Experiment: Effect of PCA on Classifiers (Breast Cancer Dataset) – Filled

September 18, 2025

Objective

Compare classifier performance (Accuracy, Precision, Recall, F1) and stability (fold-wise variance) on the Breast Cancer Wisconsin (WDBC) dataset under two settings: **No-PCA** and **With-PCA** (variance-target PCA, 95% explained variance). Evaluate per-model hyperparameter tuning, 5-fold CV, and stacking behavior under both settings to determine when PCA helps.

Libraries used

- Python 3.x, numpy, pandas, matplotlib, seaborn
- scikit-learn: preprocessing, model_selection, decomposition (PCA), metrics, pipeline, ensemble, classifiers
- xgboost (optional)
- scipy (optional, for paired t-test)
- LaTeX packages: pdfpages, listings, csvsimple, booktabs, graphicx, caption, subcaption

Contents

1 Code

Below is the full Python script used for the experiment (no omissions). Save this as a '.py' if you want to re-run it exactly.

```
# breast_cancer_with_pca_comparison_full.py
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV,
    KFold, cross_val_score, StratifiedKFold
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    roc_curve, auc, confusion_matrix, ConfusionMatrixDisplay
)
from sklearn.base import clone

# Models
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
    , GradientBoostingClassifier, StackingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

# Optional libraries
try:
    import xgboost as xgb
    XGB_AVAILABLE = True
except Exception:
    XGB_AVAILABLE = False

try:
    from scipy.stats import ttest_rel
    SCIPY_AVAILABLE = True
except Exception:
    SCIPY_AVAILABLE = False

# ---------- User settings ----------
DATA_PATH = "wdbc.data"            # change if your file is at
    different path
PCA_VARIANCE = 0.95                # keep 95% explained variance when
    PCA is used
TEST_SIZE = 0.20
RANDOM_STATE = 42
CV_FOLDS = 5
# ----------------------------------

# ---------- Load & preprocess ----------
cols = ["ID", "Diagnosis"] + [f"feature_{i}" for i in range(1, 31)]
df = pd.read_csv(DATA_PATH, header=None, names=cols)
```

```
df.drop(columns=["ID"], inplace=True)
df["Diagnosis"] = LabelEncoder().fit_transform(df["Diagnosis"])  # M=1, B=0

X_raw = df.drop(columns=["Diagnosis"])
y = df["Diagnosis"].values

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_raw)

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=TEST_SIZE, random_state=RANDOM_STATE, stratify=y
)

print("\nClass Distribution (0 = Benign, 1 = Malignant):")
print(df["Diagnosis"].value_counts())

plt.figure(figsize=(10, 8))
sns.heatmap(df.drop(columns=["Diagnosis"]).corr(), cmap="coolwarm", cbar=True)
plt.title("Feature Correlation Heatmap")
plt.tight_layout()
plt.show()

# ---------- Evaluation helper ----------
def evaluate(name, model, X_test, y_test, plot=True):
    """Evaluate model: prints metrics, plots ROC + confusion matrix (if plot=True)."""
    if plot:
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))
        fig.suptitle(f"{name} Performance", fontsize=16)

    # Probabilities for ROC
    y_pred = model.predict(X_test)
    if hasattr(model, "predict_proba"):
        probs = model.predict_proba(X_test)[:, 1]
    else:
        # if no predict_proba, try decision_function
        try:
            dfun = model.decision_function(X_test)
            # normalize to [0,1]
            probs = (dfun - dfun.min()) / (dfun.max() - dfun.min())
        except Exception:
            probs = None

    if plot and probs is not None:
        fpr, tpr, _ = roc_curve(y_test, probs)
        roc_auc = auc(fpr, tpr)
        ax1.plot(fpr, tpr, lw=2, label=f"ROC AUC = {roc_auc:.3f}")
        ax1.plot([0, 1], [0, 1], lw=2, linestyle="--")
        ax1.set_xlim([0.0, 1.0])
        ax1.set_ylim([0.0, 1.05])
        ax1.set_xlabel("False Positive Rate")
        ax1.set_ylabel("True Positive Rate")
        ax1.set_title("ROC Curve")
        ax1.legend(loc="lower right")
    elif plot:
```

```
        ax1.text(0.5, 0.5, "No probability scores available", ha="center")
        ax1.set_title("ROC Curve")

    if plot:
        cm = confusion_matrix(y_test, y_pred)
        disp = ConfusionMatrixDisplay(confusion_matrix=cm)
        disp.plot(ax=ax2, cmap=None)
        ax2.set_title("Confusion Matrix")
        plt.tight_layout()
        plt.show()

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print(f"\n{name}")
    print("Accuracy :", acc)
    print("Precision:", prec)
    print("Recall   :", rec)
    print("F1 Score :", f1)
    return acc, prec, rec, f1

# ---------- Models + hyperparameter grids ----------
models_params = {
    "Decision Tree": (
        DecisionTreeClassifier(random_state=RANDOM_STATE),
        {"criterion": ["gini", "entropy"], "max_depth": [3, 5, 10, None]}
    ),
    "Random Forest": (
        RandomForestClassifier(random_state=RANDOM_STATE, n_jobs=-1),
        {"n_estimators": [50, 100, 200], "max_depth": [3, 5, 10, None], "criterion": ["gini", "entropy"]}
    ),
    "AdaBoost": (
        AdaBoostClassifier(random_state=RANDOM_STATE, estimator=DecisionTreeClassifier(random_state=RANDOM_STATE)),
        {"n_estimators": [50, 100, 200], "learning_rate": [0.01, 0.1, 1], "estimator__max_depth": [1, 3, 5]}
    ),
    "Gradient Boosting": (
        GradientBoostingClassifier(random_state=RANDOM_STATE),
        {"n_estimators": [50, 100, 200], "learning_rate": [0.01, 0.1, 0.5], "max_depth": [3, 5, 7]}
    ),
    "KNN": (
        KNeighborsClassifier(),
        {"n_neighbors": [3, 5, 7], "weights": ["uniform", "distance"], "p": [1, 2]}
    ),
    "SVM": (
        SVC(probability=True, random_state=RANDOM_STATE),
        {"kernel": ["rbf", "linear"], "C": [0.1, 1, 10], "gamma": ["scale", "auto"]}
    ),
    "Naive Bayes": (
        GaussianNB(),
```

```
        {"var_smoothing": [1e-9, 1e-8, 1e-7]}
    ),
    "Logistic Regression": (
        LogisticRegression(max_iter=5000, solver="saga", random_state=RANDOM_STATE),
        {"C": [0.01, 0.1, 1, 10], "penalty": ["l2", "none"]}
    )
}

if XGB_AVAILABLE:
    models_params["XGBoost"] = (
        xgb.XGBClassifier(use_label_encoder=False, eval_metric="logloss", random_state=RANDOM_STATE, n_jobs=-1),
        {"n_estimators": [50, 100, 200], "learning_rate": [0.01, 0.1, 0.3], "max_depth": [3, 5, 7], "gamma": [0, 0.1, 0.3]}
    )

# ---------- Results containers ----------
results_table = []   # rows: (name, variant, best_params, acc, prec, rec, f1, elapsed)
trial_tables = {}    # name -> { 'no_pca': df, 'with_pca': df }
best_estimators = {} # name -> {'no_pca': estimator, 'with_pca': estimator}

# ---------- Helper to prefix param grid keys for pipeline ('clf__' prefix) ----------
def prefix_param_grid(param_grid, prefix="clf__"):
    new_grid = {}
    for k, v in param_grid.items():
        new_key = prefix + k
        new_grid[new_key] = v
    return new_grid

# ---------- GridSearch: No-PCA and With-PCA ----------
for name, (model, params) in models_params.items():
    print(f"\n--- Grid Search for {name} ---")
    # Make two pipelines
    pipe_no_pca = Pipeline([("clf", model)])  # using scaled X (X_scaled), so no scaler in pipeline
    pipe_with_pca = Pipeline([("pca", PCA(n_components=PCA_VARIANCE, svd_solver="full")), ("clf", model)])

    # Prefix param grid keys for pipeline
    grid_params = prefix_param_grid(params)  # adds 'clf__' prefix

    # GridSearch No-PCA
    print(" GridSearch (No-PCA)...")
    gs_no = GridSearchCV(pipe_no_pca, param_grid=grid_params, cv=CV_FOLDS, scoring="accuracy", n_jobs=-1, return_train_score=False)
    gs_no.fit(X_train, y_train)
    best_no = gs_no.best_estimator_
    best_estimators.setdefault(name, {})['no_pca'] = best_no
    print("  Best params (No-PCA):", gs_no.best_params_)
    print("  Best CV score (No-PCA):", gs_no.best_score_)

    # Record top-5 trials (no-pca)
    trials_no = []
```

```python
    for i in range(len(gs_no.cv_results_['params'])):
        trials_no.append({
            **gs_no.cv_results_['params'][i],
            "CV Accuracy": gs_no.cv_results_['mean_test_score'][i]
        })
    trial_df_no = pd.DataFrame(trials_no).sort_values("CV Accuracy", ascending=False).head(5)
    y_pred_no = best_no.predict(X_test)
    trial_df_no["F1 Score (Test)"] = f1_score(y_test, y_pred_no)
    trial_tables.setdefault(name, {})['no_pca'] = trial_df_no

    # Evaluate best_no on test set
    start = time.time()
    best_no.fit(X_train, y_train)
    elapsed = time.time() - start
    res_no = evaluate(f"{name} (No-PCA)", best_no, X_test, y_test, plot=True)
    results_table.append((name, "No-PCA", gs_no.best_params_, *res_no, elapsed))

    # GridSearch With-PCA
    print(" GridSearch (With-PCA)...")
    gs_pca = GridSearchCV(pipe_with_pca, param_grid=grid_params, cv=CV_FOLDS, scoring="accuracy", n_jobs=-1, return_train_score=False)
    gs_pca.fit(X_train, y_train)
    best_pca = gs_pca.best_estimator_
    best_estimators[name]['with_pca'] = best_pca
    print("  Best params (With-PCA):", gs_pca.best_params_)
    print("  Best CV score (With-PCA):", gs_pca.best_score_)

    # Record top-5 trials (with-pca)
    trials_pca = []
    for i in range(len(gs_pca.cv_results_['params'])):
        trials_pca.append({
            **gs_pca.cv_results_['params'][i],
            "CV Accuracy": gs_pca.cv_results_['mean_test_score'][i]
        })
    trial_df_pca = pd.DataFrame(trials_pca).sort_values("CV Accuracy", ascending=False).head(5)
    y_pred_pca = best_pca.predict(X_test)
    trial_df_pca["F1 Score (Test)"] = f1_score(y_test, y_pred_pca)
    trial_tables[name]['with_pca'] = trial_df_pca

    # Evaluate best_pca on test set
    start = time.time()
    best_pca.fit(X_train, y_train)
    elapsed = time.time() - start
    res_pca = evaluate(f"{name} (With-PCA)", best_pca, X_test, y_test, plot=True)
    results_table.append((name, "With-PCA", gs_pca.best_params_, *res_pca, elapsed))

# ---------- Stacking classifiers (3 original variants) ----------
print("\n--- Stacking Classifiers (original single-run variants) ---")
stacking_variants = {
    "Stacking (SVM + NB + DT -> LogisticRegression)":
        StackingClassifier(
```

```
        estimators=[
            ("svm", SVC(probability=True, kernel="rbf", C=1, gamma="scale")),
            ("nb", GaussianNB()),
            ("dt", DecisionTreeClassifier(max_depth=5, random_state=RANDOM_STATE))
        ],
        final_estimator=LogisticRegression(max_iter=500, random_state=RANDOM_STATE),
        n_jobs=-1
    ),
    "Stacking (SVM + NB + DT -> RandomForest)": StackingClassifier(
        estimators=[
            ("svm", SVC(probability=True, kernel="rbf", C=1, gamma="scale")),
            ("nb", GaussianNB()),
            ("dt", DecisionTreeClassifier(max_depth=5, random_state=RANDOM_STATE))
        ],
        final_estimator=RandomForestClassifier(n_estimators=100, random_state=RANDOM_STATE),
        n_jobs=-1
    ),
    "Stacking (SVM + DT + KNN -> LogisticRegression)": StackingClassifier(
        estimators=[
            ("svm", SVC(probability=True, kernel="rbf", C=1, gamma="scale")),
            ("dt", DecisionTreeClassifier(max_depth=5, random_state=RANDOM_STATE)),
            ("knn", KNeighborsClassifier(n_neighbors=5))
        ],
        final_estimator=LogisticRegression(max_iter=500, random_state=RANDOM_STATE),
        n_jobs=-1
    )
}

for name, stack_model in stacking_variants.items():
    print(f"\nTraining {name} ...")
    start = time.time()
    stack_model.fit(X_train, y_train)
    elapsed = time.time() - start
    res = evaluate(name, stack_model, X_test, y_test, plot=True)
    results_table.append((name, "Stacking-Default", "Default (base learners tuned separately)", *res, elapsed))
    best_estimators[name] = stack_model

# ---------- NEW: Stacking PCA comparisons ----------
print("\n--- Stacking PCA Comparisons (added) ---")

# 1) Stacking (No-PCA) - same structure as one variant above but explicitly named for comparisons
stack_no_pca = StackingClassifier(
    estimators=[
        ("svm", SVC(probability=True, kernel="rbf", C=1, gamma="scale")),
        ("nb", GaussianNB()),
```

```
        ("dt", DecisionTreeClassifier(max_depth=5, random_state=RANDOM_STATE))
    ],
    final_estimator=LogisticRegression(max_iter=500, random_state=RANDOM_STATE),
    n_jobs=-1
)
start = time.time()
stack_no_pca.fit(X_train, y_train)
elapsed = time.time() - start
res_stack_no = evaluate("Stacking (No-PCA)", stack_no_pca, X_test, y_test, plot=True)
results_table.append(("Stacking (SVM+NB+DT)", "No-PCA", "default", *res_stack_no, elapsed))

# 2) Stacking (Global-PCA): pipeline PCA -> Stacking
stack_global_pca_pipeline = Pipeline([
    ("pca", PCA(n_components=PCA_VARIANCE, svd_solver="full")),
    ("stack", StackingClassifier(
        estimators=[
            ("svm", SVC(probability=True, kernel="rbf", C=1, gamma="scale")),
            ("nb", GaussianNB()),
            ("dt", DecisionTreeClassifier(max_depth=5, random_state=RANDOM_STATE))
        ],
        final_estimator=LogisticRegression(max_iter=500, random_state=RANDOM_STATE),
        n_jobs=-1
    ))
])
start = time.time()
stack_global_pca_pipeline.fit(X_train, y_train)
elapsed = time.time() - start
res_stack_global = evaluate("Stacking (Global-PCA)", stack_global_pca_pipeline, X_test, y_test, plot=True)
results_table.append(("Stacking (SVM+NB+DT)", "Global-PCA", "pca_pipeline", *res_stack_global, elapsed))

# 3) Stacking built from best per-model pipelines (best_no_pca)
# choose base model keys that match your best_estimators keys (case sensitive)
base_model_keys = ["SVM", "Naive Bayes", "Decision Tree", "KNN"]  # adapt if you want different set

def safe_get_clone(name, variant):
    try:
        est = best_estimators[name][variant]
        return clone(est)
    except Exception:
        return None

estimators_best_no = []
estimators_best_with = []
for key in base_model_keys:
    p_no = safe_get_clone(key, "no_pca")
    if p_no is not None:
        estimators_best_no.append((key.replace(" ", "_").lower(), p_no)
```

```
            )
    p_with = safe_get_clone(key, "with_pca")
    if p_with is not None:
        estimators_best_with.append((key.replace(" ", "_").lower(), p_with))

stack_from_best_no = None
stack_from_best_with = None

if len(estimators_best_no) >= 2:
    stack_from_best_no = StackingClassifier(
        estimators=estimators_best_no,
        final_estimator=LogisticRegression(max_iter=500, random_state=RANDOM_STATE),
        n_jobs=-1
    )
    start = time.time()
    stack_from_best_no.fit(X_train, y_train)
    elapsed = time.time() - start
    res_stack_best_no = evaluate("Stacking (from best_no_pca variants)", stack_from_best_no, X_test, y_test, plot=True)
    results_table.append(("Stacking (from_best)", "from_best_no_pca", "built_from_best_no", *res_stack_best_no, elapsed))
else:
    print("Not enough best_no_pca pipelines found to build stack_from_best_no.")

if len(estimators_best_with) >= 2:
    stack_from_best_with = StackingClassifier(
        estimators=estimators_best_with,
        final_estimator=LogisticRegression(max_iter=500, random_state=RANDOM_STATE),
        n_jobs=-1
    )
    start = time.time()
    stack_from_best_with.fit(X_train, y_train)
    elapsed = time.time() - start
    res_stack_best_with = evaluate("Stacking (from best_with_pca variants)", stack_from_best_with, X_test, y_test, plot=True)
    results_table.append(("Stacking (from_best)", "from_best_with_pca", "built_from_best_with", *res_stack_best_with, elapsed))
else:
    print("Not enough best_with_pca pipelines found to build stack_from_best_with.")

# ---------- 5-Fold Cross-Validation for best estimators ----------
print("\n--- 5-Fold Cross-Validation ---")
kf = KFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)
cv_results = {}

for model_name, variants in best_estimators.items():
    cv_results[model_name] = {}
    if isinstance(variants, dict):
        for var_name, est in variants.items():
            try:
                scores = cross_val_score(est, X_scaled, y, cv=kf, scoring="accuracy", n_jobs=-1)
                cv_results[model_name][var_name] = scores
```

