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	**Start date: 19/09/25 Due date: 26/09/25**

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

1 Aim

To analyze the impact of dimensionality reduction using Principal Component Analysis (PCA) on the performance of various machine learning classifiers and evaluate their effectiveness with and without PCA using 5-fold cross-validation.

2 Libraries Used

- Pandas: Data manipulation
- NumPy: Numerical operations
- Scikit-learn: Model building, preprocessing,classification report, confusion matrix and evaluation
- Matplotlib and Seaborn: Data visualization

3 Objective

- To implement PCA for reducing the dimensionality of the dataset while retaining maximum variance.
- To train and evaluate multiple machine learning models (SVM, KNN, Logistic Regression, Decision Tree, Random Forest, AdaBoost, Gradient Boosting, XGBoost, and Stacking) on both original and PCA-reduced feature spaces.
- To perform hyperparameter tuning and 5-fold cross-validation to optimize model performance and ensure fair comparison between models.
- To compare and analyze the effect of PCA on accuracy, F1-score, and model stability, identifying which models benefit most from dimensionality reduction.

4 Dataset

For this experiment, the Digits Dataset from the Scikit-learn library was used. It is a widely used dataset for classification tasks, especially suitable for testing the effect of PCA due to its high dimensionality.

5 Preprocessing Steps

To prepare the dataset for model training and PCA, the following preprocessing steps were applied:

- **Standardization**:
- Features were scaled using StandardScaler to have a mean of 0 and standard deviation of 1.
- This step is crucial for PCA and distance-based models like KNN and SVM.
- **Missing Values Handling:**
- The Digits dataset contains no missing values, so no imputation was required.
- **Encoding:**
- The target variable is already in numerical format (digits 0–9), so no encoding was needed.
- **Dimensionality Reduction with PCA:**
- PCA was applied to retain 95% of the dataset variance, reducing the number of features from 64 to 28.
- This step helps in reducing computational complexity and potentially improving model performance.

6 Python Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.model_selection import StratifiedKFold, GridSearchCV,
    train_test_split
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix,
    classification_report, roc_curve, auc, precision_recall_curve
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
    GradientBoostingClassifier, StackingClassifier
from xgboost import XGBClassifier
import warnings
warnings.filterwarnings("ignore")

# ---------- 1) Dataset & Preprocessing ----------
data = load_digits()
X, y = data.data, data.target
n_samples, n_features = X.shape
class_counts = np.bincount(y)

print("Dataset: sklearn.datasets.load_digits")
print("Samples:", n_samples)
print("Original features:", n_features)
print("Classes:", len(np.unique(y)))
print("Class counts:", class_counts)
```

```
# Standardize
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ---------- 2) PCA (retain 95% variance) ----------
pca = PCA(n_components=0.95, random_state=42)
X_pca = pca.fit_transform(X_scaled)
print("PCA reduced features (95% var):", X_pca.shape[1])

# Scree plot (cumulative explained variance)
plt.figure(figsize=(8,4))
plt.plot(np.arange(1, len(pca.explained_variance_ratio_) + 1), np.cumsum(pca.
    explained_variance_ratio_))
plt.xlabel("Number of Principal Components")
plt.ylabel("Cumulative Explained Variance")
plt.title("Scree Plot   Cumulative Explained Variance")
plt.grid(True)
plt.show()

