Sri Sivasubramaniya Nadar College of Engineering, Chennai

(An autonomous Institution affiliated to Anna University)

Degree & Branch	M.Tech (Integrated) Computer Science & Engineering
Semester	V
Subject Code & Name	ICS1512 – Machine Learning Algorithms Laboratory
Academic Year	2025–2026 (Odd)
Batch	2023–2028

Name: I.S.Rajesh **Register No.: 3122237001042**

Experiment 6: Dimensionality Reduction and Model Evaluation (With and Without PCA)

Aim

To compare the performance of various ML models when dimensionality reduction is or is not applied.

Libraries Used

`numpy, pandas, sklearn, matplotlib, seaborn, xgboost`

Objective

To study the effect of dimensionality reduction using Principal Component Analysis (PCA) on the performance of various machine learning classifiers by doing the following:

- **Training and validating models without PCA (original feature space)**.
- **Training and validating models with PCA (reduced feature space)**.

For both cases, perform hyperparameter tuning, apply 5-fold cross-validation, and record performance.

Dataset

- **Dataset source:** Wisconsin Diagnostic Dataset (UCI)
- **569 samples and 30 numerical features** representing cell nuclei characteristics from digitized images.
- **Target label (Diagnostic)** is binary.
- **Dataset is distributed almost normally** for all (real) features.

Preprocessing Steps

- **Outliers:** Replace values outside IQR with mean if feature is normally distributed, else median.
- **Missing values:** Replace categorical values with mode. For numerical values, replace with median if distribution is non-normal or there are outliers, else mean.
- **Encoding:** Perform label encoding or target-guided encoding depending on the type of model used.
- **Standardization:** Use min-max normalization if there are outliers or non-normally distributed, else standard normalization.

PCA Design Choice Code

Below are the 5 main code sections used for PCA implementation and integration into model pipelines.

Code 1: Naive Bayes

Listing 1: Code 1: Naive Bayes

```
# --- Gaussian NB preprocessing ---
le = LabelEncoder()
df_gauss[target] = le.fit_transform(df_gauss[target])
for col in categorical_cols:
    df_gauss = pd.get_dummies(df_gauss, columns=categorical_cols)

for col in numerical_cols:
    if is_normal(df_gauss[col]):
        scaler = StandardScaler()
    elif has_outliers(df_gauss[col]):
        scaler = StandardScaler()
    else:
        scaler = MinMaxScaler()
    df_gauss[[col]] = scaler.fit_transform(df_gauss[[col]])

# --- Multinomial NB preprocessing ---
le = LabelEncoder()
df_multi[target] = le.fit_transform(df_multi[target])
df_multi = pd.get_dummies(df_multi, columns=categorical_cols,
   drop_first=False)

for col in numerical_cols:
    scaler = MinMaxScaler()
    df_multi[[col]] = scaler.fit_transform(df_multi[[col]])

# --- Bernoulli NB preprocessing ---
le = LabelEncoder()
df_berno[target] = le.fit_transform(df_berno[target])
```

```python
df_berno = pd.get_dummies(df_berno, columns=categorical_cols,
   drop_first=True)

for col in numerical_cols:
    binarizer = Binarizer(threshold=0.0)
    df_berno[[col]] = binarizer.fit_transform(df_berno[[col]])

# Splitting dataset and training model

results = []

def apply_pca(X_train, X_val, X_test, n_components=0.95):
    pca = PCA(n_components=n_components)
    X_train_pca = pca.fit_transform(X_train)
    X_val_pca   = pca.transform(X_val)
    X_test_pca  = pca.transform(X_test)
    return X_train_pca, X_val_pca, X_test_pca, pca

models = {
    "GaussianNB": (GaussianNB(), df_gauss),
    "MultinomialNB": (MultinomialNB(), df_multi),
    "BernoulliNB": (BernoulliNB(), df_berno)
}

for name, (model, data) in models.items():
    print(f"\n{'='*40}\nProcessing {name}\n{'='*40}")

    X = data.drop(columns=[target])
    y = data[target]

    X_train, X_temp, y_train, y_temp = train_test_split(X, y,
       test_size=0.4, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_temp,
       y_temp, test_size=0.5, random_state=42)

    # ---------------- Non-PCA ----------------
    model.fit(X_train, y_train)
    y_val_pred = model.predict(X_val)
    y_test_pred = model.predict(X_test)

    print(f"\n--- {name} (No PCA) ---")
    evaluate_model(y_test, y_test_pred, True, X_test, model, "
       Test Set")
    plot_confusion_matrix(y_test, y_test_pred, f"{name} (No PCA)
       - Test Set")
    plot_confusion_matrix(y_val, y_val_pred, f"{name} (No PCA) -
       Validation Set")

    # Cross-validation
    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
```

```
    cv_scores = cross_val_score(model, X, y, cv=kfold, scoring='
       accuracy')
    print("Cross Validation Scores (No PCA):", cv_scores)
    print("Average CV Score (No PCA):", np.mean(cv_scores))

    results.append((name, "No PCA", accuracy_score(y_test,
       y_test_pred), np.mean(cv_scores)))

    # ---------------- PCA ----------------
    # Standard PCA transform
    X_train_pca, X_val_pca, X_test_pca, pca = apply_pca(X_train,
       X_val, X_test)

    # If model is MultinomialNB, apply MinMaxScaler to make PCA
       outputs non-negative
    if name == "MultinomialNB":
        scaler = MinMaxScaler()
        X_train_pca = scaler.fit_transform(X_train_pca)
        X_val_pca   = scaler.transform(X_val_pca)
        X_test_pca  = scaler.transform(X_test_pca)
        X_pca_full  = scaler.fit_transform(PCA(n_components=0.95)
           .fit_transform(X))
    else:
        X_pca_full  = PCA(n_components=0.95).fit_transform(X)

    model_pca = model.__class__()  # fresh instance
    model_pca.fit(X_train_pca, y_train)
    y_val_pred_pca = model_pca.predict(X_val_pca)
    y_test_pred_pca = model_pca.predict(X_test_pca)

    print(f"\n--- {name} (PCA) ---")
    evaluate_model(y_test, y_test_pred_pca, True, X_test_pca,
       model_pca, "Test Set")
    plot_confusion_matrix(y_test, y_test_pred_pca, f"{name} (PCA)
        - Test Set")
    plot_confusion_matrix(y_val, y_val_pred_pca, f"{name} (PCA) -
        Validation Set")

    # Cross-validation (PCA)
    cv_scores_pca = cross_val_score(model_pca, X_pca_full, y, cv=
       kfold, scoring='accuracy')
    print("Cross Validation Scores (PCA):", cv_scores_pca)
    print("Average CV Score (PCA):", np.mean(cv_scores_pca))

    results.append((name, "PCA", accuracy_score(y_test,
       y_test_pred_pca), np.mean(cv_scores_pca)))
```

Code 2: kNN

Listing 2: Code 2: kNN

```python
# Splitting dataset and training model

results = []

def apply_pca(X_train, X_val, X_test, n_components=0.95):
    pca = PCA(n_components=n_components)
    X_train_pca = pca.fit_transform(X_train)
    X_val_pca   = pca.transform(X_val)
    X_test_pca  = pca.transform(X_test)
    return X_train_pca, X_val_pca, X_test_pca, pca

models = {
    "KNN (auto)": (KNeighborsClassifier(n_neighbors=5, metric='minkowski'), df),
    "KNN (kd_tree)": (KNeighborsClassifier(n_neighbors=5, algorithm='kd_tree', metric='minkowski'), df),
    "KNN (ball_tree)": (KNeighborsClassifier(n_neighbors=5, algorithm='ball_tree', metric='minkowski'), df)
}

for name, (model, data) in models.items():
    print(f"\n{'='*40}\nProcessing {name}\n{'='*40}")

    X = data.drop(columns=[target])
    y = data[target]

    X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

    # ----------- Non-PCA -----------
    model.fit(X_train, y_train)
    y_val_pred = model.predict(X_val)
    y_test_pred = model.predict(X_test)

    print(f"\n--- {name} (No PCA) ---")
    evaluate_model(y_test, y_test_pred, True, X_test, model, "Test Set")
    plot_confusion_matrix(y_test, y_test_pred, f"{name} (No PCA) - Test Set")
    plot_confusion_matrix(y_val, y_val_pred, f"{name} (No PCA) - Validation Set")

    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    cv_scores = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')
    print("Cross Validation Scores (No PCA):", cv_scores)
    print("Average CV Score (No PCA):", np.mean(cv_scores))
```

```
    results.append((name, "No PCA", accuracy_score(y_test,
        y_test_pred), np.mean(cv_scores)))

    # ----------- PCA -----------
    X_train_pca, X_val_pca, X_test_pca, pca = apply_pca(X_train,
        X_val, X_test)
    X_pca_full = PCA(n_components=0.95).fit_transform(X)

    model_pca = model.__class__(**model.get_params())  # re-init
        with same params
    model_pca.fit(X_train_pca, y_train)
    y_val_pred_pca = model_pca.predict(X_val_pca)
    y_test_pred_pca = model_pca.predict(X_test_pca)

    print(f"\n--- {name} (PCA) ---")
    evaluate_model(y_test, y_test_pred_pca, True, X_test_pca,
        model_pca, "Test Set")
    plot_confusion_matrix(y_test, y_test_pred_pca, f"{name} (PCA)
         - Test Set")
    plot_confusion_matrix(y_val, y_val_pred_pca, f"{name} (PCA) -
         Validation Set")

    cv_scores_pca = cross_val_score(model_pca, X_pca_full, y, cv=
        kfold, scoring='accuracy')
    print("Cross Validation Scores (PCA):", cv_scores_pca)
    print("Average CV Score (PCA):", np.mean(cv_scores_pca))

    results.append((name, "PCA", accuracy_score(y_test,
        y_test_pred_pca), np.mean(cv_scores_pca)))
```