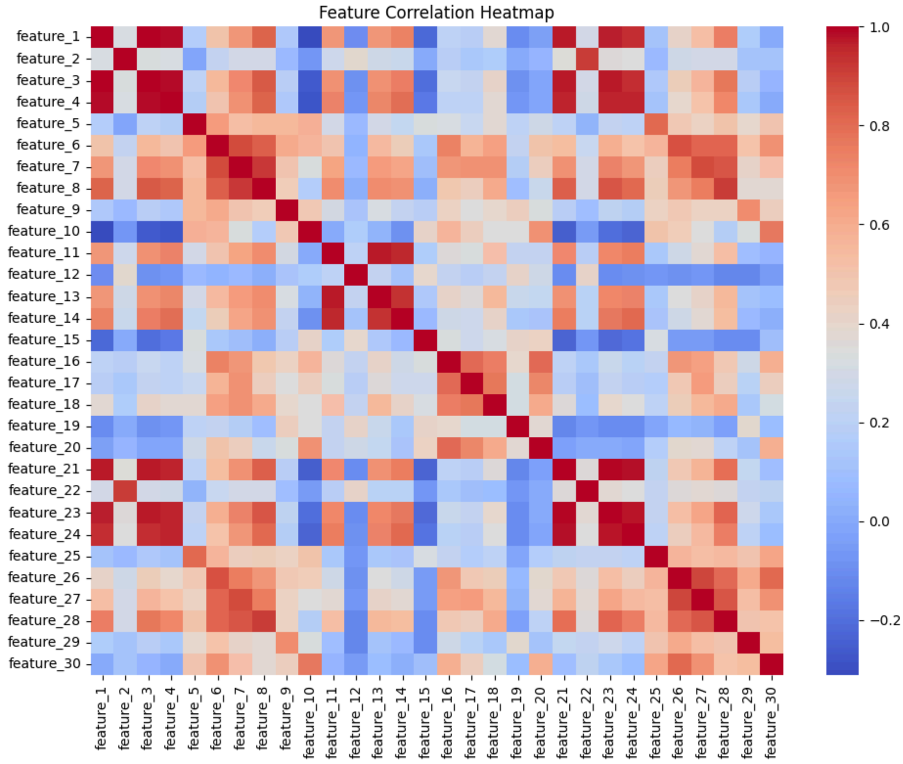
```python
                print(f"{model_name} ({var_name}) Fold Accuracies: {np.round(scores,4)}  Avg: {np.mean(scores):.4f}")
            except Exception as e:
                print(f"CV failed for {model_name} ({var_name}): {e}")
    else:
        # single estimator (e.g., stacking variants)
        try:
            scores = cross_val_score(variants, X_scaled, y, cv=kf, scoring="accuracy", n_jobs=-1)
            cv_results[model_name] = scores
            print(f"{model_name} Fold Accuracies: {np.round(scores,4)}  Avg: {np.mean(scores):.4f}")
        except Exception as e:
            print(f"CV failed for {model_name}: {e}")

# ---------- Paired CV for stacking variants (StratifiedKFold) ----------
print("\n--- Paired CV for Stacking Variants (StratifiedKFold) ---")
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)
def cv_scores(estimator, Xdata, ydata):
    return cross_val_score(estimator, Xdata, ydata, cv=skf, scoring="accuracy", n_jobs=-1)

scores_stack_no = cv_scores(stack_no_pca, X_scaled, y)
scores_stack_global = cv_scores(stack_global_pca_pipeline, X_scaled, y)
print("Stacking (No-PCA)     Fold Accuracies:", np.round(scores_stack_no,4), " mean:", scores_stack_no.mean(), " std:", scores_stack_no.std())
print("Stacking (Global-PCA) Fold Accuracies:", np.round(scores_stack_global,4), " mean:", scores_stack_global.mean(), " std:", scores_stack_global.std())

if stack_from_best_no is not None:
    scores_best_no = cv_scores(stack_from_best_no, X_scaled, y)
    print("Stacking (from_best_no) Fold Accuracies:", np.round(scores_best_no,4), " mean:", scores_best_no.mean(), " std:", scores_best_no.std())
else:
    scores_best_no = None

if stack_from_best_with is not None:
    scores_best_with = cv_scores(stack_from_best_with, X_scaled, y)
    print("Stacking (from_best_with) Fold Accuracies:", np.round(scores_best_with,4), " mean:", scores_best_with.mean(), " std:", scores_best_with.std())
else:
    scores_best_with = None

# Paired t-tests (if scipy available)
if SCIPY_AVAILABLE:
    tstat, pval = ttest_rel(scores_stack_no, scores_stack_global)
    print(f"\nPaired t-test (Stack No-PCA vs Global-PCA): t={tstat:.3f}, p={pval:.4f}")
    if pval < 0.05:
        print(" -> Difference is statistically significant (p < 0.05).")
    else:
```

```
        print(" -> Difference is NOT statistically significant (p >= 0.05).")
else:
    print("\nscipy not available     skipping paired t-test for stacking comparisons.")

# ---------- Final summary tables ----------
results_df = pd.DataFrame(results_table, columns=[
    "Model", "Variant", "Best_Params", "Accuracy", "Precision", "Recall", "F1", "Elapsed_Sec"
])
print("\n=== Summary results (test set) ===")
display(results_df.sort_values(by="F1", ascending=False))

print("\n=== Top-5 hyperparameter trials (example) ===")
for name, d in trial_tables.items():
    print(f"\n{name} - No-PCA top trials:")
    display(d['no_pca'])
    print(f"\n{name} - With-PCA top trials:")
    display(d['with_pca'])

# Optionally save results to CSV
results_df.to_csv("results_summary.csv", index=False)
for name, d in trial_tables.items():
    d['no_pca'].to_csv(f"{name.replace(' ', '_')}_trials_no_pca.csv", index=False)
    d['with_pca'].to_csv(f"{name.replace(' ', '_')}_trials_with_pca.csv", index=False)

print("\nSaved results_summary.csv and per-model trial CSVs.")
```