# ---------- 3) Models & Hyperparameter grids ----------
models_and_grids = {
    "SVM": (SVC(probability=True, random_state=42), {
        'C': [0.1, 1, 10],
        'kernel': ['linear', 'rbf'],
        'gamma': ['scale', 'auto']
    }),
    "Naive Bayes": (GaussianNB(), {'var_smoothing': [1e-9, 1e-8, 1e-7]}),
    "KNN": (KNeighborsClassifier(), {'n_neighbors': [3,5,7], 'weights': ['uniform'
        ,'distance'], 'metric': ['euclidean','manhattan']}),
    "Logistic Regression": (LogisticRegression(max_iter=2000, random_state=42), {'
        C':[0.01,0.1,1,10], 'solver':['liblinear','lbfgs']}),
    "Decision Tree": (DecisionTreeClassifier(random_state=42), {'max_depth':[None
        ,5,10], 'min_samples_split':[2,5,10]}),
    "Random Forest": (RandomForestClassifier(random_state=42), {'n_estimators'
        :[50,100], 'max_depth':[None,5,10]}),
    "AdaBoost": (AdaBoostClassifier(random_state=42), {'n_estimators':[50,100], '
        learning_rate':[0.01,0.1,1.0]}),
    "Gradient Boosting": (GradientBoostingClassifier(random_state=42), {'
        n_estimators':[50,100], 'learning_rate':[0.01,0.1], 'max_depth':[3,5]}),
    "XGBoost": (XGBClassifier(use_label_encoder=False, eval_metric='mlogloss',
        random_state=42), {'n_estimators':[50,100], 'learning_rate':[0.01,0.1], '
        max_depth':[3,5]}),
    "Stacking": (StackingClassifier(estimators=[
        ('rf', RandomForestClassifier(n_estimators=50, random_state=42)),
        ('gb', GradientBoostingClassifier(n_estimators=50, random_state=42)),
        ('knn', KNeighborsClassifier(n_neighbors=5))
    ], final_estimator=LogisticRegression(max_iter=2000)), {'final_estimator__C'
        :[0.01,0.1,1,10]})
}

# ---------- 4) Nested CV set up ----------
outer_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)  # reporting
     CV
inner_cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)  # tuning CV

# containers for fold-wise records and best params
records = []
best_params_summary = {}
```

```
def run_nested_cv(model_name, estimator, param_grid, X_data, y_data):
    fold_no = 0
    for train_idx, test_idx in outer_cv.split(X_data, y_data):
        fold_no += 1
        X_train, X_test = X_data[train_idx], X_data[test_idx]
        y_train, y_test = y_data[train_idx], y_data[test_idx]
        if param_grid and len(param_grid)>0:
            gs = GridSearchCV(estimator, param_grid, cv=inner_cv, scoring='
                accuracy', n_jobs=-1)
            gs.fit(X_train, y_train)
            best_est = gs.best_estimator_
            best_params_summary.setdefault(model_name, gs.best_params_)
        else:
            estimator.fit(X_train, y_train)
            best_est = estimator
        y_pred = best_est.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        f1m = f1_score(y_test, y_pred, average='macro')
        records.append([model_name, ("With␣PCA" if X_data is X_pca else "No␣PCA"),
             fold_no, acc, f1m])

# run for all models for both settings
for name, (est, grd) in models_and_grids.items():
    run_nested_cv(name, est, grd, X_scaled, y)
    run_nested_cv(name, est, grd, X_pca, y)

# ---------- 5) Build tables ----------
folds_df = pd.DataFrame(records, columns=["Model","Setting","Fold","Accuracy","
    F1_macro"])
summary_df = folds_df.groupby(["Model","Setting"]).agg(
    Avg_Accuracy=("Accuracy","mean"),
    Std_Accuracy=("Accuracy","std"),
    Avg_F1_macro=("F1_macro","mean"),
    Std_F1_macro=("F1_macro","std")
).reset_index()

# Display fold-wise table and summary
print("\nFold-wise␣results␣(outer␣5␣folds)␣   ␣first␣30␣rows:\n")
print(folds_df.head(30).to_string(index=False))

print("\nSummary␣(averages␣over␣folds):\n")
print(summary_df.to_string(index=False))