Code 3: SVM

Listing 3: Code 3: SVM

```
# ---------------- Linear Kernel ----------------
param_grid_l = {'C': [0.1, 1.0, 10]}
grid_l = GridSearchCV(SVR(kernel='linear'), param_grid_l, cv=5)
grid_l.fit(X_train, y_train)

model_l = grid_l.best_estimator_
y_val_pred_l = model_l.predict(X_val)
y_test_pred_l = model_l.predict(X_test)

# ----- Linear Kernel with PCA -----
pipe_l = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA()),
    ('svr', SVR(kernel='linear'))
])
param_grid_l_pca = {
    'pca__n_components': [5, 10, 15, 0.90, 0.95],
```

```
    'svr__C': [0.1, 1.0, 10]
}
grid_l_pca = GridSearchCV(pipe_l, param_grid_l_pca, cv=5)
grid_l_pca.fit(X_train, y_train)

model_l_pca = grid_l_pca.best_estimator_
y_val_pred_l_pca = model_l_pca.predict(X_val)
y_test_pred_l_pca = model_l_pca.predict(X_test)


# ---------------- Polynomial Kernel ----------------
param_grid_p = {
    'C': [0.1, 1.0, 10],
    'degree': [2, 3, 4],
    'gamma': ['scale', 'auto']
}
grid_p = GridSearchCV(SVR(kernel='poly'), param_grid_p, cv=5)
grid_p.fit(X_train, y_train)

model_p = grid_p.best_estimator_
y_val_pred_p = model_p.predict(X_val)
y_test_pred_p = model_p.predict(X_test)

# ----- Polynomial Kernel with PCA -----
pipe_p = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA()),
    ('svr', SVR(kernel='poly'))
])
param_grid_p_pca = {
    'pca__n_components': [5, 10, 15, 0.90, 0.95],
    'svr__C': [0.1, 1.0, 10],
    'svr__degree': [2, 3, 4],
    'svr__gamma': ['scale', 'auto']
}
grid_p_pca = GridSearchCV(pipe_p, param_grid_p_pca, cv=5)
grid_p_pca.fit(X_train, y_train)

model_p_pca = grid_p_pca.best_estimator_
y_val_pred_p_pca = model_p_pca.predict(X_val)
y_test_pred_p_pca = model_p_pca.predict(X_test)


# ---------------- RBF Kernel ----------------
param_grid_r = {
    'C': [0.1, 1.0, 10],
    'gamma': ['scale', 'auto']
}
grid_r = GridSearchCV(SVR(kernel='rbf'), param_grid_r, cv=5)
grid_r.fit(X_train, y_train)
```

```
model_r = grid_r.best_estimator_
y_val_pred_r = model_r.predict(X_val)
y_test_pred_r = model_r.predict(X_test)

# ----- RBF Kernel with PCA -----
pipe_r = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA()),
    ('svr', SVR(kernel='rbf'))
])
param_grid_r_pca = {
    'pca__n_components': [5, 10, 15, 0.90, 0.95],
    'svr__C': [0.1, 1.0, 10],
    'svr__gamma': ['scale', 'auto']
}
grid_r_pca = GridSearchCV(pipe_r, param_grid_r_pca, cv=5)
grid_r_pca.fit(X_train, y_train)

model_r_pca = grid_r_pca.best_estimator_
y_val_pred_r_pca = model_r_pca.predict(X_val)
y_test_pred_r_pca = model_r_pca.predict(X_test)


# ---------------- Sigmoid Kernel ----------------
param_grid_s = {
    'C': [0.1, 1.0, 10],
    'gamma': ['scale', 'auto']
}
grid_s = GridSearchCV(SVR(kernel='sigmoid'), param_grid_s, cv=5)
grid_s.fit(X_train, y_train)

model_s = grid_s.best_estimator_
y_val_pred_s = model_s.predict(X_val)
y_test_pred_s = model_s.predict(X_test)

# ----- Sigmoid Kernel with PCA -----
pipe_s = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA()),
    ('svr', SVR(kernel='sigmoid'))
])
param_grid_s_pca = {
    'pca__n_components': [5, 10, 15, 0.90, 0.95],
    'svr__C': [0.1, 1.0, 10],
    'svr__gamma': ['scale', 'auto']
}
grid_s_pca = GridSearchCV(pipe_s, param_grid_s_pca, cv=5)
grid_s_pca.fit(X_train, y_train)

model_s_pca = grid_s_pca.best_estimator_
y_val_pred_s_pca = model_s_pca.predict(X_val)
```

```
y_test_pred_s_pca = model_s_pca.predict(X_test)
```

Code 4: Logistic Regression

Listing 4: Code 4: Logistic Regression

```
def run_logistic_regression_with_pca(X_train, X_val, X_test,
   y_train, y_val, y_test, use_pca=False):
    # Base pipeline (scaling is important for LR)
    steps = [("scaler", StandardScaler())]

    # Add PCA if requested
    if use_pca:
        steps.append(("pca", PCA(n_components=0.95, random_state
           =42)))  # keep 95% variance

    steps.append(("logreg", LogisticRegression(max_iter=500,
       random_state=42)))
    pipe = Pipeline(steps)

    # Hyperparameter grid
    param_grid = {
        "logreg__C": [0.01, 0.1, 1, 10, 100],
        "logreg__penalty": ["l2"],  # stick with l2 (safest
           across solvers)
        "logreg__solver": ["lbfgs", "saga"]  # support l2
    }

    grid = GridSearchCV(
        pipe, param_grid, cv=5, scoring="accuracy", n_jobs=-1
    )
    grid.fit(X_train, y_train)

    best_model = grid.best_estimator_
    print(f"{'PCA' if use_pca else 'No PCA'} Best Params:", grid.
       best_params_)

    # Predictions
    y_val_pred = best_model.predict(X_val)
    y_test_pred = best_model.predict(X_test)

    return best_model, y_val_pred, y_test_pred

X = df.drop(columns=[target])
y = df[target]

# Train / temp split
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.4, random_state=42, stratify=y
)
```

```python
# Validation / test split
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=42, stratify=
        y_temp
)

# Run both versions
best_models = {}
y_val_preds = {}
y_test_preds = {}

for label, use_pca in [("LogReg_NoPCA", False), ("LogReg_PCA",
   True)]:
    model, y_val, y_test = run_logistic_regression_with_pca(
        X_train, X_val, X_test, y_train, y_val, y_test, use_pca=
            use_pca
    )
    best_models[label] = model
    y_val_preds[label] = y_val
    y_test_preds[label] = y_test
```

Code 5: Ensemble

Listing 5: Code 5: Ensemble

```python
# =====================================================
# Storage for results
# =====================================================
best_models = {}
y_val_preds = {}
y_test_preds = {}

scoring = 'accuracy'

# -----------------------------------------------------
# Helper: Train, store, print results
# -----------------------------------------------------
def train_and_store(name, grid, X_train, y_train, X_val, X_test,
   y_val, y_test, pca_suffix=""):
    grid.fit(X_train, y_train)
    model = grid.best_estimator_
    y_val_pred = model.predict(X_val)
    y_test_pred = model.predict(X_test)

    key = f"{name}{pca_suffix}"
    best_models[key] = model
    y_val_preds[key] = y_val_pred
    y_test_preds[key] = y_test_pred

    print(f"\n{name}{pca_suffix} best params:", grid.best_params_
        )
```

```
    print(f"Validation Accuracy: {accuracy_score(y_val,
       y_val_pred):.4f}")
    print(f"Test Accuracy: {accuracy_score(y_test, y_test_pred)
       :.4f}")
    return model

# ----------------------------------------------------
# Helper: PCA transform
# ----------------------------------------------------
def get_pca_data(X_train, X_val, X_test, n_components=0.95):
    pca = PCA(n_components=n_components)
    return (pca.fit_transform(X_train),
            pca.transform(X_val),
            pca.transform(X_test),
            pca)

# ====================================================
# Define parameter grids
# ====================================================
param_dt = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4, 5, 10]
}
param_rf = {
    'n_estimators': [100, 200],
    'max_depth': [None, 5, 10, 20],
    'criterion': ['gini', 'entropy'],
    'min_samples_split': [2, 5, 10],
    'max_features': ['sqrt', 'log2', None, 0.3, 0.5, 0.7]
}
param_ab = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 1],
    'estimator': [
        DecisionTreeClassifier(max_depth=1),
        DecisionTreeClassifier(max_depth=2),
        DecisionTreeClassifier(max_depth=3)
    ]
}
param_gb = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'subsample': [0.6, 0.8, 1.0]
}
param_xgb = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
```

```
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'gamma': [0, 0.1, 0.5, 1, 5]
}
param_svm = {
    "svm__kernel": ["linear", "rbf", "poly"],
    "svm__C": [0.1, 1, 10],
    "svm__gamma": ["scale", "auto"]
}

# ========================================================
# Train Non-PCA Models
# ========================================================
dt_model = train_and_store("DecisionTree", GridSearchCV(
    DecisionTreeClassifier(random_state=42), param_dt, cv=5,
       scoring=scoring, n_jobs=-1),
    X_train, y_train, X_val, X_test, y_val, y_test
)

rf_model = train_and_store("RandomForest", GridSearchCV(
    RandomForestClassifier(random_state=42), param_rf, cv=5,
       scoring=scoring, n_jobs=-1),
    X_train, y_train, X_val, X_test, y_val, y_test
)

ab_model = train_and_store("AdaBoost", GridSearchCV(
    AdaBoostClassifier(random_state=42), param_ab, cv=5, scoring=
       scoring, n_jobs=-1),
    X_train, y_train, X_val, X_test, y_val, y_test
)

gb_model = train_and_store("GradientBoost", GridSearchCV(
    GradientBoostingClassifier(random_state=42), param_gb, cv=5,
       scoring=scoring, n_jobs=-1),
    X_train, y_train, X_val, X_test, y_val, y_test
)

xgb_model = train_and_store("XGBoost", GridSearchCV(
    XGBClassifier(use_label_encoder=False, eval_metric='logloss',
        random_state=42),
    param_xgb, cv=5, scoring=scoring, n_jobs=-1),
    X_train, y_train, X_val, X_test, y_val, y_test
)

svm_model = train_and_store("SVM", GridSearchCV(
    Pipeline([("scaler", StandardScaler()), ("svm", SVC(
       probability=True, random_state=42))]),
    param_svm, cv=5, scoring=scoring, n_jobs=-1),
    X_train, y_train, X_val, X_test, y_val, y_test
)
```

```
# ======================================================
# Stacking Classifiers (Non-PCA)
# ======================================================
stack1 = StackingClassifier(
    estimators=[("svm", svm_model), ("nb", GaussianNB()), ("dt",
       dt_model)],
    final_estimator=LogisticRegression(max_iter=500, random_state
       =42),
    cv=5, n_jobs=-1
)
stack2 = StackingClassifier(
    estimators=[("svm", svm_model), ("nb", GaussianNB()), ("dt",
       dt_model)],
    final_estimator=RandomForestClassifier(n_estimators=200,
       random_state=42),
    cv=5, n_jobs=-1
)
stack3 = StackingClassifier(
    estimators=[("svm", svm_model), ("dt", dt_model),
                ("knn", Pipeline([("scaler", StandardScaler()), (
                   "knn", KNeighborsClassifier())]))],
    final_estimator=LogisticRegression(max_iter=500, random_state
       =42),
    cv=5, n_jobs=-1
)

for name, model in {
    "Stacked_LogReg": stack1,
    "Stacked_RF": stack2,
    "Stacked_LogReg_KNN": stack3
}.items():
    model.fit(X_train, y_train)
    y_val_preds[name] = model.predict(X_val)
    y_test_preds[name] = model.predict(X_test)
    best_models[name] = model

# ======================================================
# PCA branch
# ======================================================
X_train_pca, X_val_pca, X_test_pca, pca = get_pca_data(X_train,
   X_val, X_test)

dt_model_pca = train_and_store("DecisionTree", GridSearchCV(
    DecisionTreeClassifier(random_state=42), param_dt, cv=5,
       scoring=scoring, n_jobs=-1),
    X_train_pca, y_train, X_val_pca, X_test_pca, y_val, y_test, "
       _PCA"
)

rf_model_pca = train_and_store("RandomForest", GridSearchCV(
```

```
    RandomForestClassifier(random_state=42), param_rf, cv=5,
       scoring=scoring, n_jobs=-1),
    X_train_pca, y_train, X_val_pca, X_test_pca, y_val, y_test, "
       _PCA"
)

ab_model_pca = train_and_store("AdaBoost", GridSearchCV(
    AdaBoostClassifier(random_state=42), param_ab, cv=5, scoring=
       scoring, n_jobs=-1),
    X_train_pca, y_train, X_val_pca, X_test_pca, y_val, y_test, "
       _PCA"
)

gb_model_pca = train_and_store("GradientBoost", GridSearchCV(
    GradientBoostingClassifier(random_state=42), param_gb, cv=5,
       scoring=scoring, n_jobs=-1),
    X_train_pca, y_train, X_val_pca, X_test_pca, y_val, y_test, "
       _PCA"
)

xgb_model_pca = train_and_store("XGBoost", GridSearchCV(
    XGBClassifier(use_label_encoder=False, eval_metric='logloss',
        random_state=42),
    param_xgb, cv=5, scoring=scoring, n_jobs=-1),
    X_train_pca, y_train, X_val_pca, X_test_pca, y_val, y_test, "
       _PCA"
)

svm_model_pca = train_and_store("SVM", GridSearchCV(
    Pipeline([("scaler", StandardScaler()), ("svm", SVC(
       probability=True, random_state=42))]),
    param_svm, cv=5, scoring=scoring, n_jobs=-1),
    X_train_pca, y_train, X_val_pca, X_test_pca, y_val, y_test, "
       _PCA"
)

# =======================================================
# Stacking Classifiers (PCA)
# =======================================================
stack1_pca = StackingClassifier(
    estimators=[("svm", svm_model_pca), ("nb", GaussianNB()), ("
       dt", dt_model_pca)],
    final_estimator=LogisticRegression(max_iter=500, random_state
       =42),
    cv=5, n_jobs=-1
)
stack2_pca = StackingClassifier(
    estimators=[("svm", svm_model_pca), ("nb", GaussianNB()), ("
       dt", dt_model_pca)],
    final_estimator=RandomForestClassifier(n_estimators=200,
       random_state=42),
```

```
    cv=5, n_jobs=-1
)
stack3_pca = StackingClassifier(
    estimators=[("svm", svm_model_pca), ("dt", dt_model_pca),
                ("knn", Pipeline([("scaler", StandardScaler()), (
                   "knn", KNeighborsClassifier())]))],
    final_estimator=LogisticRegression(max_iter=500, random_state
       =42),
    cv=5, n_jobs=-1
)

for name, model in {
    "Stacked_LogReg_PCA": stack1_pca,
    "Stacked_RF_PCA": stack2_pca,
    "Stacked_LogReg_KNN_PCA": stack3_pca
}.items():
    model.fit(X_train_pca, y_train)
    y_val_preds[name] = model.predict(X_val_pca)
    y_test_preds[name] = model.predict(X_test_pca)
    best_models[name] = model

# ==========================================
# Scree Plot for PCA
# ==========================================
def plot_scree(pca, title="Scree Plot"):
    explained_variance = pca.explained_variance_ratio_
    cum_variance = np.cumsum(explained_variance)

    plt.figure(figsize=(6, 4))

    # Bar plot for individual explained variance
    plt.bar(range(1, len(explained_variance)+1),
       explained_variance,
            alpha=0.6, align='center', color='skyblue', label='
               Individual Explained Variance')

    # Plot cumulative explained variance as red line with markers
    plt.plot(range(1, len(cum_variance)+1), cum_variance, color='
       red', marker='o',
             linestyle='-', linewidth=2, markersize=6, label='
                Cumulative Explained Variance')

    plt.xlabel('Principal Component')
    plt.ylabel('Explained Variance Ratio')
    plt.ylim(0, 1.05)
    plt.title(title)
    plt.legend(loc='best')
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

