        if model_name in tuning_summary:
            tuned_models_data.append(model_name[:15])  # Truncate long names
            tuned_scores.append(tuning_summary[model_name]['results']['best_score'])
            tuned_gaps.append(tuning_summary[model_name]['results']['overfitting_gap'])

    if tuned_models_data:
        plt.scatter(tuned_scores, tuned_gaps, alpha=0.7, s=60,
                    c='purple', edgecolors='black', linewidth=0.5)
        plt.xlabel('Tuned CV Recall Score')
        plt.ylabel('Overfitting Gap')
        plt.title('Hyperparameter Tuning Results')
        plt.grid(True, alpha=0.3)

        # Add model names as annotations for top performers
        for i, (score, gap, name) in enumerate(zip(tuned_scores, tuned_gaps, tuned_models_data)):
            if score > np.percentile(tuned_scores, 75) and gap < np.percentile(tuned_gaps, 50):
                plt.annotate(name, (score, gap), xytext=(5, 5),
                           textcoords='offset points', fontsize=8)
```

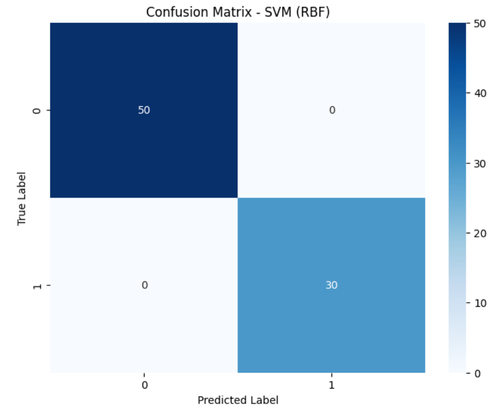
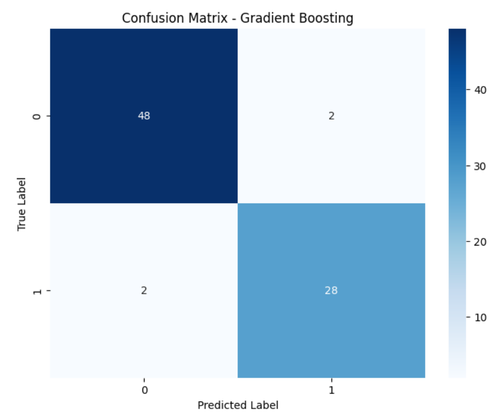
```
# Model complexity vs performance
plt.subplot(2, 3, 6)
# Create a complexity score based on model type
complexity_map = {
    'Dummy': 1, 'Naive Bayes': 2, 'Logistic Regression': 3, 'LDA': 3, 'QDA': 4,
    'Decision Tree': 4, 'KNN': 4, 'SVM': 5, 'Random Forest': 6, 'Extra Trees': 6,
    'AdaBoost': 6, 'Gradient Boosting': 7, 'XGBoost': 8, 'LightGBM': 8, 'CatBoost': 8,
    'MLP': 9, 'SGD': 3, 'Ridge': 3, 'Bagging': 5
}

complexity_scores = []
for model_name in ranked_models['Model']:
    complexity = 5  # default
    for key, value in complexity_map.items():
        if key.lower() in model_name.lower():
            complexity = value
            break
    complexity_scores.append(complexity)

plt.scatter(complexity_scores, ranked_models['Healthcare_Score'],
           c=colors_risk, alpha=0.7, s=60, edgecolors='black', linewidth=0.5)
plt.xlabel('Model Complexity')
plt.ylabel('Healthcare Score')
plt.title('Complexity vs Healthcare Performance')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

```python
    # Summary statistics
    print(f"\n SUMMARY STATISTICS:")
    print(f"    Total Models Evaluated: {len(results_list)}")
    print(f"    Models with Hyperparameter Tuning: {len(tuning_summary)}")
    print(f"    Low Risk Models: {len(ranked_models[ranked_models['Overfitting Risk'] == 'LOW'])}")
    print(f"    Medium Risk Models: {len(ranked_models[ranked_models['Overfitting Risk'] == 'MEDIUM'])}")
    print(f"    High Risk Models: {len(ranked_models[ranked_models['Overfitting Risk'] == 'HIGH'])}")
    print(f"    Critical Risk Models: {len(ranked_models[ranked_models['Overfitting Risk'] == 'CRITICAL'])}")
    print(f"    Average Healthcare Score: {ranked_models['Healthcare_Score'].mean():.4f}")
    print(f"    Average Recall: {ranked_models['Recall'].mean():.4f}")
    print(f"    Average Precision: {ranked_models['Precision'].mean():.4f}")

    return {
        'results': results_dict,
        'results_list': results_list,
        'cv_results': cv_results,
        'overfitting_summary': overfitting_summary,
        'risk_groups': risk_groups,
        'best_model': best_model_name,
        'ranked_models': ranked_models,
        'safe_models': safe_models,
        'trained_models': trained_models,
        'tuning_summary': tuning_summary,
        'hyperparameter_insights': {
            'tuned_models_count': len(tuning_summary),
            'best_tuned_recall': best_tuned_recall if tuning_summary else None,
            'lowest_overfitting_tuned': lowest_overfitting_tuned if tuning_summary else None,
            'risk_distribution': tuned_risks if tuning_summary else None
        }
    }
```