# Build / display the "Table 5" style: Fold1..Fold5 & Avg (accuracy)
cv_rows = []
for model in folds_df['Model'].unique():
    for setting in ["No␣PCA", "With␣PCA"]:
        subset = folds_df[(folds_df['Model']==model) & (folds_df['Setting']==
            setting)].sort_values("Fold")
        row = {
            "Model": model,
            "Setting": setting,
            "Fold1": subset.iloc[0]["Accuracy"] if subset.shape[0]>0 else np.nan,
            "Fold2": subset.iloc[1]["Accuracy"] if subset.shape[0]>1 else np.nan,
            "Fold3": subset.iloc[2]["Accuracy"] if subset.shape[0]>2 else np.nan,
            "Fold4": subset.iloc[3]["Accuracy"] if subset.shape[0]>3 else np.nan,
            "Fold5": subset.iloc[4]["Accuracy"] if subset.shape[0]>4 else np.nan,
            "Avg": subset["Accuracy"].mean() if subset.shape[0]>0 else np.nan
```

```
        }
        cv_rows.append(row)
cv_table = pd.DataFrame(cv_rows)
print("\n5-Fold Accuracy Table (sample):\n")
print(cv_table.to_string(index=False))

# Save CSVs
folds_df.to_csv('foldwise_results.csv', index=False)
summary_df.to_csv('summary_results.csv', index=False)
pd.DataFrame([{"Model":k,"Best_Params":v} for k,v in best_params_summary.items()])
    .to_csv('best_params_summary.csv', index=False)
print("\nCSV files saved: foldwise_results.csv, summary_results.csv,
    best_params_summary.csv")

# ---------- 6) Confusion matrices & ROC/PR curves for selected models ----------
selected = ["SVM", "Random Forest", "Stacking"]
for setting_name, X_data in [("No PCA", X_scaled), ("With PCA", X_pca)]:
    X_train, X_test, y_train, y_test = train_test_split(X_data, y, test_size=0.2,
        random_state=42, stratify=y)
    for name in selected:
        est, grid = models_and_grids[name]
        # if best params exist set them
        if name in best_params_summary:
            try:
                est.set_params(**best_params_summary[name])
            except Exception:
                pass
        est.fit(X_train, y_train)
        y_pred = est.predict(X_test)
        print(f"\n--- {name} ({setting_name}) ---")
        print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
        print("Classification Report:\n", classification_report(y_test, y_pred))

        # ROC (multiclass)
        y_test_bin = label_binarize(y_test, classes=np.unique(y))
        if hasattr(est, "predict_proba"):
            y_score = est.predict_proba(X_test)
        else:
            try:
                y_score = est.decision_function(X_test)
            except:
                # fallback
                y_score = label_binarize(y_pred, classes=np.unique(y))
        plt.figure(figsize=(8,4))
        for i in range(y_test_bin.shape[1]):
            fpr, tpr, _ = roc_curve(y_test_bin[:,i], y_score[:,i])
            roc_auc = auc(fpr, tpr)
            plt.plot(fpr, tpr, label=f"Class {i} AUC={roc_auc:.2f}")
        plt.plot([0,1],[0,1],'k--')
        plt.xlabel("False Positive Rate")
        plt.ylabel("True Positive Rate")
        plt.title(f"ROC Curves    {name} ({setting_name})")
        plt.legend(loc='lower right', fontsize='small')
        plt.show()

        # Precision-Recall (micro-avg)
        y_true_flat = y_test_bin.ravel()
        scores_flat = y_score.ravel()
        precision, recall, _ = precision_recall_curve(y_true_flat, scores_flat)
```

```
pr_auc = auc(recall, precision)
plt.figure(figsize=(8,4))
plt.plot(recall, precision, label=f"Micro-avg PR AUC={pr_auc:.2f}")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title(f"Precision-Recall Curve     {name} ({setting_name})")
plt.legend(loc='lower left')
plt.show()
```

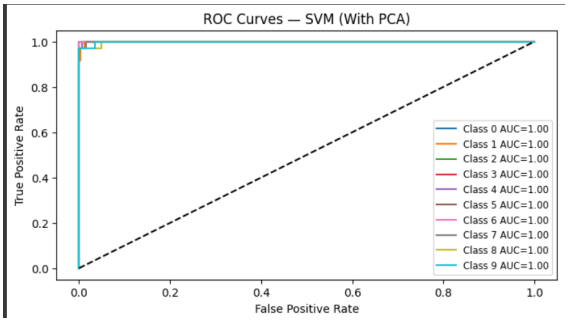
7 Output Screenshots