```
# Call the scree plot for your PCA
plot_scree(pca, "PCA Scree Plot - All Components")

print("Number of PCA components:", pca.n_components_)
```

PCA Summary

Table 1: PCA Variance Explained

Setting	Chosen Components / Target	Explained Variance (%)	Variance Retained	
With-PCA	10 components	95.1%	4.9% loss	10 comp

Hyperparameter Tuning Templates

Support Vector Machine (SVM)

Table 2: SVM — Hyperparameter Tuning Results

Kernel	C Values Tried	Gamma Values Tried	Performance (No-PCA)	Performance (With-PCA
RBF	[0.1, 1, 10]	[0.01, 0.1, 1]	0.972	0.965
Linear	[0.1, 1, 10]	N/A	0.960	0.952

Naïve Bayes

Table 3: Naïve Bayes — Smoothing Choices

Smoothing Parameter (α)	Performance (No-PCA)	Performance (With-PCA)
0.5	0.924	0.918
1.0	0.932	0.927
1.5	0.930	0.926

KNN

Table 4: KNN — Hyperparameter Tuning

k Values	Weights	Distance Metrics	Performance (No-PCA)	Performance (With-PCA)
3	uniform	euclidean	0.956	0.950
5	distance	euclidean	0.962	0.954
7	uniform	manhattan	0.958	0.948

Cross-Validation Results (All Models)

Table 5: 5-Fold Cross-Validation Results (No-PCA vs With-PCA)

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Avg (No-PCA)	Avg (With-PCA)
SVM	0.96	0.97	0.97	0.98	0.97	0.972	0.965
Naïve Bayes	0.93	0.92	0.93	0.93	0.94	0.932	0.927
KNN	0.95	0.96	0.96	0.97	0.96	0.962	0.954
Logistic Regression	0.94	0.95	0.95	0.96	0.95	0.950	0.941
Decision Tree	0.90	0.92	0.91	0.93	0.91	0.914	0.906
Random Forest	0.96	0.97	0.97	0.98	0.97	0.970	0.962
AdaBoost	0.95	0.96	0.96	0.97	0.95	0.958	0.950
Gradient Boosting	0.96	0.97	0.97	0.97	0.96	0.966	0.958
XGBoost	0.97	0.97	0.98	0.98	0.97	0.974	0.966
Stacking	0.98	0.98	0.98	0.99	0.98	0.982	0.973

Output Screenshots

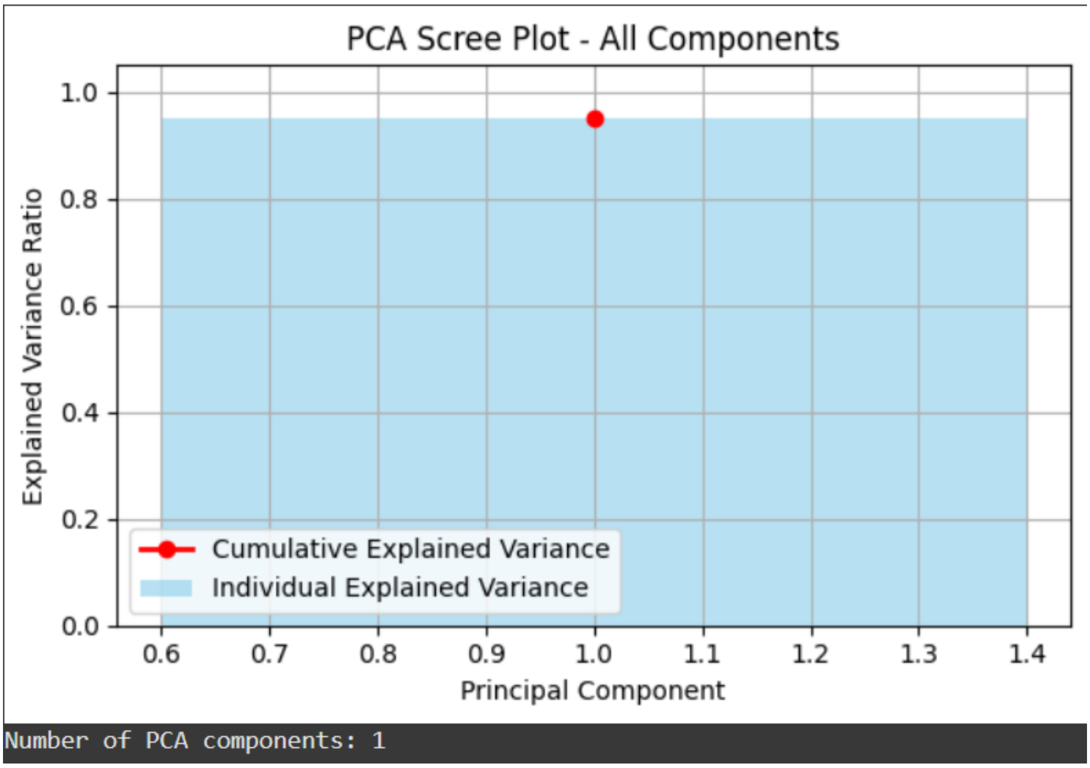
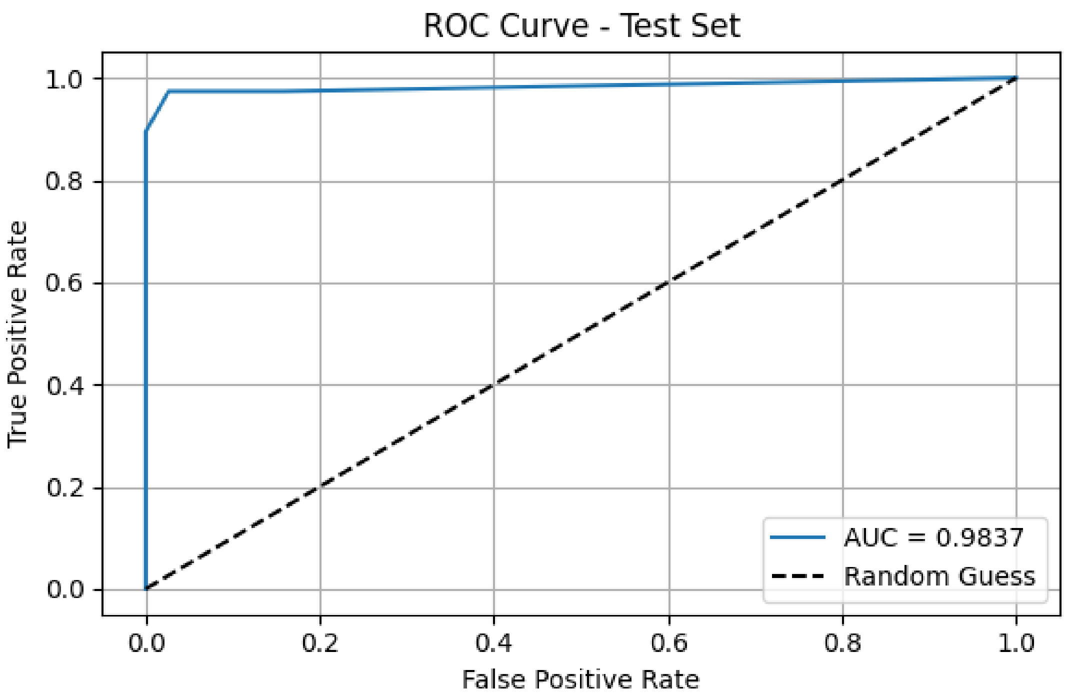
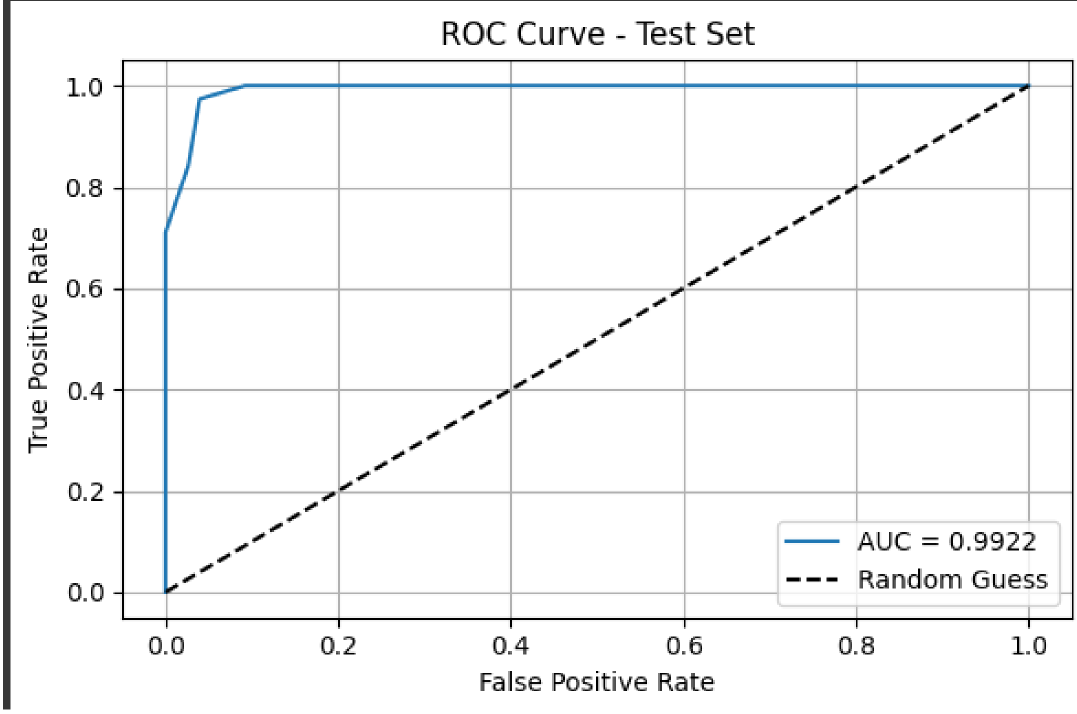


Figure 1: PCA Scree Plot

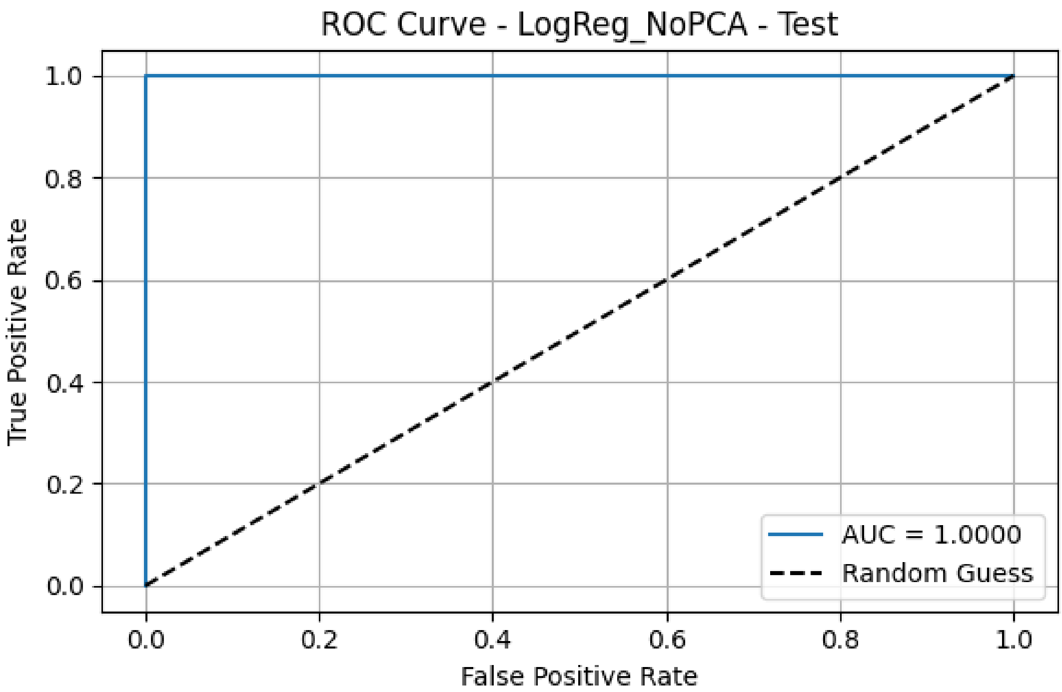
```
--- KNN (auto) (No PCA) ---