Model Validation and Evaluation Report:

Model	Classification Report	Accuracy	Confusion Matrix
Random Forest	COMPREHENSIVE ANALYSIS: Random Forest == PERFORMANCE METRICS: Test Accuracy: 0.9250 Precision: 0.9375 Recall: 0.9250 F1 Score: 0.9260 ROC AUC: 0.9933 PR AUC: 0.9889 ENHANCED OVERFITTING ANALYSIS: Train F1 Score: 0.9561 Test F1 Score: 0.9260 F1 Performance Gap: 0.0301 Accuracy Gap: 0.0307 Overfitting Score: 0.150 Overfitting Risk: LOW STRATIFIED K-FOLD CROSS VALIDATION: CV Scores: ['0.9844', '0.9203', '0.9372', '0.9526', '0.9683'] CV Mean: 0.9526 CV Std: 0.0225 CV Coefficient of Variation: 0.0237 DETAILED CLASSIFICATION REPORT: precision recall f1-score support 0 1.00 0.88 0.94 50 1 0.83 1.00 0.91 30 accuracy 0.93 80 macro avg 0.92 0.94 0.92 80 weighted avg 0.94 0.93 0.93 80	92.50%	
XGBoost	COMPREHENSIVE ANALYSIS: XGBoost == PERFORMANCE METRICS: Test Accuracy: 0.9625 Precision: 0.9626 Recall: 0.9625 F1 Score: 0.9624 ROC AUC: 0.9933 PR AUC: 0.9905 ENHANCED OVERFITTING ANALYSIS: Train F1 Score: 0.9905 Test F1 Score: 0.9624 F1 Performance Gap: 0.0281 Accuracy Gap: 0.0280 Overfitting Score: 0.141 Overfitting Risk: LOW STRATIFIED K-FOLD CROSS VALIDATION: CV Scores: ['1.0000', '0.9685', '0.9217', '0.9842', '0.9683'] CV Mean: 0.9685 CV Std: 0.0262 CV Coefficient of Variation: 0.0271 DETAILED CLASSIFICATION REPORT: precision recall f1-score support 0 0.96 0.98 0.97 50 1 0.97 0.93 0.95 30 accuracy 0.96 80 macro avg 0.96 0.96 0.96 80 weighted avg 0.96 0.96 0.96 80	96.25%	