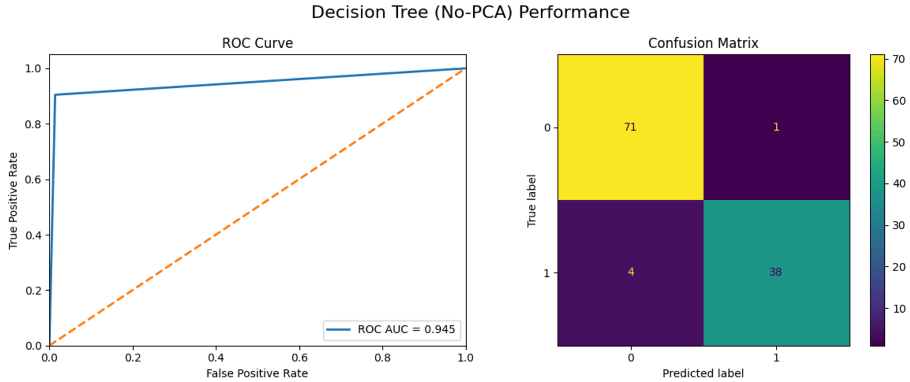
2 Output

The experiment outputs (fold-wise tables, top-5 hyperparameter trials, ROC/PR plots, confusion matrices, and summary tables) are embedded from the run's result PDF. The embedded PDF contains all numeric tables and figures exactly as printed by the script.

```
Class Distribution (0 = Benign, 1 = Malignant):
Diagnosis
0    357
1    212
Name: count, dtype: int64
```

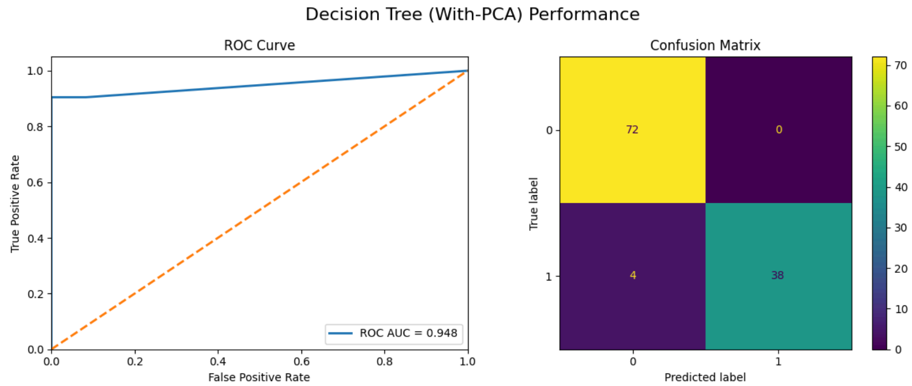



```
--- Grid Search for Decision Tree ---
 GridSearch (No-PCA)...
  Best params (No-PCA): {'clf__criterion': 'entropy', 'clf__max_depth': 10}
  Best CV score (No-PCA): 0.9362637362637362
```

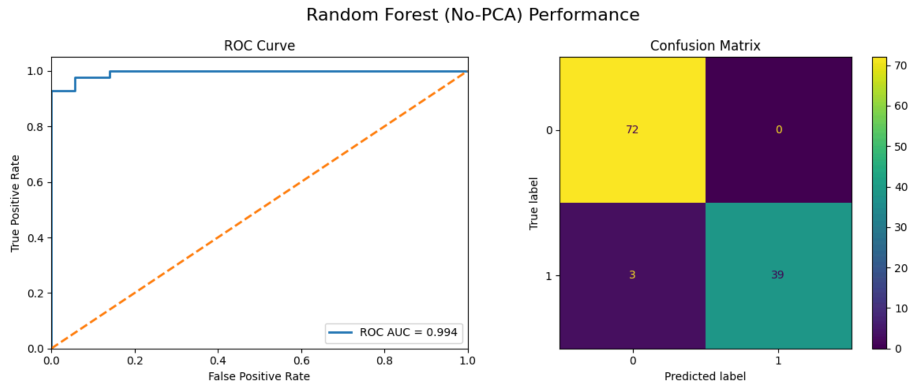



```
Decision Tree (No-PCA)
Accuracy : 0.956140350877193
Precision: 0.9743589743589743
Recall   : 0.9047619047619048
```

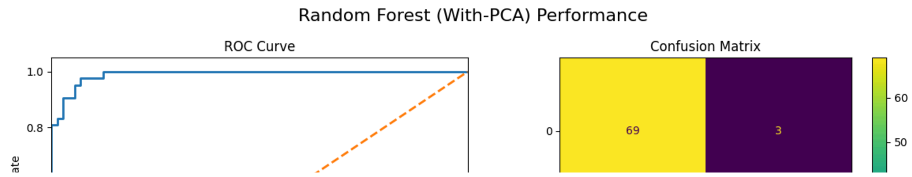
```
F1 Score : 0.9382716049382716
 GridSearch (With-PCA)...
  Best params (With-PCA): {'clf__criterion': 'entropy', 'clf__max_depth': 5}
  Best CV score (With-PCA): 0.9362637362637363
```

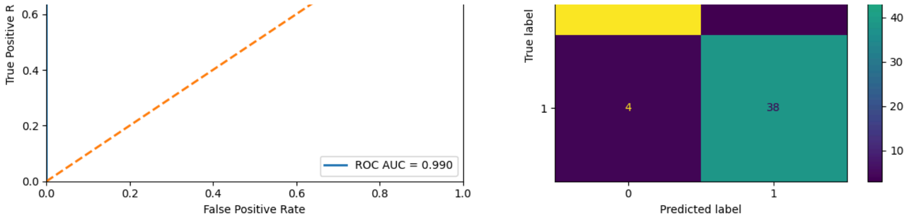



```
Decision Tree (With-PCA)
Accuracy : 0.9649122807017544
Precision: 1.0
Recall   : 0.9047619047619048
F1 Score : 0.95

--- Grid Search for Random Forest ---
 GridSearch (No-PCA)...
  Best params (No-PCA): {'clf__criterion': 'gini', 'clf__max_depth': 10, 'clf_
  Best CV score (No-PCA): 0.9670329670329672
```