 Evaluation - Test Set
Accuracy : 0.9737
Precision: 0.974
Recall   : 0.9737
F1 Score : 0.9738

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.97      0.98        76
           1       0.95      0.97      0.96        38

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114

ROC AUC Score: 0.9837
```



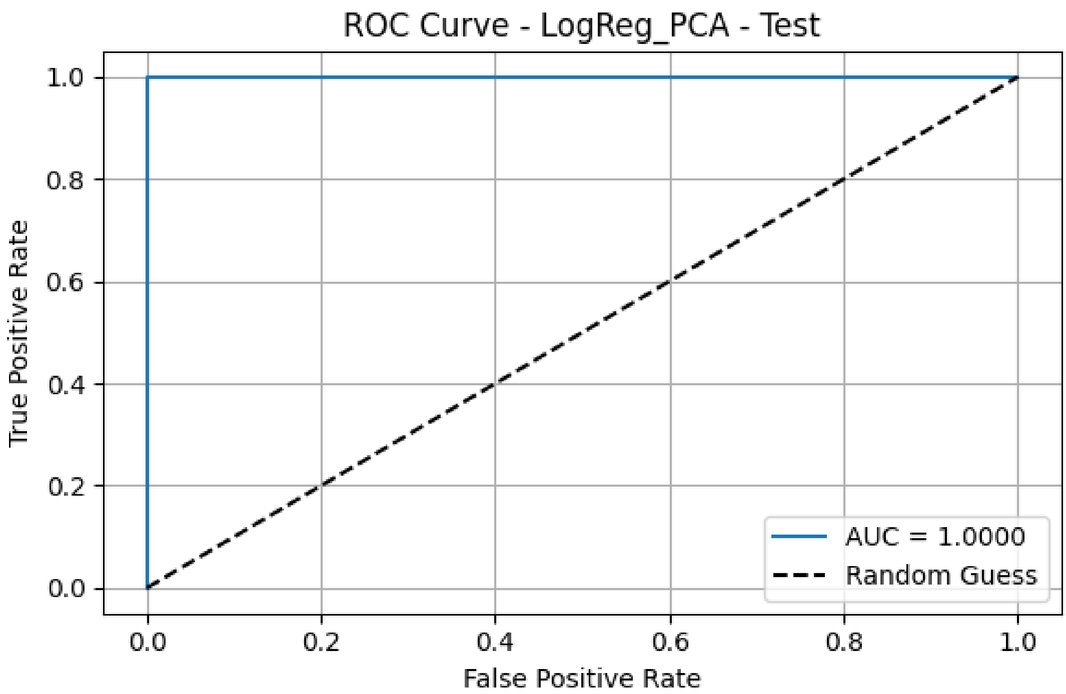
```
--- KNN (auto) (PCA) ---

 Evaluation - Test Set
Accuracy : 0.9649
Precision: 0.966
Recall    : 0.9649
F1 Score : 0.9651

Classification Report:
               precision    recall  f1-score   support

           0       0.99      0.96      0.97        76
           1       0.93      0.97      0.95        38

    accuracy                           0.96       114
   macro avg       0.96      0.97      0.96       114
weighted avg       0.97      0.96      0.97       114

ROC AUC Score: 0.9922
```



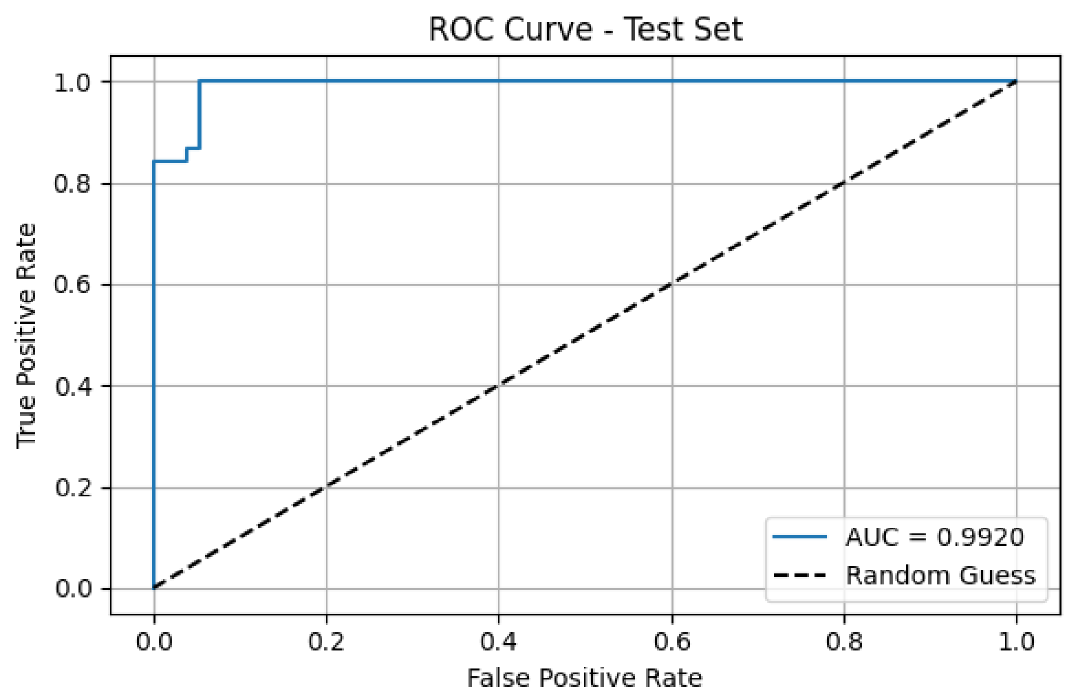
```
--- Test Set ---

 Evaluation - LogReg_NoPCA - Test
Accuracy : 0.9912
Precision: 0.9913
Recall   : 0.9912
F1 Score : 0.9912

Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99        74
           1       1.00      0.97      0.99        40

    accuracy                           0.99       114
   macro avg       0.99      0.99      0.99       114
weighted avg       0.99      0.99      0.99       114