LightGBM	COMPREHENSIVE ANALYSIS: LightGBM == 📊 PERFORMANCE METRICS: Test Accuracy: 1.0000 Precision: 1.0000 Recall: 1.0000 F1 Score: 1.0000 ROC AUC: 1.0000 PR AUC: 1.0000 🔍 ENHANCED OVERFITTING ANALYSIS: Train F1 Score: 0.9905 Test F1 Score: 1.0000 F1 Performance Gap: -0.0095 Accuracy Gap: -0.0095 Overfitting Score: 0.800 Overfitting Risk: CRITICAL Overfitting Indicators: • Perfect test score detected (99.9%+) - possible data leakage 🎯 STRATIFIED K-FOLD CROSS VALIDATION: CV Scores: ['0.9844', '0.9842', '0.9842', '0.9842', '0.9841'] CV Mean: 0.9842 CV Std: 0.0001 CV Coefficient of Variation: 0.0001 📋 DETAILED CLASSIFICATION REPORT: precision recall f1-score support 0 1.00 1.00 1.00 50 1 1.00 1.00 1.00 30 accuracy 1.00 80 macro avg 1.00 1.00 1.00 80 weighted avg 1.00 1.00 1.00 80	100%	
Logistic Regression	COMPREHENSIVE ANALYSIS: Logistic Regression == 📊 PERFORMANCE METRICS: Test Accuracy: 1.0000 Precision: 1.0000 Recall: 1.0000 F1 Score: 1.0000 ROC AUC: 1.0000 PR AUC: 1.0000 🔍 ENHANCED OVERFITTING ANALYSIS: Train F1 Score: 0.9968 Test F1 Score: 1.0000 F1 Performance Gap: -0.0032 Accuracy Gap: -0.0032 Overfitting Score: 1.000 Overfitting Risk: CRITICAL Overfitting Indicators: • High confidence predictions: 87.0% of training predictions are >95% or <5% confident • Perfect test score detected (99.9%+) - possible data leakage 🎯 STRATIFIED K-FOLD CROSS VALIDATION: CV Scores: ['0.9844', '1.0000', '0.9842', '0.9842', '0.9842'] CV Mean: 0.9874 CV Std: 0.0063 CV Coefficient of Variation: 0.0064 📋 DETAILED CLASSIFICATION REPORT: precision recall f1-score support 0 1.00 1.00 1.00 50 1 1.00 1.00 1.00 30 accuracy 1.00 80 macro avg 1.00 1.00 1.00 80 weighted avg 1.00 1.00 1.00 80	100%	

SVM (RBF)	COMPREHENSIVE ANALYSIS: SVM (RBF) == 📊 PERFORMANCE METRICS: Test Accuracy: 1.0000 Precision: 1.0000 Recall: 1.0000 F1 Score: 1.0000 ROC AUC: 1.0000 PR AUC: 1.0000 🔍 ENHANCED OVERFITTING ANALYSIS: Train F1 Score: 1.0000 Test F1 Score: 1.0000 F1 Performance Gap: 0.0000 Accuracy Gap: 0.0000 Overfitting Score: 1.000 Overfitting Risk: CRITICAL Overfitting Indicators: • High confidence predictions: 97.5% of training predictions are >95% or <5% confident • Perfect training score detected (99.9%+) - highly suspicious • Perfect test score detected (99.9%+) - possible data leakage 🎯 STRATIFIED K-FOLD CROSS VALIDATION: CV Scores: ['0.9844', '0.9842', '1.0000', '1.0000', '1.0000'] CV Mean: 0.9937 CV Std: 0.0077 CV Coefficient of Variation: 0.0077 📋 DETAILED CLASSIFICATION REPORT: precision recall f1-score support 0 1.00 1.00 1.00 50 1 1.00 1.00 1.00 30 accuracy 1.00 80 macro avg 1.00 1.00 1.00 80 weighted avg 1.00 1.00 1.00 80	100%	
Gradient Boosting	COMPREHENSIVE ANALYSIS: Gradient Boosting == 📊 PERFORMANCE METRICS: Test Accuracy: 0.9500 Precision: 0.9500 Recall: 0.9500 F1 Score: 0.9500 ROC AUC: 0.9947 PR AUC: 0.9920 🔍 ENHANCED OVERFITTING ANALYSIS: Train F1 Score: 0.9874 Test F1 Score: 0.9500 F1 Performance Gap: 0.0374 Accuracy Gap: 0.0373 Overfitting Score: 0.187 Overfitting Risk: LOW 🎯 STRATIFIED K-FOLD CROSS VALIDATION: CV Scores: ['0.9844', '0.9842', '0.9528', '0.9842', '0.9841'] CV Mean: 0.9779 CV Std: 0.0126 CV Coefficient of Variation: 0.0128 📋 DETAILED CLASSIFICATION REPORT: precision recall f1-score support 0 0.96 0.96 0.96 50 1 0.93 0.93 0.93 30 accuracy 0.95 80 macro avg 0.95 0.95 0.95 80 weighted avg 0.95 0.95 0.95 80	95%	

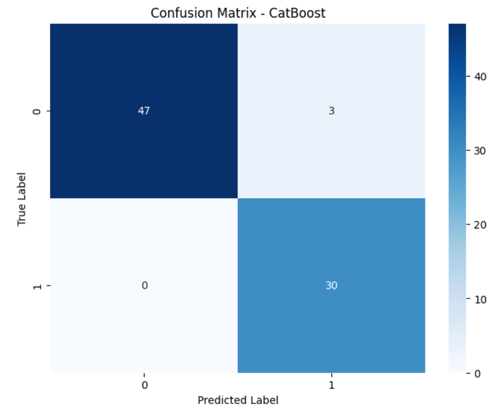
<table>
<tr>
<td>CatBoost</td>
<td>COMPREHENSIVE ANALYSIS: CatBoost
==
PERFORMANCE METRICS:
Test Accuracy: 0.9625
Precision: 0.9659
Recall: 0.9625
F1 Score: 0.9628
ROC AUC: 0.9973
PR AUC: 0.9958