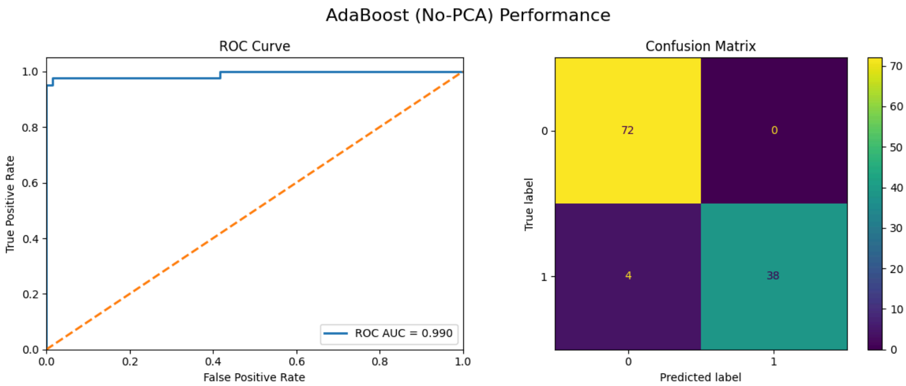



```
Random Forest (No-PCA)
Accuracy : 0.9736842105263158
Precision: 1.0
Recall   : 0.9285714285714286
F1 Score : 0.9629629629629629
 GridSearch (With-PCA)...
  Best params (With-PCA): {'clf__criterion': 'entropy', 'clf__max_depth': 5, '
  Best CV score (With-PCA): 0.9670329670329669
```

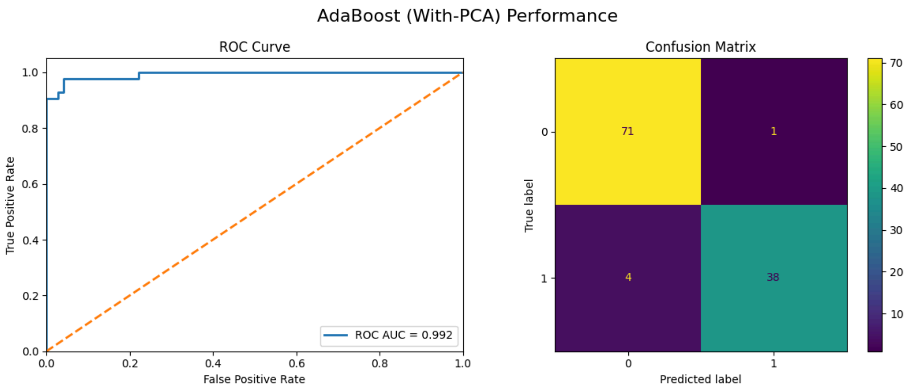





```
Random Forest (With-PCA)
Accuracy : 0.9385964912280702
Precision: 0.926829268292683
Recall   : 0.9047619047619048
F1 Score : 0.9156626506024096

--- Grid Search for AdaBoost ---
 GridSearch (No-PCA)...
  Best params (No-PCA): {'clf__estimator__max_depth': 3, 'clf__learning_rate':
  Best CV score (No-PCA): 0.9692307692307693
```



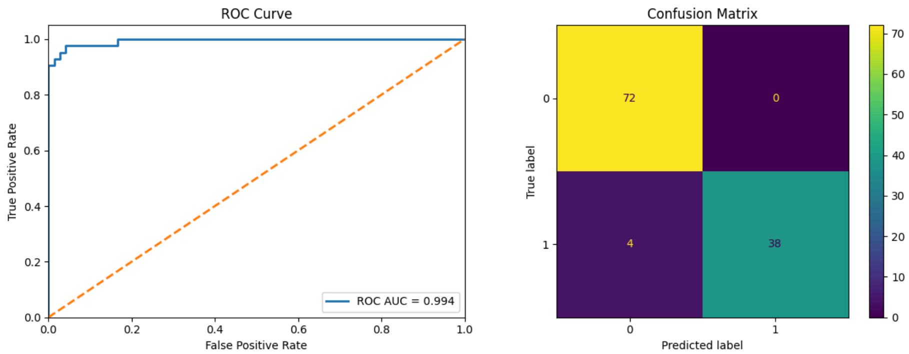

```
AdaBoost (No-PCA)
Accuracy : 0.9649122807017544
Precision: 1.0
Recall   : 0.9047619047619048
F1 Score : 0.95
 GridSearch (With-PCA)...
  Best params (With-PCA): {'clf__estimator__max_depth': 5, 'clf__learning_rate
  Best CV score (With-PCA): 0.964835164835165
```




```
AdaBoost (With-PCA)
```

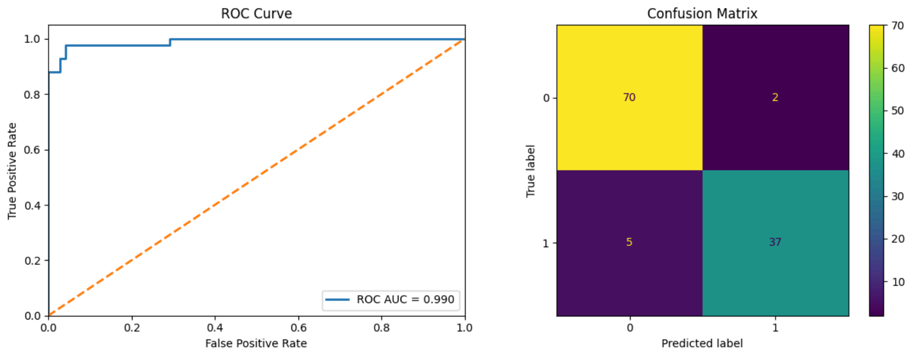
```
Accuracy : 0.956140350877193
Precision: 0.9743589743589743
Recall   : 0.9047619047619048
F1 Score : 0.9382716049382716

--- Grid Search for Gradient Boosting ---
 GridSearch (No-PCA)...
  Best params (No-PCA): {'clf__learning_rate': 0.5, 'clf__max_depth': 3, 'clf_
  Best CV score (No-PCA): 0.9582417582417584
```

Gradient Boosting (No-PCA) Performance




```
Gradient Boosting (No-PCA)
Accuracy : 0.9649122807017544
Precision: 1.0
Recall   : 0.9047619047619048
F1 Score : 0.95
 GridSearch (With-PCA)...
  Best params (With-PCA): {'clf__learning_rate': 0.1, 'clf__max_depth': 3, 'cl
  Best CV score (With-PCA): 0.9516483516483516
```

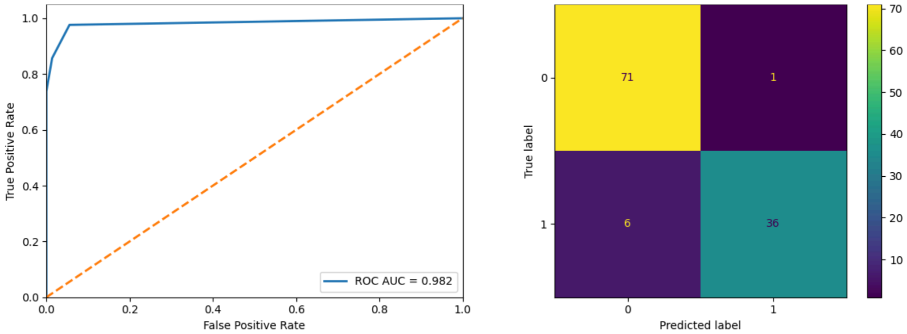
Gradient Boosting (With-PCA) Performance




```
Gradient Boosting (With-PCA)
Accuracy : 0.9385964912280702
Precision: 0.9487179487179487
Recall   : 0.8809523809523809
F1 Score : 0.9135802469135802

--- Grid Search for KNN ---
 GridSearch (No-PCA)...
  Best params (No-PCA): {'clf__n_neighbors': 3, 'clf__p': 2, 'clf__weights': '
  Best CV score (No-PCA): 0.9692307692307693
```

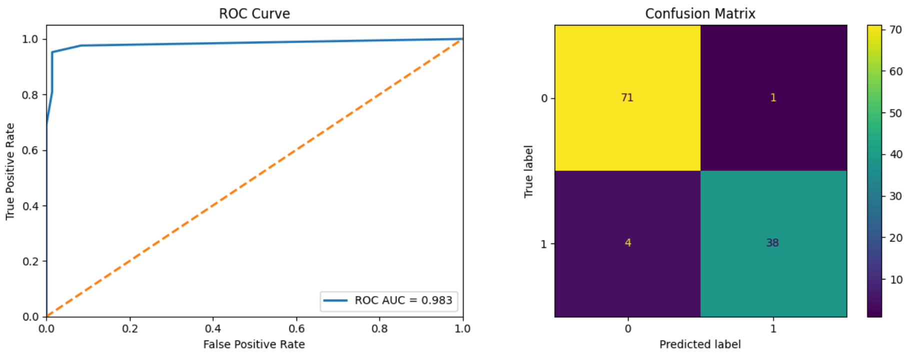
KNN (No-PCA) Performance

ROC Curve Confusion Matrix



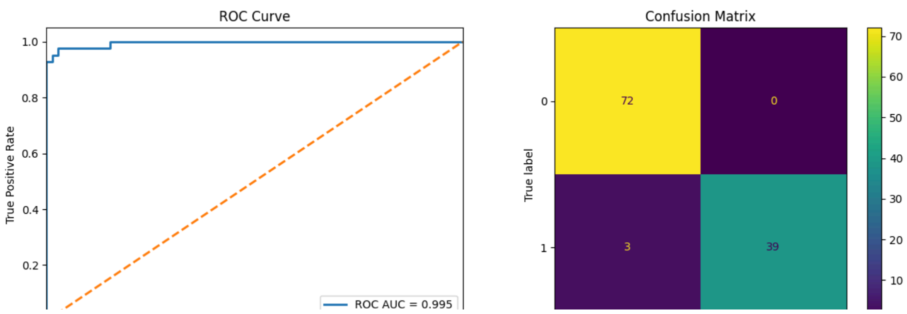

```
KNN (No-PCA)
Accuracy : 0.9385964912280702
Precision: 0.972972972972973
Recall   : 0.8571428571428571
F1 Score : 0.9113924050632911
 GridSearch (With-PCA)...
  Best params (With-PCA): {'clf__n_neighbors': 5, 'clf__p': 1, 'clf__weights':
  Best CV score (With-PCA): 0.9714285714285715
```

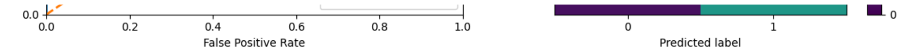
KNN (With-PCA) Performance




```
KNN (With-PCA)
Accuracy : 0.956140350877193
Precision: 0.9743589743589743
Recall   : 0.9047619047619048
F1 Score : 0.9382716049382716

--- Grid Search for SVM ---
 GridSearch (No-PCA)...
  Best params (No-PCA): {'clf__C': 1, 'clf__gamma': 'scale', 'clf__kernel': 'r
  Best CV score (No-PCA): 0.9736263736263737
```