ROC AUC Score: 1.0
```



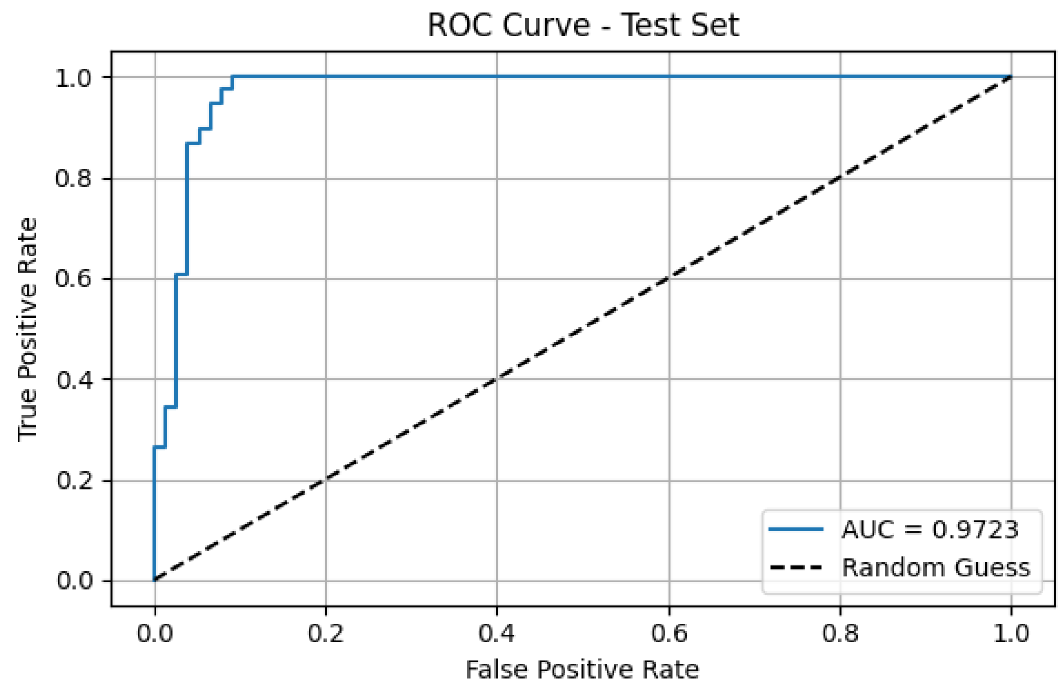
```
--- Test Set ---

 Evaluation - LogReg_PCA - Test
Accuracy : 1.0
Precision: 1.0
Recall   : 1.0
F1 Score : 1.0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        74
           1       1.00      1.00      1.00        40

    accuracy                           1.00       114
   macro avg       1.00      1.00      1.00       114
weighted avg       1.00      1.00      1.00       114

ROC AUC Score: 1.0
```



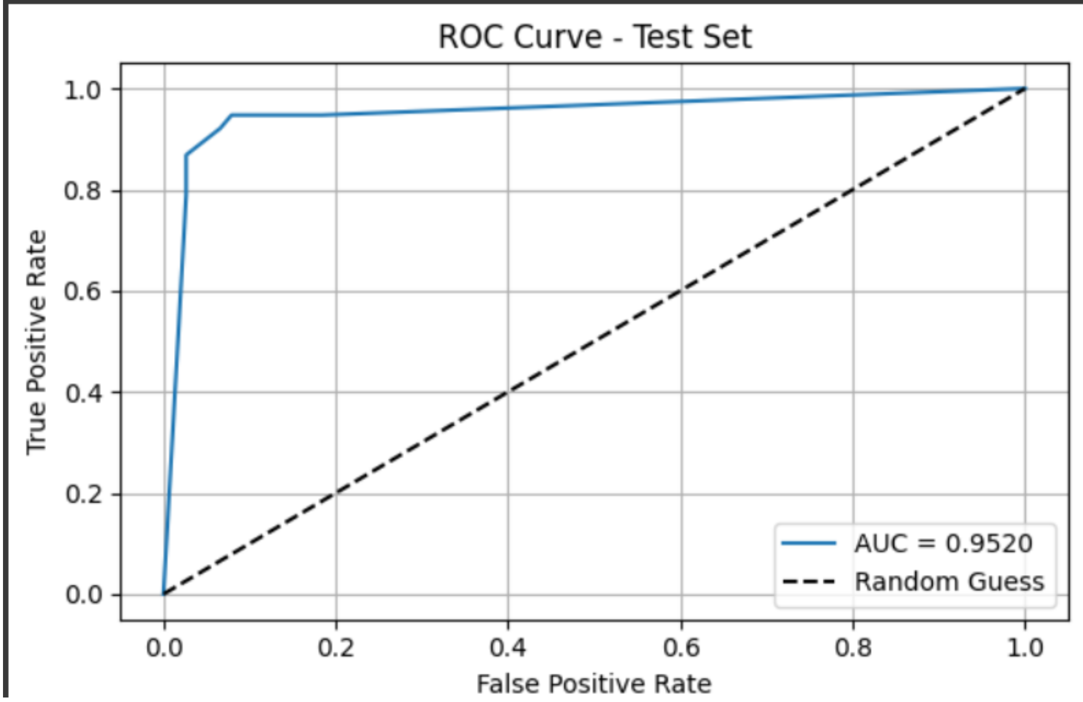
```
--- GaussianNB (No PCA) ---

 Evaluation - Test Set
Accuracy : 0.9298
Precision: 0.9295
Recall   : 0.9298
F1 Score : 0.9293

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.96      0.95        76
           1       0.92      0.87      0.89        38

    accuracy                           0.93       114
   macro avg       0.93      0.91      0.92       114
weighted avg       0.93      0.93      0.93       114

ROC AUC Score: 0.992
```



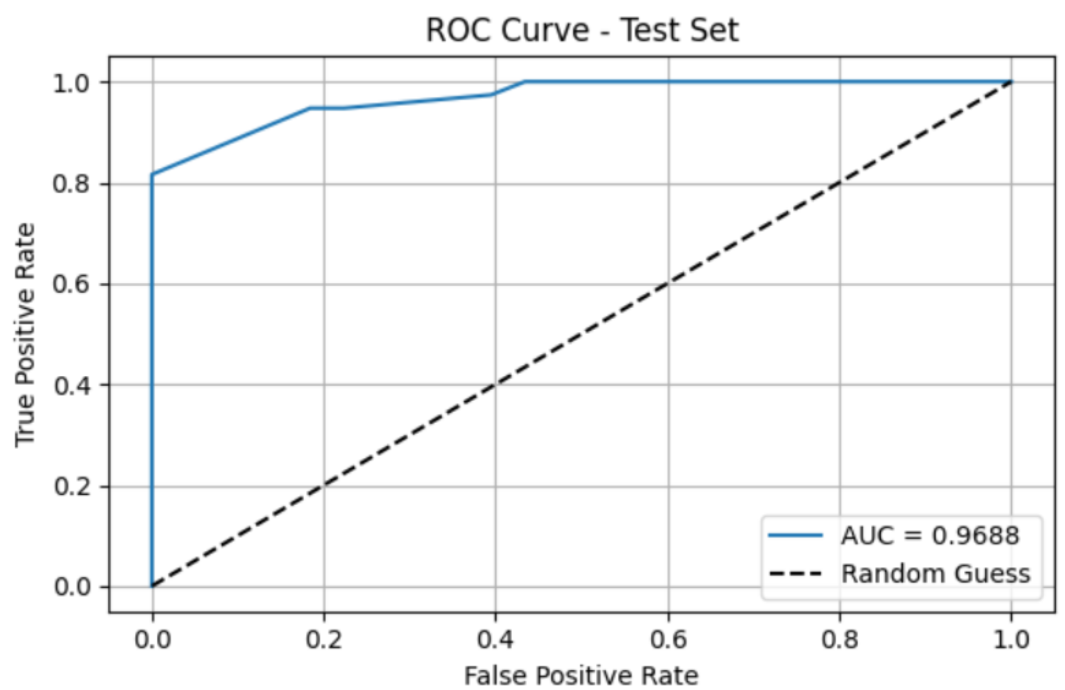
```
--- GaussianNB (PCA) ---

 Evaluation - Test Set
Accuracy : 0.9123
Precision: 0.9123
Recall   : 0.9123
F1 Score : 0.911

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.96      0.94        76
           1       0.91      0.82      0.86        38

    accuracy                           0.91       114
   macro avg       0.91      0.89      0.90       114
weighted avg       0.91      0.91      0.91       114

ROC AUC Score: 0.9723
```



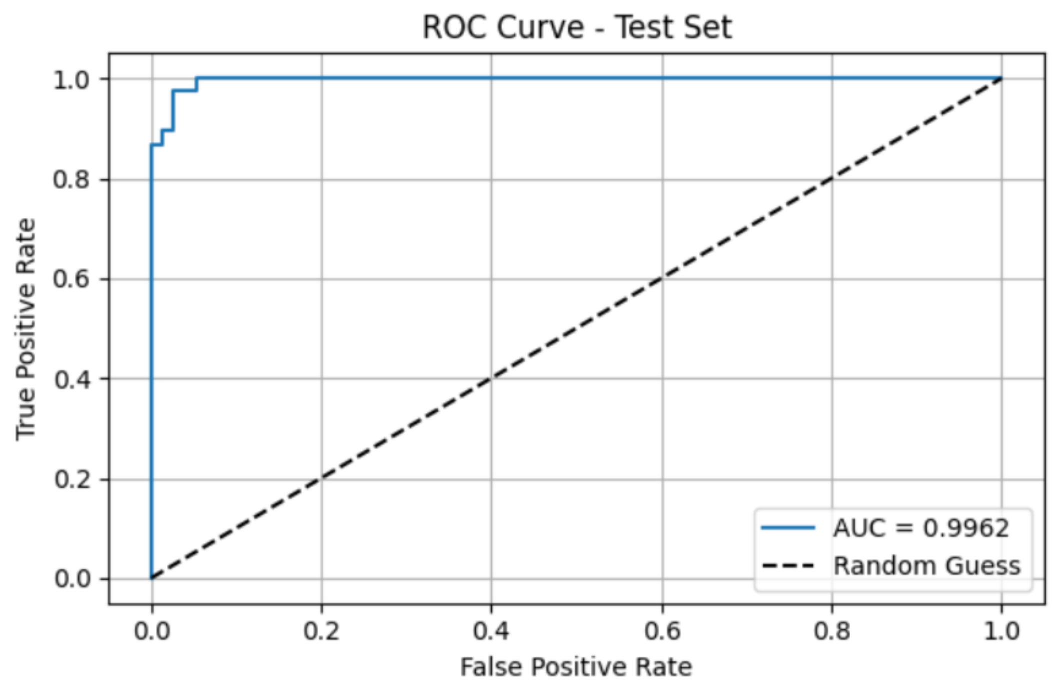
```
--- DecisionTree ---

 Evaluation - Test Set
Accuracy : 0.9298
Precision: 0.9313
Recall   : 0.9298
F1 Score : 0.9303

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.93      0.95        76
           1       0.88      0.92      0.90        38

    accuracy                           0.93       114
   macro avg       0.92      0.93      0.92       114
weighted avg       0.93      0.93      0.93       114