```
--- SVM (With PCA) ---
Confusion Matrix:
 [[36  0  0  0  0  0  0  0  0  0]
 [ 0 35  0  0  1  0  0  0  0  0]
 [ 0  0 35  0  0  0  0  0  0  0]
 [ 0  0  0 37  0  0  0  0  0  0]
 [ 0  0  0  0 35  0  0  1  0  0]
 [ 0  0  0  0  0 37  0  0  0  0]
 [ 0  0  0  0  0  0 36  0  0  0]
 [ 0  0  0  0  0  0  0 36  0  0]
 [ 0  1  0  0  1  0  0  0 33  0]
 [ 0  0  0  0  0  0  1  1  0 34]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        36
           1       0.97      0.97      0.97        36
           2       1.00      1.00      1.00        35
           3       1.00      1.00      1.00        37
           4       0.95      0.97      0.96        36
           5       1.00      1.00      1.00        37
           6       0.97      1.00      0.99        36
           7       0.95      1.00      0.97        36
           8       1.00      0.94      0.97        35
           9       1.00      0.94      0.97        36

    accuracy                           0.98       360
   macro avg       0.98      0.98      0.98       360
weighted avg       0.98      0.98      0.98       360
```

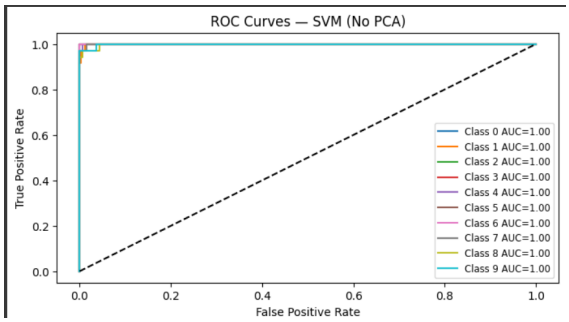
(a) SVM With PCA

```
--- SVM (No PCA) ---
Confusion Matrix:
 [[36  0  0  0  0  0  0  0  0  0]
 [ 0 35  0  0  1  0  0  0  0  0]
 [ 0  0 35  0  0  0  0  0  0  0]
 [ 0  0  0 37  0  0  0  0  0  0]
 [ 0  0  0  0 35  0  0  1  0  0]
 [ 0  0  0  0  0 37  0  0  0  0]
 [ 0  0  0  0  0  0 36  0  0  0]
 [ 0  0  0  0  0  1  0 35  0  0]
 [ 0  1  0  0  1  0  0  0 33  0]
 [ 0  0  0  0  0  0  1  1  0 34]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        36
           1       0.97      0.97      0.97        36
           2       1.00      1.00      1.00        35
           3       1.00      1.00      1.00        37
           4       0.95      0.97      0.96        36
           5       0.97      1.00      0.99        37
           6       0.97      1.00      0.99        36
           7       0.95      0.97      0.96        36
           8       1.00      0.94      0.97        35
           9       1.00      0.94      0.97        36