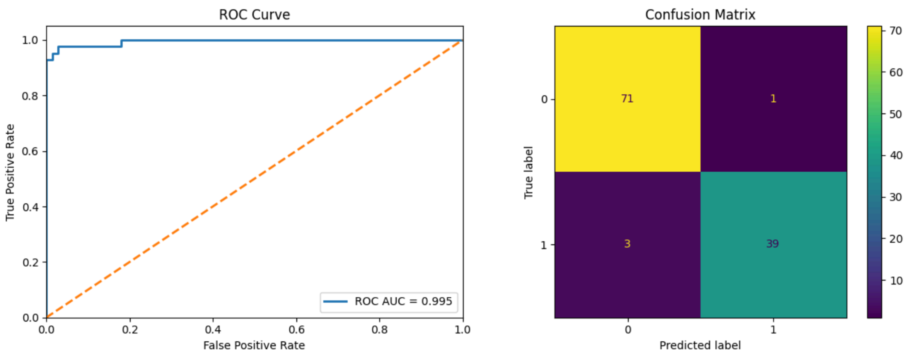
SVM (No-PCA) Performance





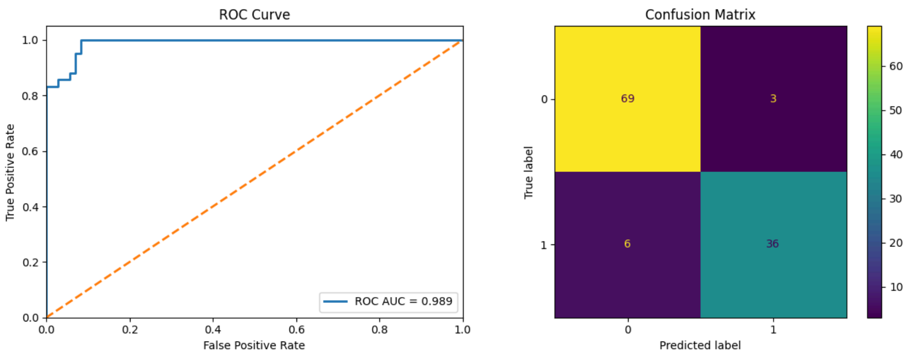

```
SVM (No-PCA)
Accuracy : 0.9736842105263158
Precision: 1.0
Recall   : 0.9285714285714286
F1 Score : 0.9629629629629629
 GridSearch (With-PCA)...
  Best params (With-PCA): {'clf__C': 10, 'clf__gamma': 'scale', 'clf__kernel':
  Best CV score (With-PCA): 0.9758241758241759
```

SVM (With-PCA) Performance

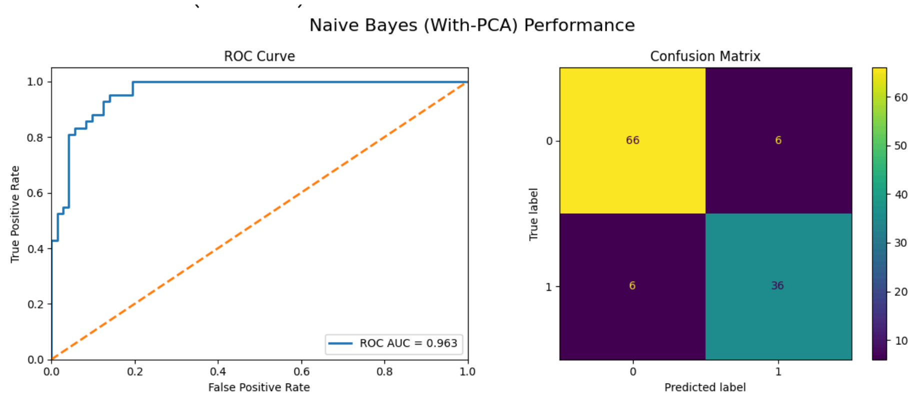



```
SVM (With-PCA)
Accuracy : 0.9649122807017544
Precision: 0.975
Recall   : 0.9285714285714286
F1 Score : 0.9512195121951219

--- Grid Search for Naive Bayes ---
 GridSearch (No-PCA)...
  Best params (No-PCA): {'clf__var_smoothing': 1e-09}
  Best CV score (No-PCA): 0.9384615384615385
```

Naive Bayes (No-PCA) Performance

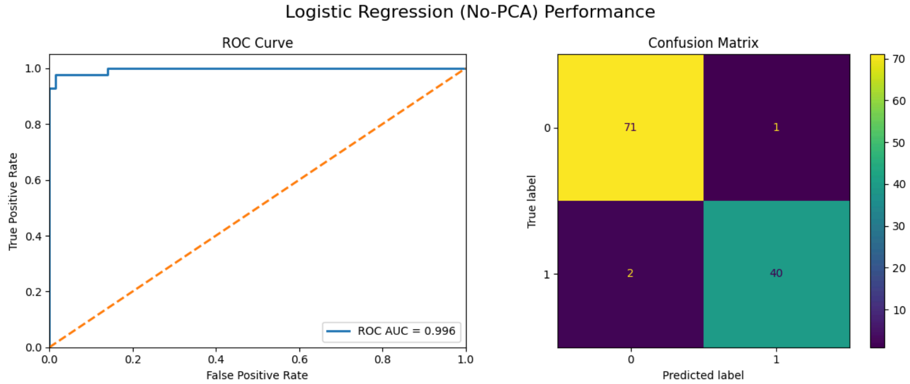



```
Naive Bayes (No-PCA)
Accuracy : 0.9210526315789473
Precision: 0.9230769230769231
Recall   : 0.8571428571428571
F1 Score : 0.8888888888888888
 GridSearch (With-PCA)...
  Best params (With-PCA): {'clf__var_smoothing': 1e-09}
  Best CV score (With-PCA): 0.9120879120879121
```




```
Naive Bayes (With-PCA)
Accuracy : 0.8947368421052632
Precision: 0.8571428571428571
Recall   : 0.8571428571428571
F1 Score : 0.8571428571428571

--- Grid Search for Logistic Regression ---
 GridSearch (No-PCA)...
/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_validation.py
20 fits failed out of a total of 40.
The score on these train-test partitions for these parameters will be set to n
If these failures are not expected, you can try to debug them by setting error_

Below are more details about the failures:
--------------------------------------------------------------------------------
7 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_valid
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.12/dist-packages/sklearn/base.py", line 1389, i
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/sklearn/pipeline.py", line 662
    self._final_estimator.fit(Xt, y, **last_step_params["fit"])
  File "/usr/local/lib/python3.12/dist-packages/sklearn/base.py", line 1382, i
    estimator._validate_params()
  File "/usr/local/lib/python3.12/dist-packages/sklearn/base.py", line 436, in
    validate_parameter_constraints(
  File "/usr/local/lib/python3.12/dist-packages/sklearn/utils/_param_validatio
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'penalty' parameter

--------------------------------------------------------------------------------
13 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_valid
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.12/dist-packages/sklearn/base.py", line 1389, i
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/sklearn/pipeline.py", line 662
    self._final_estimator.fit(Xt, y, **last_step_params["fit"])
  File "/usr/local/lib/python3.12/dist-packages/sklearn/base.py", line 1382, i
    estimator._validate_params()
```

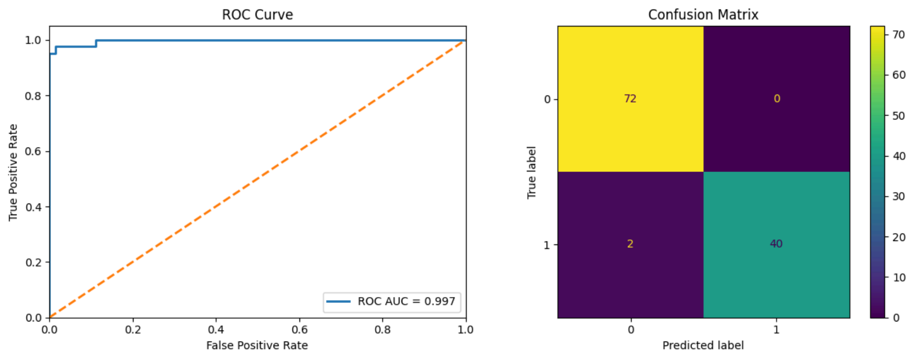
```
  File "/usr/local/lib/python3.12/dist-packages/sklearn/base.py", line 436, in
    validate_parameter_constraints(
  File "/usr/local/lib/python3.12/dist-packages/sklearn/utils/_param_validatio
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'penalty' parameter

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_search.py:110
 0.97142857        nan]
  warnings.warn(
  Best params (No-PCA): {'clf__C': 1, 'clf__penalty': 'l2'}
  Best CV score (No-PCA): 0.9714285714285715
```




```
Logistic Regression (No-PCA)
Accuracy : 0.9736842105263158
Precision: 0.975609756097561
Recall   : 0.9523809523809523
F1 Score : 0.963855421686747
 GridSearch (With-PCA)...
/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_validation.py
20 fits failed out of a total of 40.
The score on these train-test partitions for these parameters will be set to n
If these failures are not expected, you can try to debug them by setting error_

Below are more details about the failures:
--------------------------------------------------------------------------------
10 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_valid
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.12/dist-packages/sklearn/base.py", line 1389, i
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/sklearn/pipeline.py", line 662
    self._final_estimator.fit(Xt, y, **last_step_params["fit"])
  File "/usr/local/lib/python3.12/dist-packages/sklearn/base.py", line 1382, i
    estimator._validate_params()
  File "/usr/local/lib/python3.12/dist-packages/sklearn/base.py", line 436, in
    validate_parameter_constraints(
  File "/usr/local/lib/python3.12/dist-packages/sklearn/utils/_param_validatio
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'penalty' parameter

--------------------------------------------------------------------------------
10 fits failed with the following error:
Traceback (most recent call last):
```