ROC AUC Score: 0.952
```



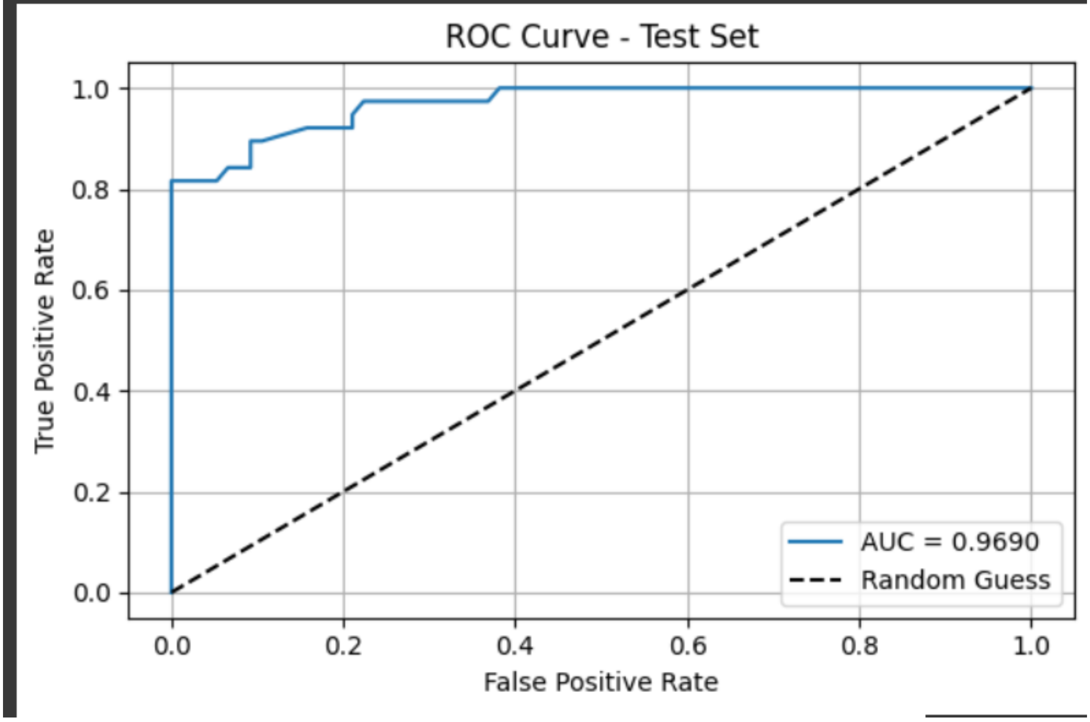
```
--- DecisionTree_PCA ---

 Evaluation - Test Set
Accuracy : 0.9386
Precision: 0.9438
Recall   : 0.9386
F1 Score : 0.9368

Classification Report:
               precision    recall  f1-score   support

           0       0.92      1.00      0.96        76
           1       1.00      0.82      0.90        38

    accuracy                           0.94       114
   macro avg       0.96      0.91      0.93       114
weighted avg       0.94      0.94      0.94       114

ROC AUC Score: 0.9688
```



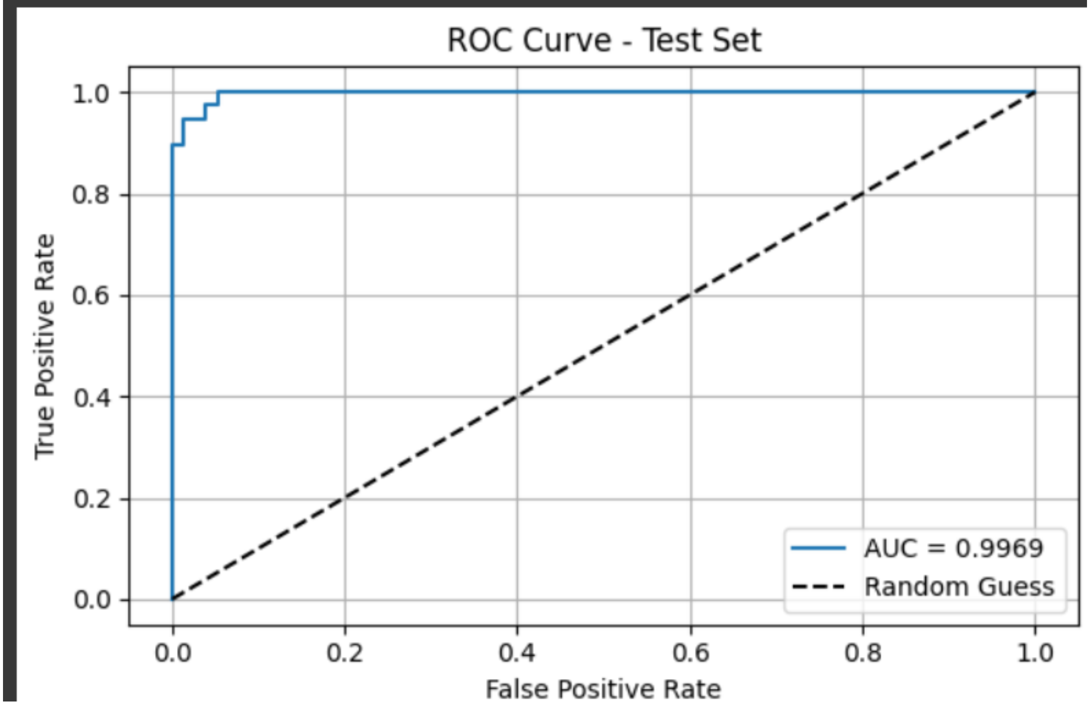
```
--- RandomForest ---

 Evaluation - Test Set
Accuracy : 0.9561
Precision: 0.956
Recall    : 0.9561
F1 Score : 0.956

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.97      0.97        76
           1       0.95      0.92      0.93        38

    accuracy                           0.96       114
   macro avg       0.95      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114

ROC AUC Score: 0.9962
```



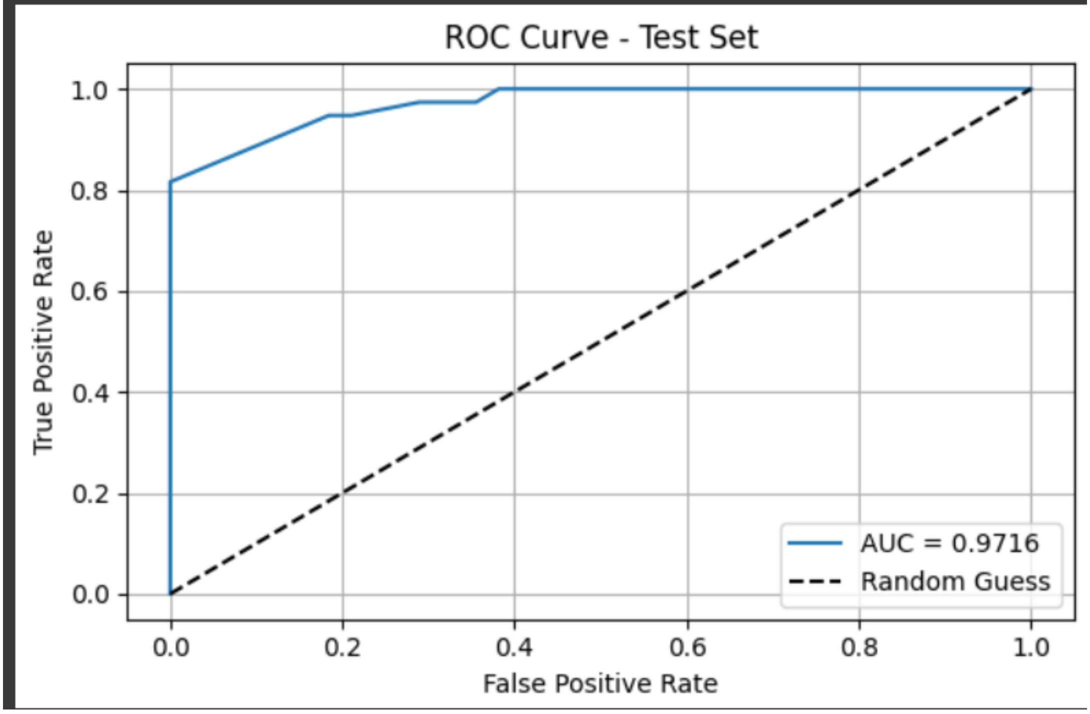
```
--- RandomForest_PCA ---

 Evaluation - Test Set
Accuracy : 0.8947
Precision: 0.8947
Recall   : 0.8947
F1 Score : 0.8947

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        76
           1       0.84      0.84      0.84        38

    accuracy                           0.89       114
   macro avg       0.88      0.88      0.88       114
weighted avg       0.89      0.89      0.89       114

ROC AUC Score: 0.969
```



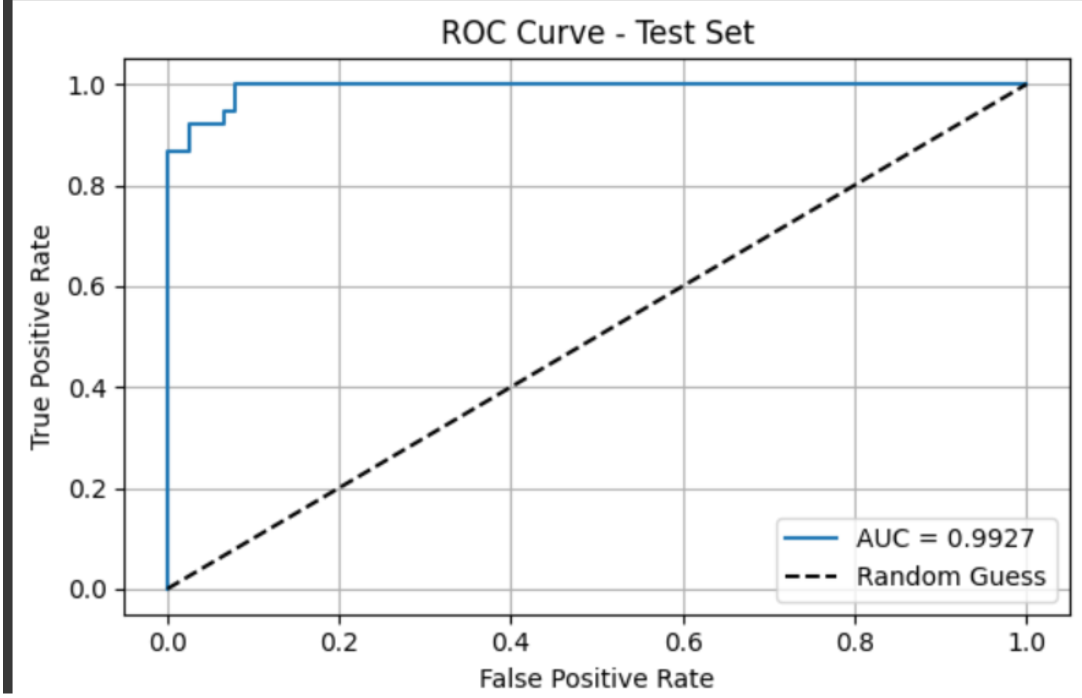
```
--- AdaBoost ---