ENHANCED OVERFITTING ANALYSIS:
Train F1 Score: 0.9874
Test F1 Score: 0.9628
F1 Performance Gap: 0.0246
Accuracy Gap: 0.0248
Overfitting Score: 0.123
Overfitting Risk: LOW

STRATIFIED K-FOLD CROSS VALIDATION:
CV Scores: ['1.0000', '0.9842', '0.9842', '0.9685', '0.9526']
CV Mean: 0.9779
CV Std: 0.0161
CV Coefficient of Variation: 0.0165

DETAILED CLASSIFICATION REPORT:
precision recall f1-score support

0 1.00 0.94 0.97 50
1 0.91 1.00 0.95 30

accuracy 0.96 80
macro avg 0.95 0.97 0.96 80
weighted avg 0.97 0.96 0.96 80</td>
<td>96.25%</td>
<td>
</td>
</tr>
</table>

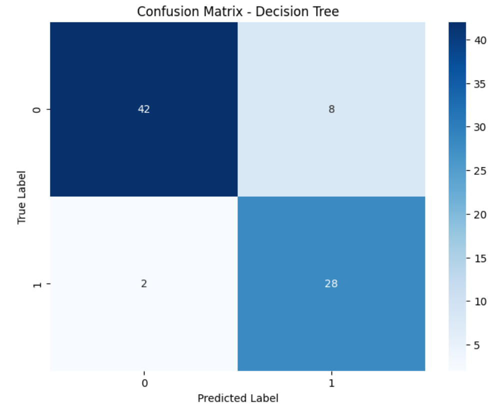
<table>
<tr>
<td>Decision Tree</td>
<td>COMPREHENSIVE ANALYSIS: Decision Tree
==
📊 PERFORMANCE METRICS:
Test Accuracy: 0.8750
Precision: 0.8883
Recall: 0.8750
F1 Score: 0.8767
ROC AUC: 0.8867
PR AUC: 0.7509

🔍 ENHANCED OVERFITTING ANALYSIS:
Train F1 Score: 0.9561
Test F1 Score: 0.8767
F1 Performance Gap: 0.0794
Accuracy Gap: 0.0807
Overfitting Score: 0.397
Overfitting Risk: MEDIUM
Overfitting Indicators:
• Large F1 gap: 0.079 (train: 0.956, test: 0.877)
• Large accuracy gap: 0.081

🎯 STRATIFIED K-FOLD CROSS VALIDATION:
CV Scores: ['0.9690', '0.9685', '0.9217', '0.9685', '0.9369']
CV Mean: 0.9529
CV Std: 0.0199
CV Coefficient of Variation: 0.0209

📋 DETAILED CLASSIFICATION REPORT:
precision recall f1-score support

0 0.95 0.84 0.89 50
1 0.78 0.93 0.85 30

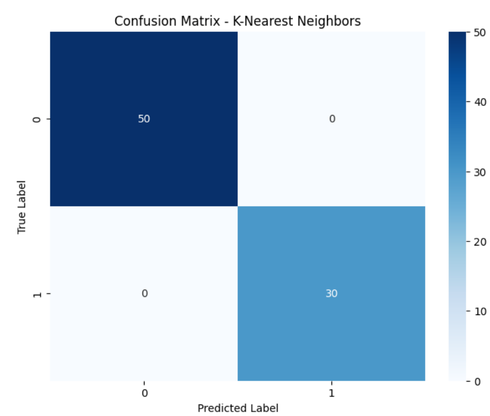
accuracy 0.88 80
macro avg 0.87 0.89 0.87 80
weighted avg 0.89 0.88 0.88 80</td>
<td>87.50%</td>
<td>
</td>
</tr>
<tr>
<td>K-Nearest Neighbors</td>
<td>COMPREHENSIVE ANALYSIS: K-Nearest Neighbors
==
📊 PERFORMANCE METRICS:
Test Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1 Score: 1.0000
ROC AUC: 1.0000
PR AUC: 1.0000