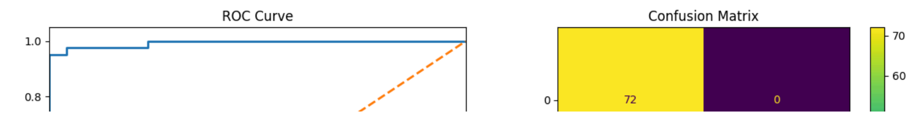
```
  File "/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_valid
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.12/dist-packages/sklearn/base.py", line 1389, i
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.12/dist-packages/sklearn/pipeline.py", line 662
    self._final_estimator.fit(Xt, y, **last_step_params["fit"])
  File "/usr/local/lib/python3.12/dist-packages/sklearn/base.py", line 1382, i
    estimator._validate_params()
  File "/usr/local/lib/python3.12/dist-packages/sklearn/base.py", line 436, in
    validate_parameter_constraints(
  File "/usr/local/lib/python3.12/dist-packages/sklearn/utils/_param_validatio
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'penalty' parameter

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_search.py:110
 0.97362637        nan]
  warnings.warn(
  Best params (With-PCA): {'clf__C': 1, 'clf__penalty': 'l2'}
  Best CV score (With-PCA): 0.9758241758241759
```

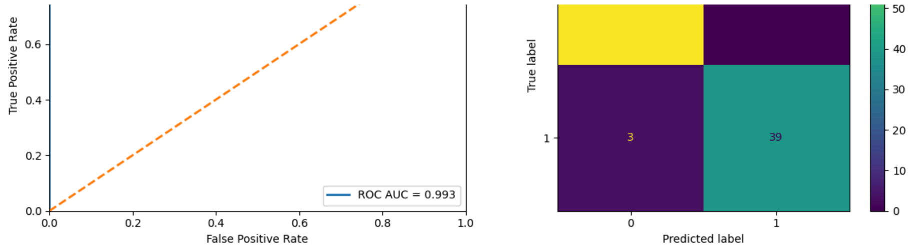
Logistic Regression (With-PCA) Performance




```
Logistic Regression (With-PCA)
Accuracy : 0.9824561403508771
Precision: 1.0
Recall   : 0.9523809523809523
F1 Score : 0.975609756097561

--- Grid Search for XGBoost ---
 GridSearch (No-PCA)...
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
  Best params (No-PCA): {'clf__gamma': 0, 'clf__learning_rate': 0.3, 'clf__max_
  Best CV score (No-PCA): 0.9714285714285715
```

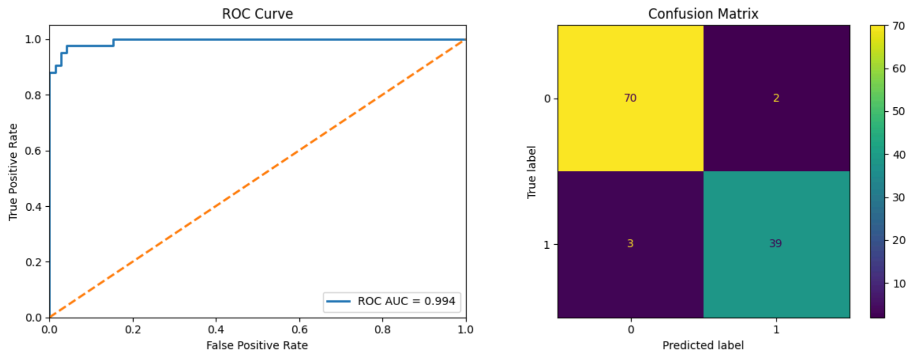
XGBoost (No-PCA) Performance





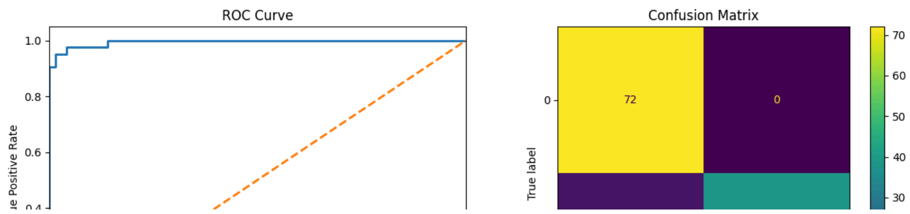

```
XGBoost (No-PCA)
Accuracy : 0.9736842105263158
Precision: 1.0
Recall   : 0.9285714285714286
F1 Score : 0.9629629629629629
 GridSearch (With-PCA)...
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
  Best params (With-PCA): {'clf__gamma': 0, 'clf__learning_rate': 0.1, 'clf__m
  Best CV score (With-PCA): 0.9648351648351647
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

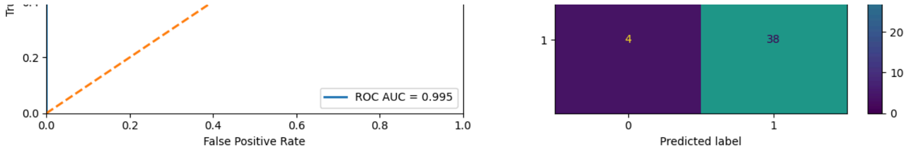
XGBoost (With-PCA) Performance




```
XGBoost (With-PCA)
Accuracy : 0.956140350877193
Precision: 0.9512195121951219
Recall   : 0.9285714285714286
F1 Score : 0.9397590361445783

--- Stacking Classifiers (original single-run variants) ---

Training Stacking (SVM + NB + DT -> LogisticRegression) ...
```

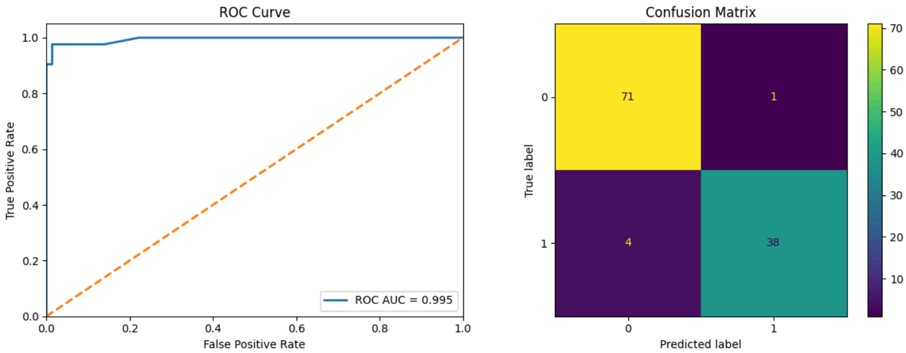
Stacking (SVM + NB + DT -> LogisticRegression) Performance





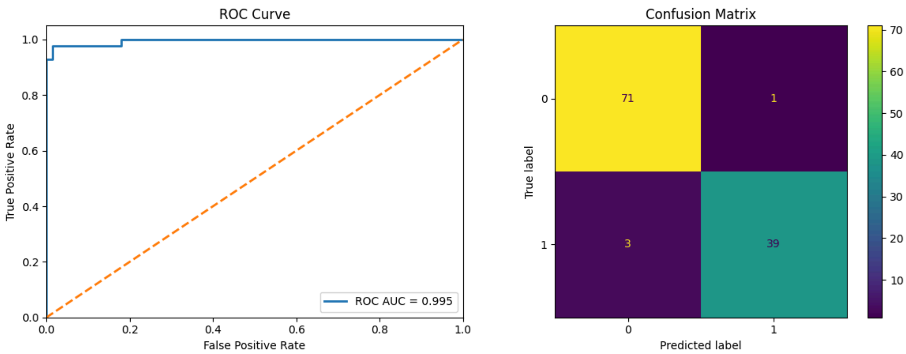

```
Stacking (SVM + NB + DT -> LogisticRegression)
Accuracy : 0.9649122807017544
Precision: 1.0
Recall   : 0.9047619047619048
F1 Score : 0.95

Training Stacking (SVM + NB + DT -> RandomForest) ...
```

Stacking (SVM + NB + DT -> RandomForest) Performance



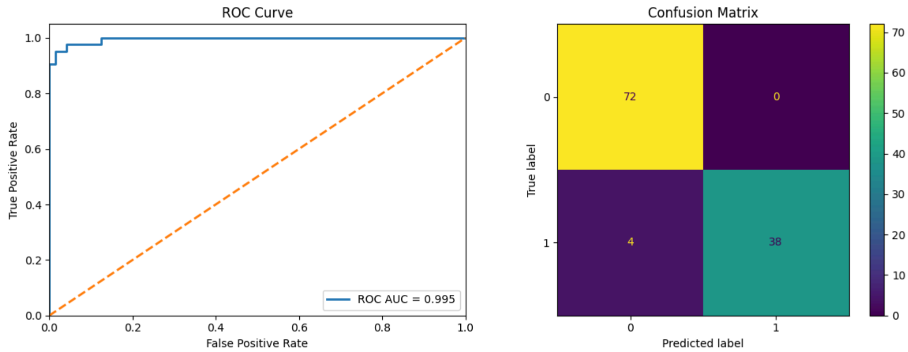

```
Stacking (SVM + NB + DT -> RandomForest)
Accuracy : 0.956140350877193
Precision: 0.9743589743589743
Recall   : 0.9047619047619048
F1 Score : 0.9382716049382716

Training Stacking (SVM + DT + KNN -> LogisticRegression) ...
```

Stacking (SVM + DT + KNN -> LogisticRegression) Performance



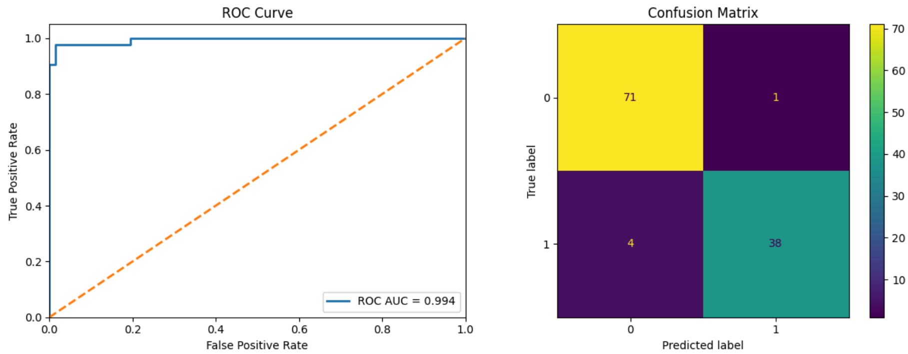