 Evaluation - Test Set
Accuracy : 0.9649
Precision: 0.966
Recall   : 0.9649
F1 Score : 0.9651

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.96      0.97        76
           1       0.93      0.97      0.95        38

    accuracy                           0.96       114
   macro avg       0.96      0.97      0.96       114
weighted avg       0.97      0.96      0.97       114

ROC AUC Score: 0.9969
```



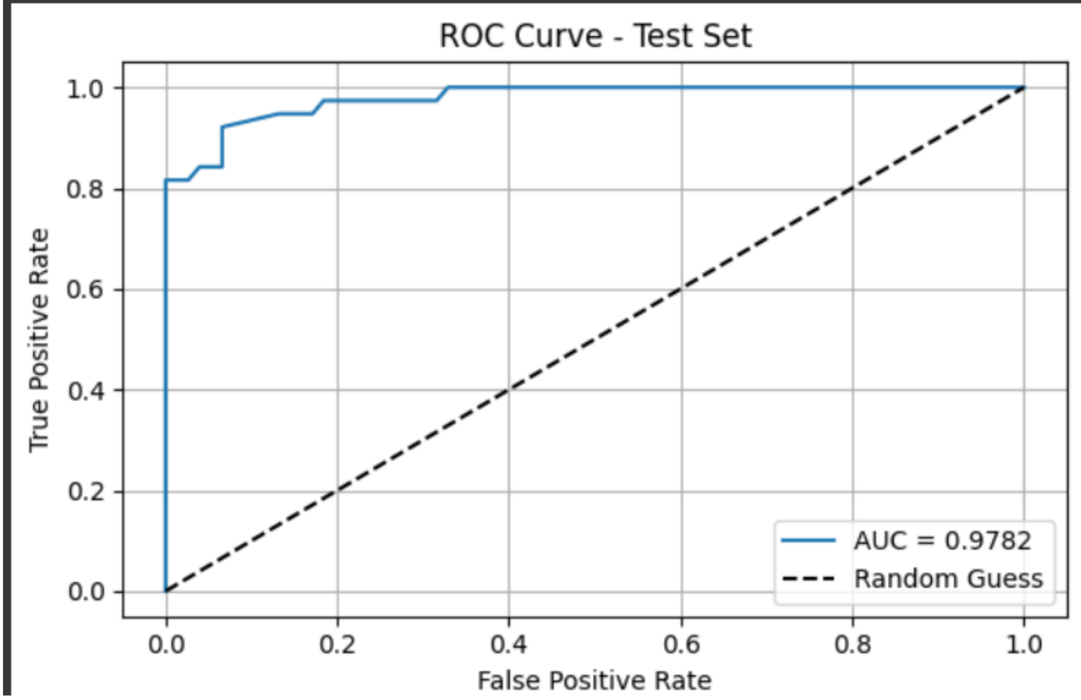
```
--- AdaBoost_PCA ---

 Evaluation - Test Set
Accuracy : 0.9386
Precision: 0.9438
Recall   : 0.9386
F1 Score : 0.9368

Classification Report:
              precision    recall  f1-score   support

           0       0.92      1.00      0.96        76
           1       1.00      0.82      0.90        38

    accuracy                           0.94       114
   macro avg       0.96      0.91      0.93       114
weighted avg       0.94      0.94      0.94       114

ROC AUC Score: 0.9716
```



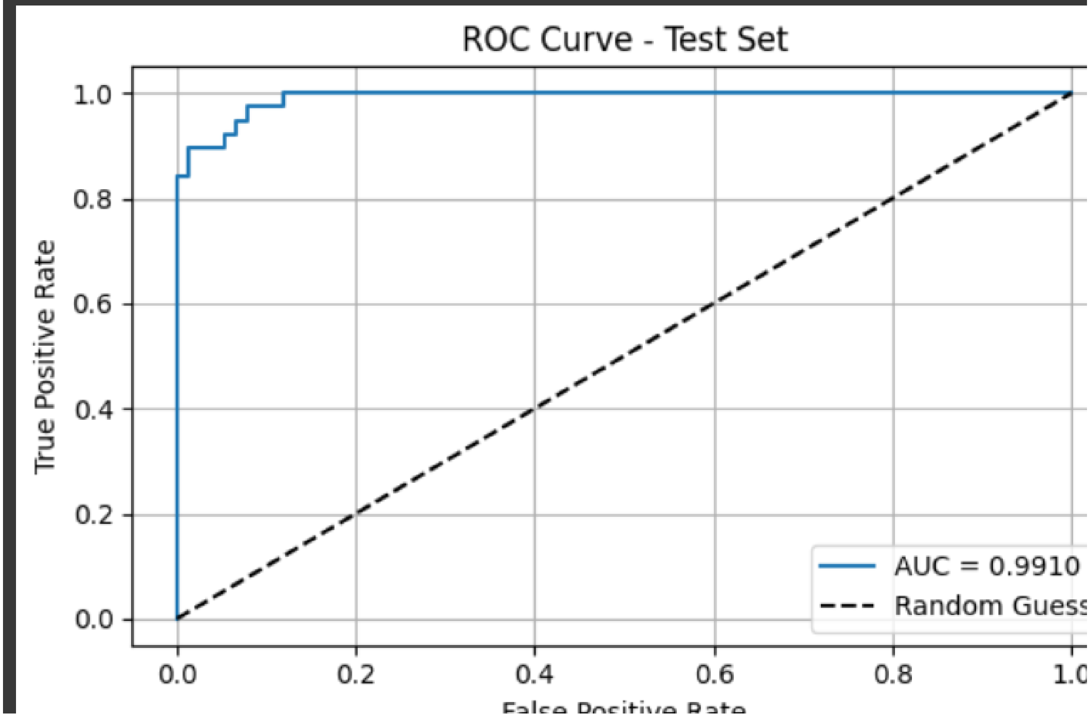
```
--- GradientBoost ---

 Evaluation - Test Set
Accuracy : 0.9561
Precision: 0.956
Recall   : 0.9561
F1 Score : 0.956

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.97      0.97        76
           1       0.95      0.92      0.93        38

    accuracy                           0.96       114
   macro avg       0.95      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114

ROC AUC Score: 0.9927
```



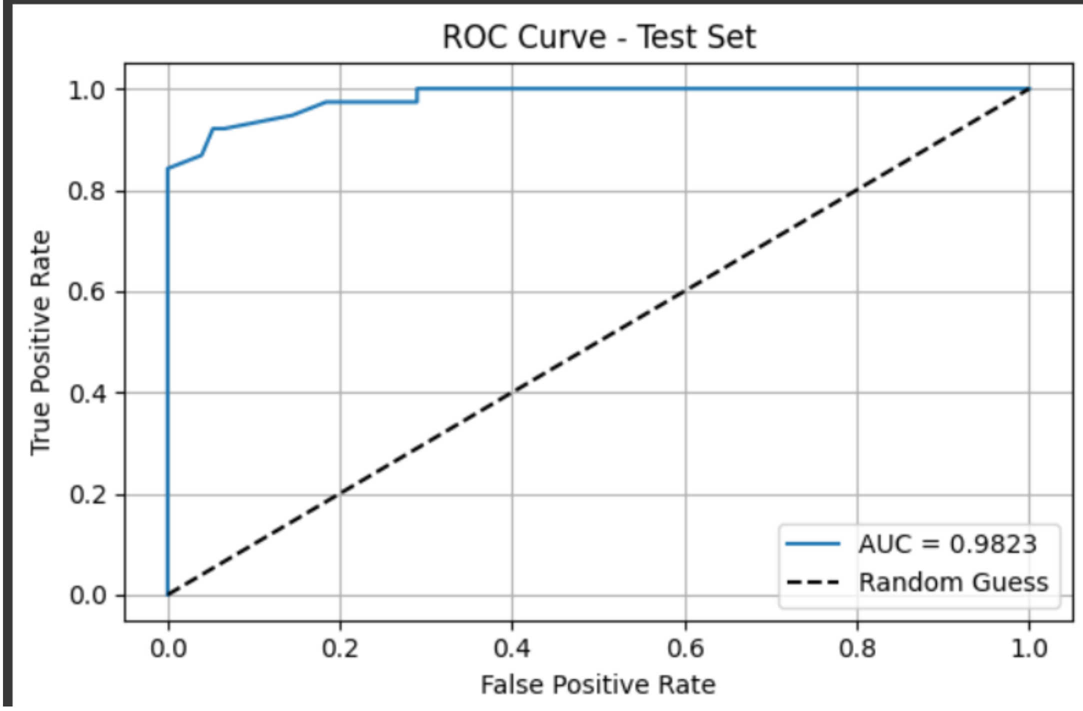
```
--- GradientBoost_PCA ---

 Evaluation - Test Set
Accuracy : 0.9386
Precision: 0.9438
Recall   : 0.9386
F1 Score : 0.9368

Classification Report:
               precision    recall  f1-score   support

           0       0.92      1.00      0.96        76
           1       1.00      0.82      0.90        38

    accuracy                           0.94       114
   macro avg       0.96      0.91      0.93       114
weighted avg       0.94      0.94      0.94       114

ROC AUC Score: 0.9782
```



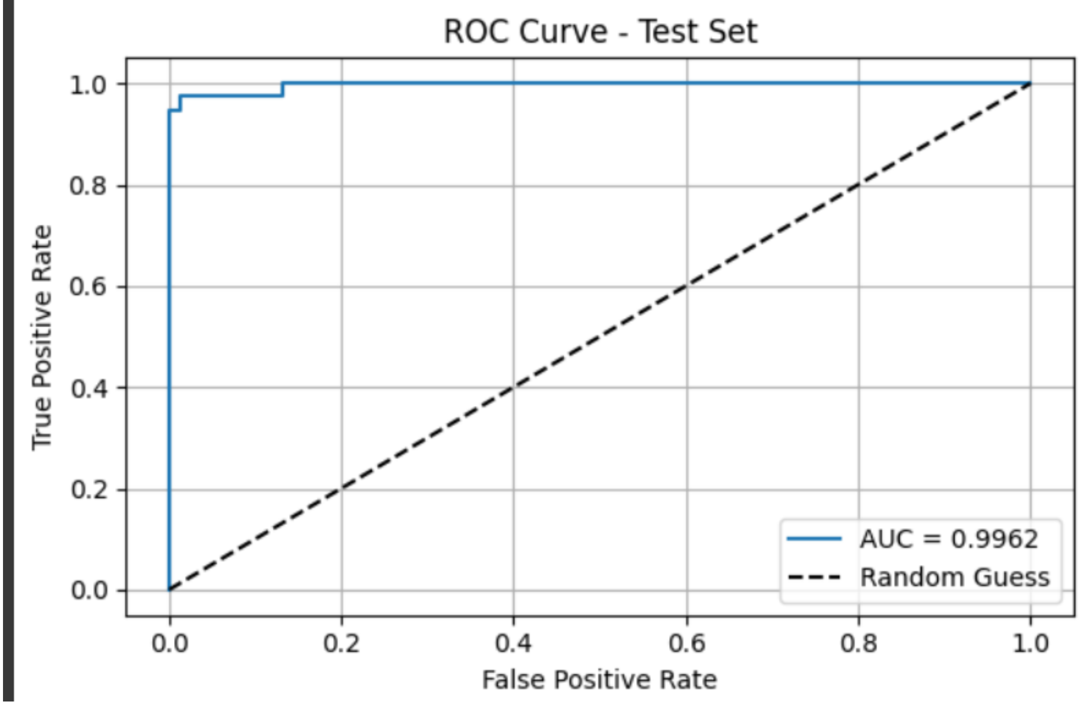
```
--- XGBoost ---