🔍 ENHANCED OVERFITTING ANALYSIS:
Train F1 Score: 1.0000
Test F1 Score: 1.0000
F1 Performance Gap: 0.0000
Accuracy Gap: 0.0000
Overfitting Score: 1.000
Overfitting Risk: CRITICAL
Overfitting Indicators:
• High confidence predictions: 100.0% of training predictions are >95% or <5% confident
• Perfect training score detected (99.9%+) - highly suspicious
• Perfect test score detected (99.9%+) - possible data leakage

🎯 STRATIFIED K-FOLD CROSS VALIDATION:
CV Scores: ['0.9844', '0.9685', '0.9528', '0.9842', '1.0000']
CV Mean: 0.9780
CV Std: 0.0161
CV Coefficient of Variation: 0.0164

📋 DETAILED CLASSIFICATION REPORT:
precision recall f1-score support

0 1.00 1.00 1.00 50
1 1.00 1.00 1.00 30

accuracy 1.00 80
macro avg 1.00 1.00 1.00 80
weighted avg 1.00 1.00 1.00 80</td>
<td>100%</td>
<td>
</td>
</tr>
</table>

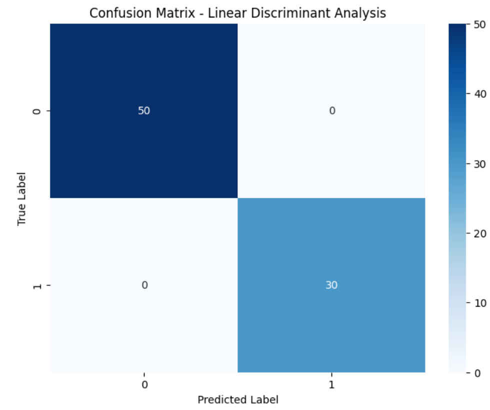
<table>
<tr>
<td>Linear Discriminant Analysis</td>
<td>COMPREHENSIVE ANALYSIS: Linear Discriminant Analysis
==
📊 PERFORMANCE METRICS:
Test Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1 Score: 1.0000
ROC AUC: 1.0000
PR AUC: 1.0000

🔍 ENHANCED OVERFITTING ANALYSIS:
Train F1 Score: 0.9779
Test F1 Score: 1.0000
F1 Performance Gap: -0.0221
Accuracy Gap: -0.0222
Overfitting Score: 1.000
Overfitting Risk: CRITICAL
Overfitting Indicators:
• High confidence predictions: 93.0% of training predictions are >95% or <5% confident
• Perfect test score detected (99.9%+) - possible data leakage

🎯 STRATIFIED K-FOLD CROSS VALIDATION:
CV Scores: ['1.0000', '0.9842', '0.9528', '0.9369', '1.0000']
CV Mean: 0.9748
CV Std: 0.0256
CV Coefficient of Variation: 0.0263

📋 DETAILED CLASSIFICATION REPORT:
precision recall f1-score support

0 1.00 1.00 1.00 50
1 1.00 1.00 1.00 30

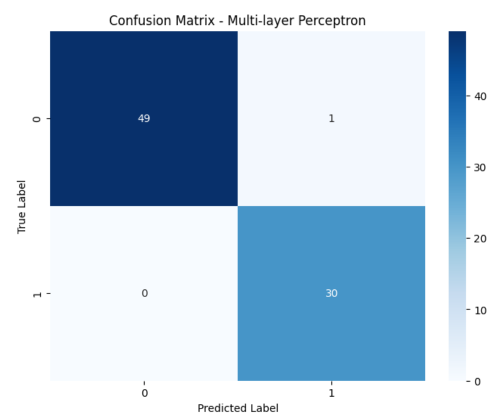
accuracy 1.00 80
macro avg 1.00 1.00 1.00 80
weighted avg 1.00 1.00 1.00 80</td>
<td>100%</td>
<td>
</td>
</tr>
<tr>
<td>Multi-Layer Perceptron</td>
<td>COMPREHENSIVE ANALYSIS: Multi-layer Perceptron
==
📊 PERFORMANCE METRICS:
Test Accuracy: 0.9875
Precision: 0.9879
Recall: 0.9875
F1 Score: 0.9875
ROC AUC: 1.0000
PR AUC: 1.0000