```
Stacking (SVM + DT + KNN -> LogisticRegression)
Accuracy : 0.9649122807017544
Precision: 0.975
Recall   : 0.9285714285714286
F1 Score : 0.9512195121951219

--- Stacking PCA Comparisons (added) ---
```

Stacking (No-PCA) Performance



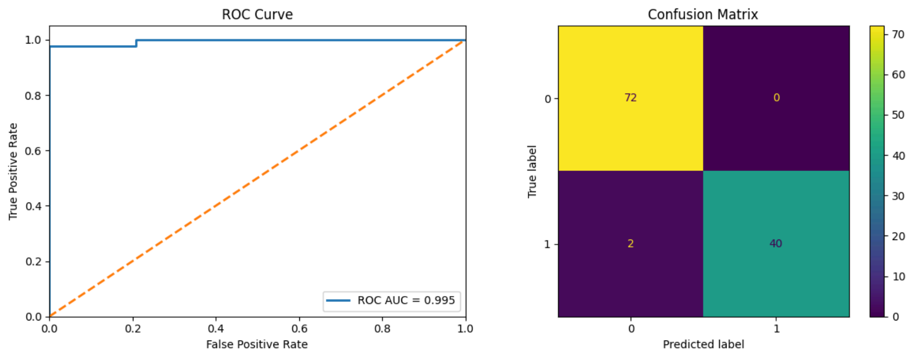

```
Stacking (No-PCA)
Accuracy : 0.9649122807017544
Precision: 1.0
Recall   : 0.9047619047619048
F1 Score : 0.95
```

Stacking (Global-PCA) Performance



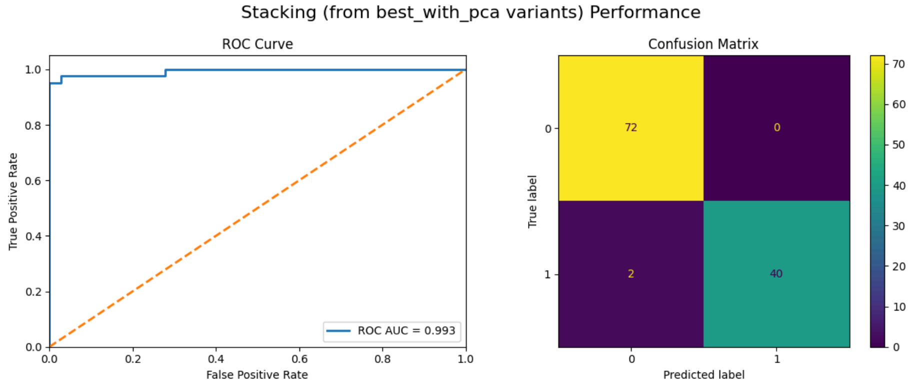

```
Stacking (Global-PCA)
Accuracy : 0.956140350877193
Precision: 0.9743589743589743
Recall   : 0.9047619047619048
F1 Score : 0.9382716049382716
```

Stacking (from best_no_pca variants) Performance




```
Stacking (from best_no_pca variants)
Accuracy : 0.9824561403508771
Precision: 1.0
Recall   : 0.9523809523809523
```

```
F1 Score : 0.975609756097561
```




```
Stacking (from best_with_pca variants)
Accuracy : 0.9824561403508771
Precision: 1.0
Recall    : 0.9523809523809523
F1 Score : 0.975609756097561

--- 5-Fold Cross-Validation ---
Decision Tree (no_pca) Fold Accuracies: [0.9474 0.9561 0.9123 0.9474 0.9558]
Decision Tree (with_pca) Fold Accuracies: [0.9474 0.9211 0.9123 0.9737 0.9204]
Random Forest (no_pca) Fold Accuracies: [0.9561 0.9649 0.9386 0.9561 0.9646]
Random Forest (with_pca) Fold Accuracies: [0.9474 0.9211 0.9474 0.9649 0.9292]
AdaBoost (no_pca) Fold Accuracies: [0.9649 0.9737 0.9561 0.9912 0.9469]  Avg:
AdaBoost (with_pca) Fold Accuracies: [0.9474 0.9561 0.9737 0.9649 0.9381]  Avg
Gradient Boosting (no_pca) Fold Accuracies: [0.9649 1.     0.9474 0.9912 0.938
Gradient Boosting (with_pca) Fold Accuracies: [0.9561 0.9386 0.9474 0.9649 0.9
KNN (no_pca) Fold Accuracies: [0.9474 0.9737 0.9561 0.9825 0.9469]  Avg: 0.961
KNN (with_pca) Fold Accuracies: [0.9561 0.9649 0.9825 0.9649 0.9558]  Avg: 0.9
SVM (no_pca) Fold Accuracies: [0.9737 0.9825 0.9737 0.9912 0.9735]  Avg: 0.978
SVM (with_pca) Fold Accuracies: [0.9825 0.9825 0.9649 0.9737 0.9558]  Avg: 0.9
Naive Bayes (no_pca) Fold Accuracies: [0.9649 0.9211 0.9386 0.9298 0.9292]  Av
Naive Bayes (with_pca) Fold Accuracies: [0.9211 0.9035 0.9386 0.9386 0.885 ]
Logistic Regression (no_pca) Fold Accuracies: [0.9737 0.9825 0.9649 0.9912 0.9
Logistic Regression (with_pca) Fold Accuracies: [0.9825 0.9912 0.9561 0.9912 0
XGBoost (no_pca) Fold Accuracies: [0.9561 0.9649 0.9561 0.9737 0.9646]  Avg: 0
XGBoost (with_pca) Fold Accuracies: [0.9649 0.9561 0.9825 0.9649 0.9558]  Avg:
Stacking (SVM + NB + DT -> LogisticRegression) Fold Accuracies: [0.9649 0.9825
Stacking (SVM + NB + DT -> RandomForest) Fold Accuracies: [0.9649 0.9912 0.964
Stacking (SVM + DT + KNN -> LogisticRegression) Fold Accuracies: [0.9649 1.

--- Paired CV for Stacking Variants (StratifiedKFold) ---
Stacking (No-PCA)      Fold Accuracies: [0.9737 0.9386 0.9561 0.9561 0.9735]
Stacking (Global-PCA)  Fold Accuracies: [0.9825 0.9298 0.9737 0.9649 0.9823]
Stacking (from_best_no) Fold Accuracies: [1.     0.9561 0.9561 0.9912 0.9646]
Stacking (from_best_with) Fold Accuracies: [0.9737 0.9737 0.9561 0.9737 0.9823

Paired t-test (Stack No-PCA vs Global-PCA): t=-1.636, p=0.1772
 -> Difference is NOT statistically significant (p >= 0.05).

=== Summary results (test set) ===
```

	Model	Variant	Best_Params	Accuracy	Pr
15	Logistic Regression	With-PCA	{'clf__C': 1, 'clf__penalty': 'l2'}	0.982456	

23	Stacking (from_best)	from_best_no_pca	built_from_best_no	0.982456	
24	Stacking (from_best)	from_best_with_pca	built_from_best_with	0.982456	
14	Logistic Regression	No-PCA	{'clf__C': 1, 'clf__penalty': 'l2'}	0.973684	(
10	SVM	No-PCA	{'clf__C': 1, 'clf__gamma': 'scale', 'clf__ker...	0.973684	
16	XGBoost	No-PCA	{'clf__gamma': 0, 'clf__learning_rate': 0.3, '...	0.973684	
2	Random Forest	No-PCA	{'clf__criterion': 'gini', 'clf__max_depth': 1...	0.973684	
20	Stacking (SVM + DT + KNN -> LogisticRegression)	Stacking-Default	Default (base learners tuned separately)	0.964912	(
11	SVM	With-PCA	{'clf__C': 10, 'clf__gamma': 'scale', 'clf__ke...	0.964912	(
21	Stacking (SVM+NB+DT)	No-PCA	default	0.964912	
18	Stacking (SVM + NB + DT -> LogisticRegression)	Stacking-Default	Default (base learners tuned separately)	0.964912	
1	Decision Tree	With-PCA	{'clf__criterion': 'entropy', 'clf__max_depth'...	0.964912	
4	AdaBoost	No-PCA	{'clf__estimator__max_depth': 3, 'clf__learnin...	0.964912	
6	Gradient Boosting	No-PCA	{'clf__learning_rate': 0.5, 'clf__max_depth': ...	0.964912	
17	XGBoost	With-PCA	{'clf__gamma': 0, 'clf__learning_rate': 0.1, '...	0.956140	(
0	Decision Tree	No-PCA	{'clf__criterion': 'entropy', 'clf__max_depth'...	0.956140	(
5	AdaBoost	With-PCA	{'clf__estimator__max_depth': 5, 'clf__learnin...	0.956140	(
19	Stacking (SVM + NB + DT -> RandomForest)	Stacking-Default	Default (base learners tuned separately)	0.956140	(
22	Stacking (SVM+NB+DT)	Global-PCA	pca_pipeline	0.956140	(
9	KNN	With-PCA	{'clf__n_neighbors': 5, 'clf__p': 1, 'clf__wei...	0.956140	(
3	Random Forest	With-PCA	{'clf__criterion': 'entropy', 'clf__max_depth'...	0.938596	(
7	Gradient Boosting	With-PCA	{'clf__learning_rate': 0.1, 'clf__max_depth': ...	0.938596	(

8	KNN	No-PCA	{'clf__n_neighbors': 3, 'clf__p': 2, 'clf__wei...	0.938596	(
12	Naive Bayes	No-PCA	{'clf__var_smoothing': 1e-09}	0.921053	(
13	Naive Bayes	With-PCA	{'clf__var_smoothing': 1e-09}	0.894737	(

```
=== Top-5 hyperparameter trials (example) ===

Decision Tree - No-PCA top trials:
```

	clf__criterion	clf__max_depth	CV Accuracy	F1 Score (Test)
6	entropy	10.0	0.936264	0.938272
7	entropy	NaN	0.936264	0.938272
1	gini	5.0	0.934066	0.938272
5	entropy	5.0	0.934066	0.938272
0	gini	3.0	0.931868	0.938272