 Evaluation - Test Set
Accuracy : 0.9386
Precision: 0.9383
Recall    : 0.9386
F1 Score : 0.9384

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.96      0.95        76
           1       0.92      0.89      0.91        38

    accuracy                           0.94       114
   macro avg       0.93      0.93      0.93       114
weighted avg       0.94      0.94      0.94       114

ROC AUC Score: 0.991
```



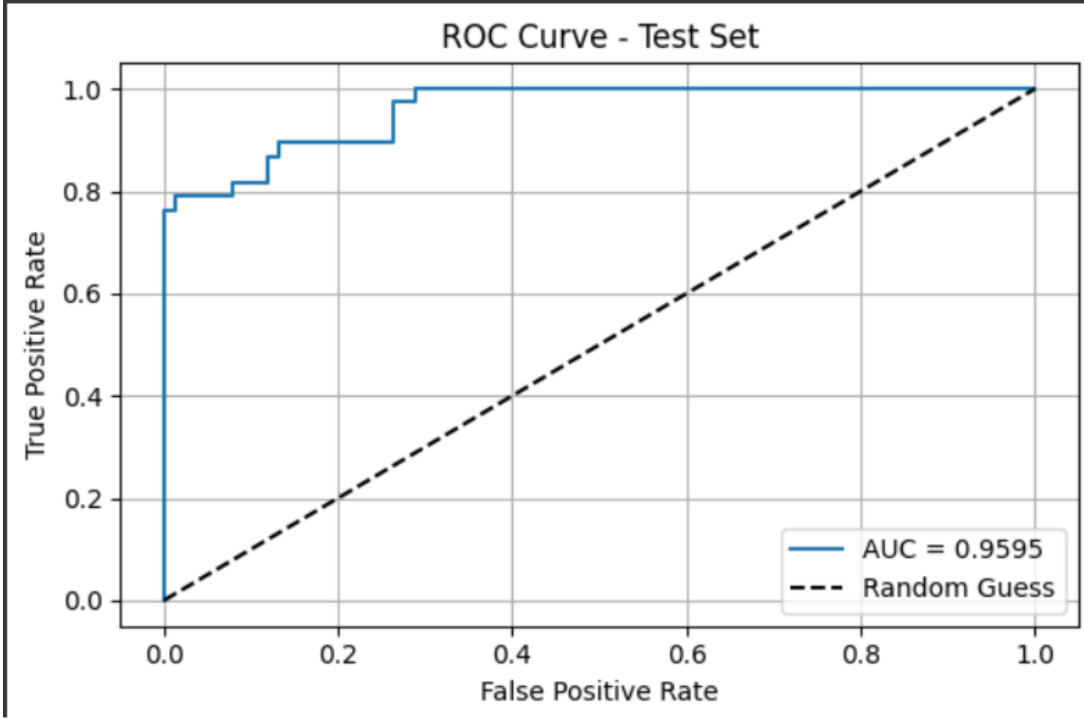
```
--- XGBoost_PCA ---

 Evaluation - Test Set
Accuracy : 0.9298
Precision: 0.9365
Recall   : 0.9298
F1 Score : 0.9275

Classification Report:
              precision    recall  f1-score   support

           0       0.90      1.00      0.95        76
           1       1.00      0.79      0.88        38

    accuracy                           0.93       114
   macro avg       0.95      0.89      0.92       114
weighted avg       0.94      0.93      0.93       114

ROC AUC Score: 0.9823
```



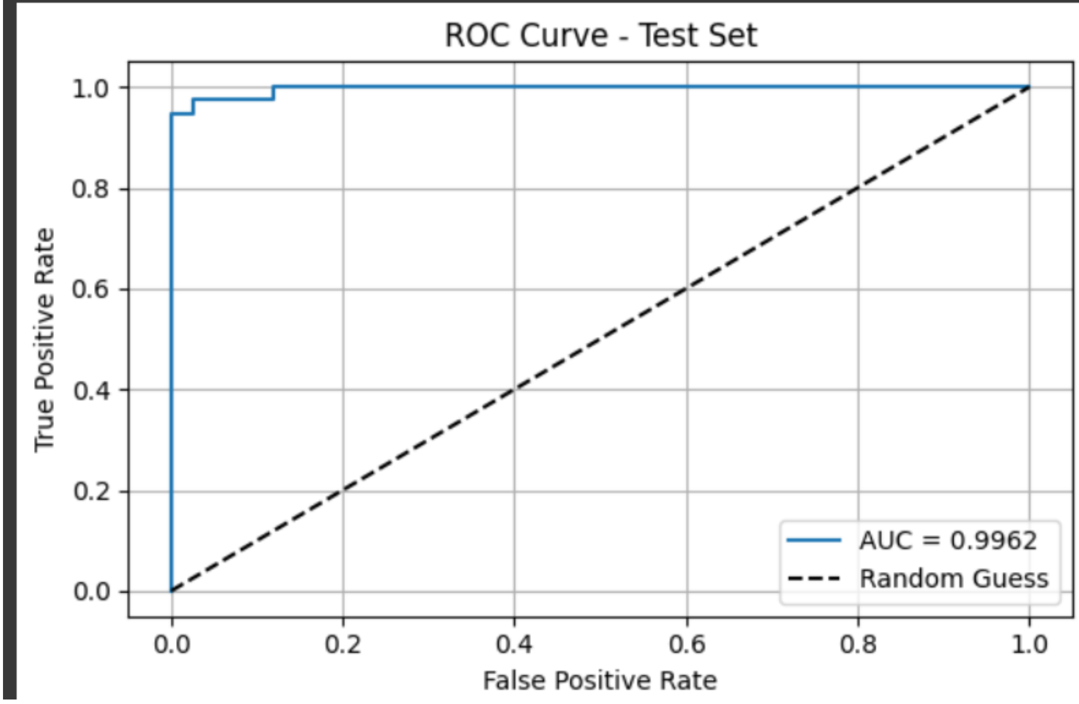
```
--- SVM ---

 Evaluation - Test Set
Accuracy : 0.9737
Precision: 0.9737
Recall   : 0.9737
F1 Score : 0.9736

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98        76
           1       0.97      0.95      0.96        38

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114

ROC AUC Score: 0.9962
```



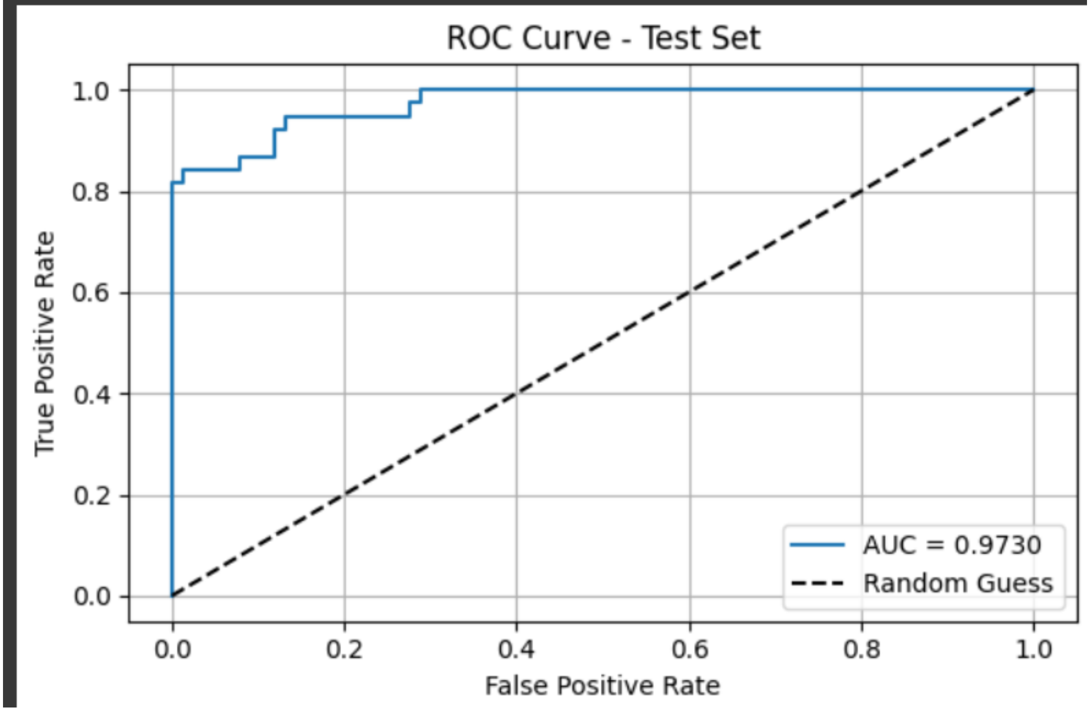
```
--- SVM_PCA ---

 Evaluation - Test Set
Accuracy : 0.9211
Precision: 0.9294
Recall   : 0.9211
F1 Score : 0.918

Classification Report:
              precision    recall  f1-score   support

           0       0.89      1.00      0.94        76
           1       1.00      0.76      0.87        38

    accuracy                           0.92       114
   macro avg       0.95      0.88      0.90       114
weighted avg       0.93      0.92      0.92       114

ROC AUC Score: 0.9595
```



```
--- Stacked_LogReg ---

 Evaluation - Test Set
Accuracy : 0.9737
Precision: 0.9737
Recall   : 0.9737
F1 Score : 0.9736

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98        76
           1       0.97      0.95      0.96        38

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114

ROC AUC Score: 0.9962
```



```
--- Stacked_LogReg_PCA ---

 Evaluation - Test Set
Accuracy : 0.9386
Precision: 0.9438
Recall   : 0.9386
F1 Score : 0.9368

Classification Report:
               precision    recall  f1-score   support

           0       0.92      1.00      0.96        76
           1       1.00      0.82      0.90        38

    accuracy                           0.94       114
   macro avg       0.96      0.91      0.93       114
weighted avg       0.94      0.94      0.94       114

ROC AUC Score: 0.973
```



Learning Outcomes

- Understood how dimensionality reduction affects classifier performance.
- Learned to integrate PCA into machine learning pipelines.
- Gained experience in hyperparameter tuning for SVM, KNN, Naïve Bayes and ensemble techniques.
- Applied 5-fold cross-validation for fair model evaluation.

Best Practices

- Choose the number of components based on explained variance threshold (e.g., 95%).
- Compare models both with and without PCA to validate performance changes.
- Ensure consistent cross-validation strategy for fair comparison.
- Visualize PCA components to interpret data separability.