🔍 ENHANCED OVERFITTING ANALYSIS:
Train F1 Score: 0.9717
Test F1 Score: 0.9875
F1 Performance Gap: -0.0158
Accuracy Gap: -0.0160
Overfitting Score: 0.000
Overfitting Risk: LOW

🎯 STRATIFIED K-FOLD CROSS VALIDATION:
CV Scores: ['0.9690', '0.9528', '0.9685', '0.9685', '0.9842']
CV Mean: 0.9686
CV Std: 0.0099
CV Coefficient of Variation: 0.0102

📋 DETAILED CLASSIFICATION REPORT:
precision recall f1-score support

0 1.00 0.98 0.99 50
1 0.97 1.00 0.98 30

accuracy 0.99 80
macro avg 0.98 0.99 0.99 80
weighted avg 0.99 0.99 0.99 80</td>
<td>98.75%</td>
<td>
</td>
</tr>
</table>

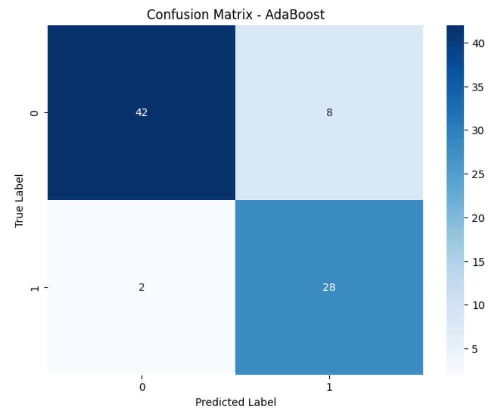
<table>
<tr>
<td>AdaBoost</td>
<td>COMPREHENSIVE ANALYSIS: AdaBoost
==
📊 PERFORMANCE METRICS:
Test Accuracy: 0.8750
Precision: 0.8883
Recall: 0.8750
F1 Score: 0.8767
ROC AUC: 0.8867
PR AUC: 0.7509

🔍 ENHANCED OVERFITTING ANALYSIS:
Train F1 Score: 0.9561
Test F1 Score: 0.8767
F1 Performance Gap: 0.0794
Accuracy Gap: 0.0807
Overfitting Score: 0.397
Overfitting Risk: MEDIUM
Overfitting Indicators:
• Large F1 gap: 0.079 (train: 0.956, test: 0.877)
• Large accuracy gap: 0.081

🎯 STRATIFIED K-FOLD CROSS VALIDATION:
CV Scores: ['0.9690', '0.9685', '0.9217', '0.9685', '0.9369']
CV Mean: 0.9529
CV Std: 0.0199
CV Coefficient of Variation: 0.0209

📋 DETAILED CLASSIFICATION REPORT:
precision recall f1-score support

0 0.95 0.84 0.89 50
1 0.78 0.93 0.85 30

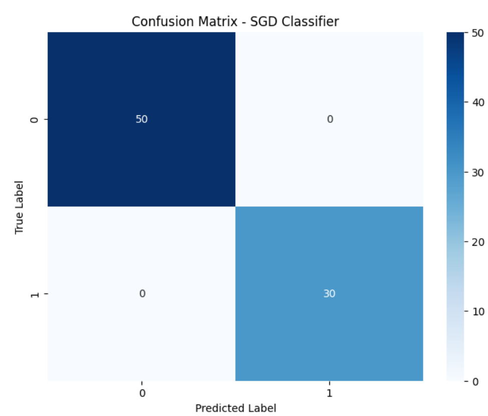
accuracy 0.88 80
macro avg 0.87 0.89 0.87 80
weighted avg 0.89 0.88 0.88 80</td>
<td>87.50%</td>
<td>
</td>
</tr>
<tr>
<td>SGD Classifier</td>
<td>COMPREHENSIVE ANALYSIS: SGD Classifier
==
📊 PERFORMANCE METRICS:
Test Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1 Score: 1.0000
ROC AUC: 1.0000
PR AUC: 1.0000