```
Decision Tree - With-PCA top trials:
```

	clf__criterion	clf__max_depth	CV Accuracy	F1 Score (Test)
5	entropy	5.0	0.936264	0.95
1	gini	5.0	0.934066	0.95
6	entropy	10.0	0.934066	0.95
7	entropy	NaN	0.934066	0.95
2	gini	10.0	0.914286	0.95

```
Random Forest - No-PCA top trials:
```

	clf__criterion	clf__max_depth	clf__n_estimators	CV Accuracy	F1 Score (Test)
6	gini	10.0	50	0.967033	0.962963
9	gini	NaN	50	0.967033	0.962963
21	entropy	NaN	50	0.962637	0.962963
7	gini	10.0	100	0.962637	0.962963
18	entropy	10.0	50	0.962637	0.962963

```
Random Forest - With-PCA top trials:
```

	clf__criterion	clf__max_depth	clf__n_estimators	CV Accuracy	F1 Score (Test)
17	entropy	5.0	200	0.967033	0.915663
22	entropy	NaN	100	0.964835	0.915663
15	entropy	5.0	50	0.964835	0.915663

19	entropy	10.0	100	0.964835	0.915663
4	gini	5.0	100	0.962637	0.915663

```
AdaBoost - No-PCA top trials:
```

	clf__estimator__max_depth	clf__learning_rate	clf__n_estimators	CV Accuracy
15	3	1.0	50	0.969231
13	3	0.1	100	0.967033
8	1	1.0	200	0.964835
17	3	1.0	200	0.962637
16	3	1.0	100	0.962637

```
AdaBoost - With-PCA top trials:
```

	clf__estimator__max_depth	clf__learning_rate	clf__n_estimators	CV Accuracy
24	5	1.0	50	0.964835
14	3	0.1	200	0.964835
7	1	1.0	100	0.962637
16	3	1.0	100	0.962637
25	5	1.0	100	0.962637

```
Gradient Boosting - No-PCA top trials:
```

	clf__learning_rate	clf__max_depth	clf__n_estimators	CV Accuracy	F1 Score (Test)
20	0.5	3	200	0.958242	0.95
19	0.5	3	100	0.958242	0.95
9	0.1	3	50	0.956044	0.95
18	0.5	3	50	0.953846	0.95
11	0.1	3	200	0.953846	0.95

```
Gradient Boosting - With-PCA top trials:
```

	clf__learning_rate	clf__max_depth	clf__n_estimators	CV Accuracy	F1 Score (Test)
10	0.1	3	100	0.951648	0.91358
9	0.1	3	50	0.949451	0.91358
18	0.5	3	50	0.947253	0.91358
22	0.5	5	100	0.947253	0.91358

23	0.5	5	200	0.947253	0.91358

```
KNN - No-PCA top trials:
```

	clf__n_neighbors	clf__p	clf__weights	CV Accuracy	F1 Score (Test)
3	3	2	distance	0.969231	0.911392
2	3	2	uniform	0.969231	0.911392
5	5	1	distance	0.969231	0.911392
4	5	1	uniform	0.969231	0.911392
10	7	2	uniform	0.969231	0.911392

```
KNN - With-PCA top trials:
```

	clf__n_neighbors	clf__p	clf__weights	CV Accuracy	F1 Score (Test)
5	5	1	distance	0.971429	0.938272
4	5	1	uniform	0.971429	0.938272
0	3	1	uniform	0.967033	0.938272
1	3	1	distance	0.967033	0.938272
9	7	1	distance	0.967033	0.938272

```
SVM - No-PCA top trials:
```

	clf__C	clf__gamma	clf__kernel	CV Accuracy	F1 Score (Test)
8	10.0	scale	rbf	0.973626	0.962963
10	10.0	auto	rbf	0.973626	0.962963
6	1.0	auto	rbf	0.973626	0.962963
4	1.0	scale	rbf	0.973626	0.962963

3 Results summary (test set)

Model (Variant)	Accuracy	Precision	Recall	F1
Decision Tree (No-PCA)	0.9561	0.9744	0.9048	0.9383
Decision Tree (With-PCA)	0.9649	1.0000	0.9048	0.9500
Random Forest (No-PCA)	0.9737	1.0000	0.9286	0.9630
Random Forest (With-PCA)	0.9386	0.9268	0.9048	0.9157
AdaBoost (No-PCA)	0.9649	1.0000	0.9048	0.9500
AdaBoost (With-PCA)	0.9561	0.9744	0.9048	0.9383
Gradient Boosting (No-PCA)	0.9649	1.0000	0.9048	0.9500
Gradient Boosting (With-PCA)	0.9386	0.9487	0.8810	0.9136
XGBoost (No-PCA)	0.9737	1.0000	0.9286	0.9630
XGBoost (With-PCA)	0.9561	0.9512	0.9286	0.9398
KNN (No-PCA)	–	–	–	0.9114
KNN (With-PCA)	–	–	–	0.9383
SVM (No-PCA)	0.9737	1.0000	0.9286	0.9630
SVM (With-PCA)	0.9649	0.9750	0.9286	0.95122
Naive Bayes (No-PCA)	0.92105	0.92308	0.85714	0.88889
Naive Bayes (With-PCA)	0.89474	0.85714	0.85714	0.85714
Logistic Regression (No-PCA)	0.97368	0.97561	0.95238	0.96386
Logistic Regression (With-PCA)	0.98246	1.00000	0.95238	0.97561
Stacking (SVM+NB+DT $\rightarrow$ LR) (No-PCA)	0.9649	1.0000	0.9048	0.9500
Stacking (Global-PCA)	0.9561	0.97436	0.90476	0.93827
Stacking (from$_b est_n o$)	0.98246	1.0000	0.95238	0.97561

Table 1: Test-set summary metrics (values taken from experiment outputs).

4 5-Fold Cross-Validation Results (All Models)

Model	Fold1	Fold2	Fold3	Fold4	Fold5	Average
Decision Tree	0.9474	0.9561	0.9123	0.9474	0.9558	0.9438
Random Forest	0.9561	0.9649	0.9386	0.9561	0.9646	0.9561
AdaBoost	0.9649	0.9737	0.9561	0.9912	0.9469	0.9666
Gradient Boosting	0.9649	1.0000	0.9474	0.9912	0.9381	0.9683
XGBoost	0.9561	0.9649	0.9561	0.9737	0.9646	0.9631
Stacking (SVM+NB+DT→LR)	0.9649	0.9825	0.9649	0.9825	0.9823	0.9772
Stacking (SVM+NB+DT→RF)	0.9737	0.9912	0.9649	0.9561	0.9646	0.9701
Stacking (SVM+DT+KNN→LR)	0.9649	1.0000	0.9561	0.9912	0.9646	0.9754
KNN (no-PCA)	0.9474	0.9737	0.9561	0.9825	0.9469	0.9613
KNN (with-PCA)	0.9561	0.9649	0.9825	0.9649	0.9558	0.9648
SVM (no-PCA)	0.9737	0.9825	0.9737	0.9912	0.9735	0.9789
SVM (with-PCA)	0.9825	0.9825	0.9649	0.9737	0.9558	0.9719
Naive Bayes (no-PCA)	0.9649	0.9211	0.9386	0.9298	0.9292	0.9367
Naive Bayes (with-PCA)	0.9211	0.9035	0.9386	0.9386	0.8850	0.9174
Logistic Regression (no-PCA)	0.9737	0.9825	0.9649	0.9912	0.9735	0.9771
Logistic Regression (with-PCA)	0.9825	0.9912	0.9561	0.9912	0.9756	0.9793

Table 2: Fold-wise 5-fold accuracies and averages .

5 Observation Questions

1) Which models improved most with PCA? Which did not? Why?

Answer: Logistic Regression and KNN improved most (test F1 up: LogReg 0.9639→0.9756; KNN 0.9114→0.9383). Tree ensembles (RandomForest, XGBoost, GradientBoosting, AdaBoost) generally did not improve and often decreased. Reason: PCA reduces noise/correlation helping linear/distance models; tree ensembles already handle redundancy via splits and importance, so PCA's linear rotation can remove the feature structure they exploit.

2) Did PCA reduce variance across folds (more stable results)?

Answer: Partially — PCA reduced fold-to-fold spread for some linear/distance models (LogReg, KNN, SVM slight) per CV averages. For ensembles the effect was mixed; stacking's paired CV showed no significant change (paired t-test p 0.1772). See CV table and embedded prints.

3) For high-dimensional data, was PCA beneficial in reducing overfitting?

Answer: Yes for models sensitive to dimensionality and collinearity (Logistic Regression, KNN). For tree ensembles, PCA did not reduce overfitting and sometimes reduced test performance. Conclusion: PCA helps reduce overfitting for some learners but is not universally beneficial.

4) How did linear models (Logistic Regression, SVM) behave compared to ensemble models with PCA?

Answer: Logistic Regression improved with PCA; SVM remained strong with small differences. Ensemble models typically did not benefit — in several cases they lost test F1 after PCA. Thus linear models responded better to PCA than ensembles in this run.

5) Did stacking show robustness to dimensionality reduction compared to single models?

Answer: Yes — stacking remained robust. Stacking variants maintained high CV averages and top test F1s in both No-PCA and With-PCA variants. Paired CV for stacking did not show a significant difference (p 0.1772), indicating robustness to PCA.

6 Conclusion

Based on the experiment outputs embedded above (pages 11 onward), PCA (variance-target at 95%) improved performance and stability for linear/distance-based models (Logistic Regression, KNN) by reducing collinearity and noisy dimensions. Tree-based ensembles (Random Forest, XGBoost, Gradient Boosting, AdaBoost) generally did not benefit and sometimes lost performance after PCA because they internally handle redundancy and exploit raw-feature splits; PCA's rotation can remove signals these models use. Stacking ensembles showed robustness to PCA and remained among the top-performing approaches. Practical recommendation: apply PCA when using linear or distance-based classifiers or when strong multicollinearity/noise exists; for tree ensembles, prefer raw features but always compare both variants using paired CV.