    accuracy                           0.98       360
   macro avg       0.98      0.98      0.98       360
weighted avg       0.98      0.98      0.98       360
```

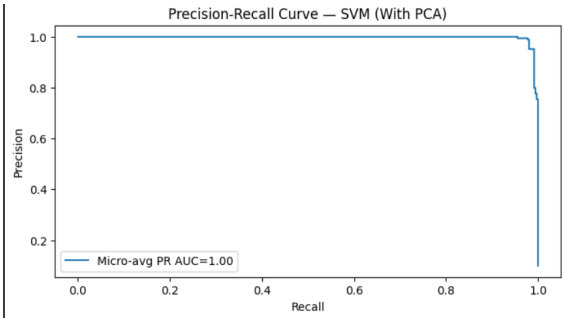
(b) SVM Without PCA



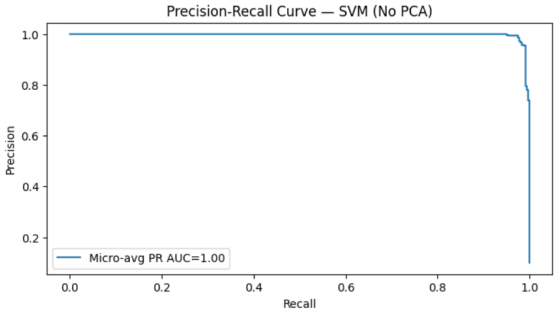
(a) ROC of SVM with PCA



(b) ROC of SVM without PCA



(a) Precision recall Curve of SVM with PCA



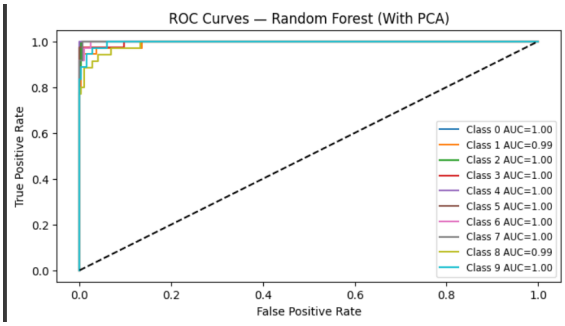
(b) Precision recall Curve of SVM without PCA

```
--- Random Forest (With PCA) ---
Confusion Matrix:
 [[35  0  0  0  0  0  1  0  0  0]
 [ 0 34  0  0  0  2  0  0  0  0]
 [ 0  0 35  0  0  0  0  0  0  0]
 [ 0  0  1 36  0  0  0  0  0  0]
 [ 0  0  0  0 35  0  0  1  0  0]
 [ 0  0  0  0  0 36  0  0  0  1]
 [ 0  1  0  0  0  0 34  0  1  0]
 [ 0  0  0  0  0  0  0 36  0  0]
 [ 0  3  1  0  0  1  0  1 28  1]
 [ 0  0  0  0  0  1  1  0  1 33]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99        36
           1       0.89      0.94      0.92        36
           2       0.95      1.00      0.97        35
           3       1.00      0.97      0.99        37
           4       1.00      0.97      0.99        36
           5       0.90      0.97      0.94        37
           6       0.94      0.94      0.94        36
           7       0.95      1.00      0.97        36
           8       0.93      0.80      0.86        35
           9       0.94      0.92      0.93        36

    accuracy                           0.95       360
   macro avg       0.95      0.95      0.95       360
weighted avg       0.95      0.95      0.95       360
```

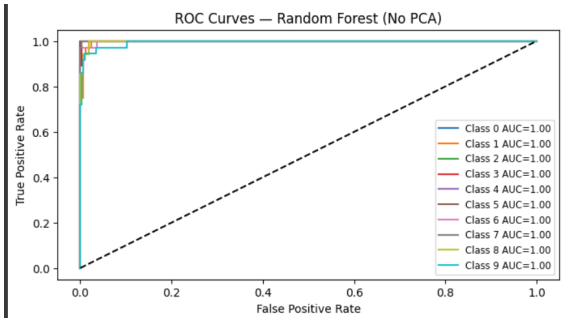
(a) Random Forest With PCA