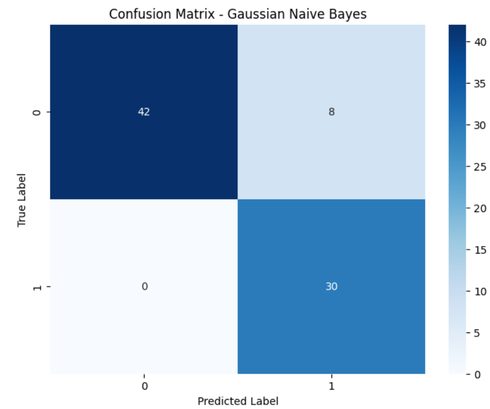
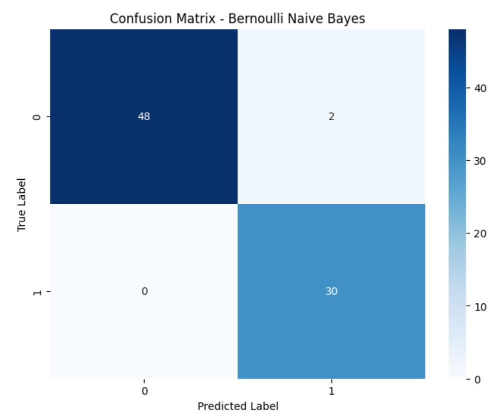
🔍 ENHANCED OVERFITTING ANALYSIS:
Train F1 Score: 0.9937
Test F1 Score: 1.0000
F1 Performance Gap: -0.0063
Accuracy Gap: -0.0063
Overfitting Score: 1.000
Overfitting Risk: CRITICAL
Overfitting Indicators:
• High confidence predictions: 97.2% of training predictions are >95% or <5% confident
• Perfect test score detected (99.9%+) - possible data leakage

🎯 STRATIFIED K-FOLD CROSS VALIDATION:
CV Scores: ['0.9844', '1.0000', '0.9842', '0.9683', '0.9842']
CV Mean: 0.9842
CV Std: 0.0100
CV Coefficient of Variation: 0.0102

📋 DETAILED CLASSIFICATION REPORT:
precision recall f1-score support

0 1.00 1.00 1.00 50
1 1.00 1.00 1.00 30

accuracy 1.00 80
macro avg 1.00 1.00 1.00 80
weighted avg 1.00 1.00 1.00 80</td>
<td>100%</td>
<td>
</td>
</tr>
</table>

Gaussian Naïve Bayes	COMPREHENSIVE ANALYSIS: Gaussian Naive Bayes == 📊 PERFORMANCE METRICS: Test Accuracy: 0.9000 Precision: 0.9211 Recall: 0.9000 F1 Score: 0.9015 ROC AUC: 1.0000 PR AUC: 1.0000 🔍 ENHANCED OVERFITTING ANALYSIS: Train F1 Score: 0.9219 Test F1 Score: 0.9015 F1 Performance Gap: 0.0204 Accuracy Gap: 0.0209 Overfitting Score: 0.402 Overfitting Risk: MEDIUM Overfitting Indicators: • High confidence predictions: 99.7% of training predictions are >95% or <5% confident 🎯 STRATIFIED K-FOLD CROSS VALIDATION: CV Scores: ['0.8616', '0.9372', '0.9528', '0.9061', '0.9528'] CV Mean: 0.9221 CV Std: 0.0348 CV Coefficient of Variation: 0.0377 📋 DETAILED CLASSIFICATION REPORT: precision recall f1-score support 0 1.00 0.84 0.91 50 1 0.79 1.00 0.88 30 accuracy 0.90 80 macro avg 0.89 0.92 0.90 80 weighted avg 0.92 0.90 0.90 80	90%	
Bernoulli Naïve Bayes	COMPREHENSIVE ANALYSIS: Bernoulli Naive Bayes == 📊 PERFORMANCE METRICS: Test Accuracy: 0.9750 Precision: 0.9766 Recall: 0.9750 F1 Score: 0.9751 ROC AUC: 1.0000 PR AUC: 1.0000 🔍 ENHANCED OVERFITTING ANALYSIS: Train F1 Score: 0.9874 Test F1 Score: 0.9751 F1 Performance Gap: 0.0122 Accuracy Gap: 0.0123 Overfitting Score: 0.361 Overfitting Risk: MEDIUM Overfitting Indicators: • High confidence predictions: 97.2% of training predictions are >95% or <5% confident 🎯 STRATIFIED K-FOLD CROSS VALIDATION: CV Scores: ['1.0000', '0.9842', '0.9842', '0.9685', '0.9841'] CV Mean: 0.9842 CV Std: 0.0100 CV Coefficient of Variation: 0.0101 📋 DETAILED CLASSIFICATION REPORT: precision recall f1-score support 0 1.00 0.96 0.98 50 1 0.94 1.00 0.97 30 accuracy 0.97 80 macro avg 0.97 0.98 0.97 80 weighted avg 0.98 0.97 0.98 80	97.50%	

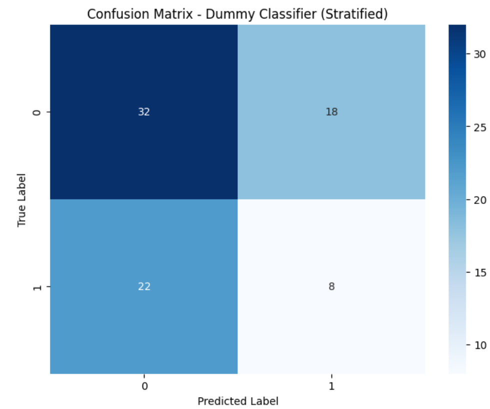
<table>
<tr>
<td>Dummy Classifier (Stratified)</td>
<td>COMPREHENSIVE ANALYSIS: Dummy Classifier (Stratified)
==
📊 PERFORMANCE METRICS:
Test Accuracy: 0.5000
Precision: 0.4858
Recall: 0.5000
F1 Score: 0.4918
ROC AUC: 0.4533
PR AUC: 0.3571

🔍 ENHANCED OVERFITTING ANALYSIS:
Train F1 Score: 0.5709
Test F1 Score: 0.4918
F1 Performance Gap: 0.0792
Accuracy Gap: 0.0696
Overfitting Score: 1.000
Overfitting Risk: CRITICAL
Overfitting Indicators:
• High confidence predictions: 100.0% of training predictions are >95% or <5% confident
• Low test performance (0.492) with large gap (0.079) suggests learning noise
• Large F1 gap: 0.079 (train: 0.571, test: 0.492)
• Large accuracy gap: 0.070

🎯 STRATIFIED K-FOLD CROSS VALIDATION:
CV Scores: ['0.5982', '0.5594', '0.4941', '0.5594', '0.4615']
CV Mean: 0.5345
CV Std: 0.0495
CV Coefficient of Variation: 0.0926

📋 DETAILED CLASSIFICATION REPORT:
precision recall f1-score support

0 0.59 0.64 0.62 50
1 0.31 0.27 0.29 30

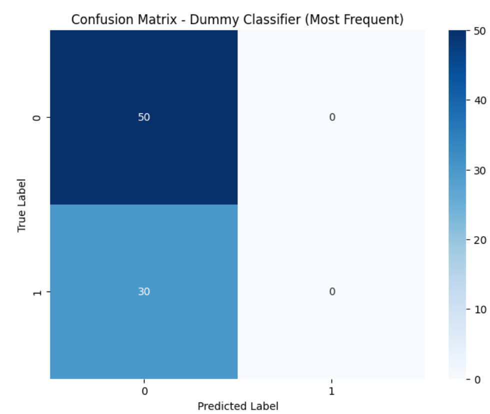
accuracy 0.50 80
macro avg 0.45 0.45 0.45 80
weighted avg 0.49 0.50 0.49 80</td>
<td>50%</td>
<td></td>
</tr>
<tr>
<td>Dummy Classifier (Most Frequent)</td>
<td>COMPREHENSIVE ANALYSIS: Dummy Classifier (Most Frequent)
==
📊 PERFORMANCE METRICS:
Test Accuracy: 0.6250
Precision: 0.3906
Recall: 0.6250
F1 Score: 0.4808
ROC AUC: 0.5000
PR AUC: 0.3750

🔍 ENHANCED OVERFITTING ANALYSIS:
Train F1 Score: 0.4749
Test F1 Score: 0.4808
F1 Performance Gap: -0.0059
Accuracy Gap: -0.0047
Overfitting Score: 0.300
Overfitting Risk: MEDIUM
Overfitting Indicators:
• High confidence predictions: 100.0% of training predictions are >95% or <5% confident

🎯 STRATIFIED K-FOLD CROSS VALIDATION:
CV Scores: ['0.4808', '0.4734', '0.4734', '0.4734', '0.4734']
CV Mean: 0.4749
CV Std: 0.0030
CV Coefficient of Variation: 0.0062

📋 DETAILED CLASSIFICATION REPORT:
precision recall f1-score support

0 0.62 1.00 0.77 50
1 0.00 0.00 0.00 30

accuracy 0.62 80
macro avg 0.31 0.50 0.38 80
weighted avg 0.39 0.62 0.48 80</td>
<td>62.50%</td>
<td></td>
</tr>
</table>