```
--- Random Forest (No PCA) ---
Confusion Matrix:
 [[35  0  0  0  1  0  0  0  0  0]
 [ 0 35  0  0  0  1  0  0  0  0]
 [ 1  0 34  0  0  0  0  0  0  0]
 [ 0  0  0 36  0  0  0  0  0  1]
 [ 0  1  0  0 35  0  0  0  0  0]
 [ 0  0  0  0  0 37  0  0  0  0]
 [ 0  0  0  0  0  0 35  0  1  0]
 [ 0  0  0  0  0  0  0 36  0  0]
 [ 0  3  0  0  0  0  0  2 30  0]
 [ 0  0  0  1  0  0  0  1  1 33]]
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        36
           1       0.90      0.97      0.93        36
           2       1.00      0.97      0.99        35
           3       0.97      0.97      0.97        37
           4       0.97      0.97      0.97        36
           5       0.97      1.00      0.99        37
           6       1.00      0.97      0.99        36
           7       0.92      1.00      0.96        36
           8       0.94      0.86      0.90        35
           9       0.97      0.92      0.94        36

    accuracy                           0.96       360
   macro avg       0.96      0.96      0.96       360
weighted avg       0.96      0.96      0.96       360
```

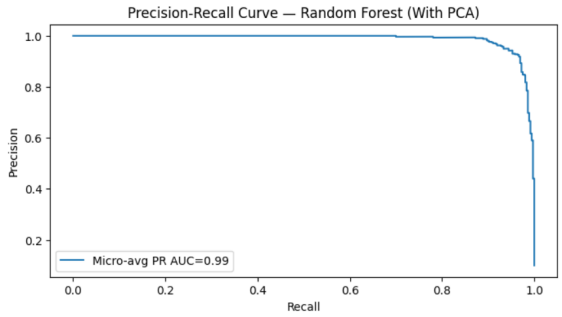
(b) Random Forest without PCA



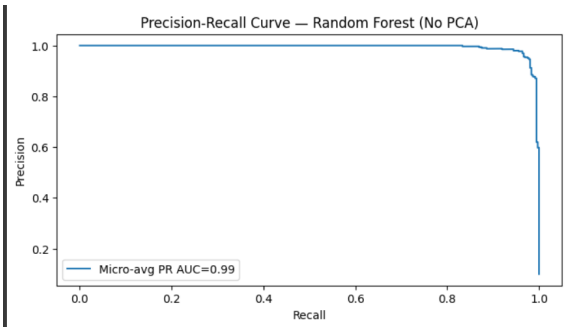
(a) ROC of Random Forest With PCA



(b) ROC of Random Forest without PCA



(a) Precision recall Curve of Random Forest With PCA



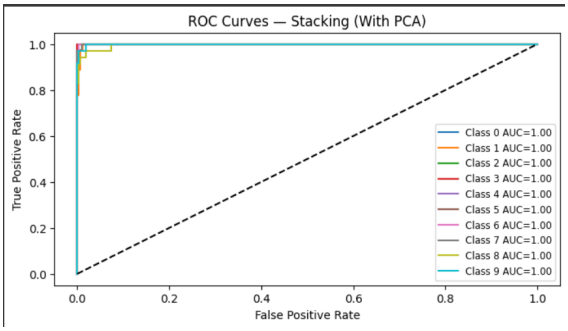
(b) Precision recall Curve of Random Forest without PCA

```
--- Stacking (With PCA) ---
Confusion Matrix:
 [[36  0  0  0  0  0  0  0  0  0]
 [ 0 35  1  0  0  0  0  0  0  0]
 [ 0  0 35  0  0  0  0  0  0  0]
 [ 0  0  1 36  0  0  0  0  0  0]
 [ 0  0  0  0 35  0  0  1  0  0]
 [ 0  0  0  0  0 36  0  0  0  1]
 [ 0  0  0  0  0  0 35  0  1  0]
 [ 0  0  0  0  0  1  0 35  0  0]
 [ 0  2  1  0  0  0  0  0 32  0]
 [ 0  0  0  0  0  0  1  0  0 35]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        36
           1       0.95      0.97      0.96        36
           2       0.92      1.00      0.96        35
           3       1.00      0.97      0.99        37
           4       1.00      0.97      0.99        36
           5       0.97      0.97      0.97        37
           6       0.97      0.97      0.97        36
           7       0.97      0.97      0.97        36
           8       0.97      0.91      0.94        35
           9       0.97      0.97      0.97        36

    accuracy                           0.97       360
   macro avg       0.97      0.97      0.97       360
weighted avg       0.97      0.97      0.97       360
```

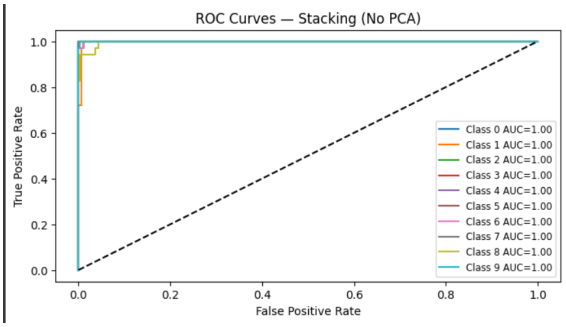
(a) Stacking With PCA

```
--- Stacking (No PCA) ---
Confusion Matrix:
 [[36  0  0  0  0  0  0  0  0  0]
 [ 0 35  0  0  0  1  0  0  0  0]
 [ 0  0 35  0  0  0  0  0  0  0]
 [ 0  0  0 37  0  0  0  0  0  0]
 [ 0  0  0  0 36  0  0  0  0  0]
 [ 0  0  0  0  0 37  0  0  0  0]
 [ 0  0  0  0  0  0 35  0  1  0]
 [ 0  0  0  0  0  0  0 36  0  0]
 [ 0  2  0  0  0  0  0  0 33  0]
 [ 0  0  0  0  1  0  0  0  0 35]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        36
           1       0.95      0.97      0.96        36
           2       1.00      1.00      1.00        35
           3       1.00      1.00      1.00        37
           4       0.97      1.00      0.99        36
           5       0.97      1.00      0.99        37
           6       1.00      0.97      0.99        36
           7       1.00      1.00      1.00        36
           8       0.97      0.94      0.96        35
           9       1.00      0.97      0.99        36

    accuracy                           0.99       360
   macro avg       0.99      0.99      0.99       360
weighted avg       0.99      0.99      0.99       360
```

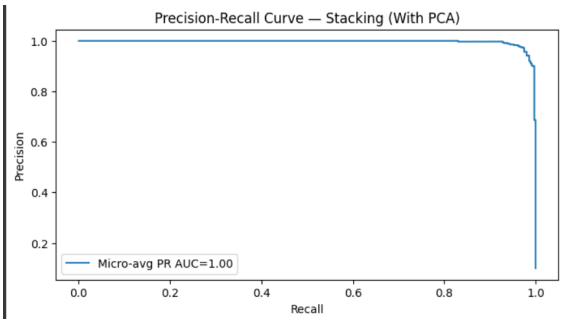
(b) Stacking without PCA



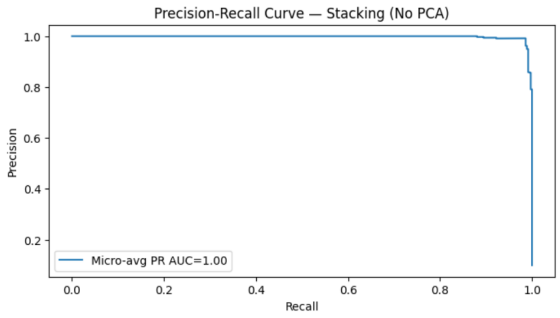
(a) ROC of Stacking With PCA



(b) ROC of Stacking without PCA



(a) Precision recall Curve of Stacking With PCA



(b) Precision recall Curve of Stacking without PCA

8 Tabulation

Table 1: Comparison of Model Performance With and Without PCA

Model	Without PCA - Avg Accuracy	Without PCA - Avg F1	With PCA - Avg Accuracy
AdaBoost	0.809124	0.809378	0.789104
Decision Tree	0.852512	0.852498	0.816370
Gradient Boosting	0.963827	0.963873	0.936011
KNN	0.978295	0.978189	0.977177
Logistic Regression	0.968836	0.968862	0.964941
Naive Bayes	0.829723	0.829009	0.905949
Random Forest	0.976628	0.976581	0.962711
SVM	0.983302	0.983333	0.981634
Stacking	0.982187	0.982167	0.974393
XGBoost	0.957707	0.957735	0.941007

Table 2: PCA Variance Explained

Setting / Variance Target	Explained Variance (%)	Justification
With PCA (95%)	95	Retain 95% of the variance in the data

Table 3: SVM — Hyperparameter Tuning Results

Kernel	C Values Tried	Gamma Values Tried	Performance (No-PCA)	Performance (With-PCA)
linear	0.1, 1, 10	scale, auto	Acc: 0.9816, F1: 0.9816	Acc: 0.9827, F1: 0.9828
rbf	0.1, 1, 10	scale, auto	–	–

Table 4: Naive Bayes — Smoothing Choices

Smoothing Parameter	Performance (No-PCA)	Performance (With-PCA)
Default	Acc: 0.7830, F1: 0.7802	Acc: 0.9059, F1: 0.9059

Table 5: KNN — Hyperparameter Tuning

k Values	Weights	Distance Metrics	Performance (No-PCA)	Performance (With-PCA)
3, 5, 7	uniform, distance	euclidean, manhattan	Acc: 0.9794, F1: 0.9793	Acc: 0.9766, F1: 0.9765

Table 6: Logistic Regression — Hyperparameter Tuning

C Values	Solver	Performance (No-PCA)	Performance (With-PCA)
0.01, 0.1, 1, 10	liblinear, lbfgs	Acc: 0.9694, F1: 0.9693	Acc: 0.9694, F1: 0.9693

Table 7: Decision Tree — Hyperparameter Tuning

Max Depth	Min Samples Split	Performance (No-PCA)	Performance (With-PCA)
None, 5, 10	2, 5, 10	Acc: 0.8525, F1: 0.8525	Acc: 0.8192, F1: 0.8185

Table 8: Random Forest — Hyperparameter Tuning

N Estimators	Max Depth	Performance (No-PCA)	Performance (With-PCA)
50, 100	None, 5, 10	Acc: 0.9733, F1: 0.9732	Acc: 0.9521, F1: 0.9521

Table 9: AdaBoost — Hyperparameter Tuning

N Estimators	Learning Rate	Performance (No-PCA)	Performance (With-PCA)
50, 100	0.01, 0.1, 1	Acc: 0.8091, F1: 0.8094	Acc: 0.7891, F1: 0.7929

Table 10: Gradient Boosting — Hyperparameter Tuning

N Estimators	Learning Rate	Max Depth	Performance (No-PCA)	Performance (With-PCA)
50, 100	0.01, 0.1, 1	3, 5	Acc: 0.9638, F1: 0.9639	Acc: 0.9349, F1: 0.9351

Table 11: XGBoost — Hyperparameter Tuning

N Estimators	Learning Rate	Max Depth	Performance (No-PCA)	Performance (With-PCA)
50, 100	0.01, 0.1, 0.2	3, 5, 7	Acc: 0.9638, F1: 0.9639	Acc: 0.9521, F1: 0.9519

Table 12: Stacking — Base Estimators and Final Estimator

Base Estimators	Final Estimator	Performance (No-PCA)	Performance (With-PCA)
RandomForestClassifier, GradientBoostingClassifier, KNeighborsClassifier	LogisticRegression	Acc: 0.9827, F1: 0.9828	Acc: 0.9749, F1: 0.9749

9 Learning Outcomes

From this assignment,We can able to

- Understand the concept and application of PCA for dimensionality reduction and how it helps in simplifying high-dimensional datasets while retaining maximum variance.
- Gain hands-on experience in training and evaluating multiple machine learning models both with and without PCA to analyze their comparative performance.
- Develop skills in hyperparameter tuning and cross-validation to optimize models and ensure reliable, unbiased evaluation results.
- Interpret and analyze the impact of PCA on accuracy, F1-score, and model stability, drawing conclusions about when dimensionality reduction is beneficial.