Dimensionality Reduction

Principle Component Analysis

PCA (Principal Component Analysis) is a **dimensionality reduction technique** used in machine learning and data analysis.
It helps simplify large datasets by **reducing the number of features** (by finding linear combinations of features) while **keeping most of the important information**.

```
def pcaFeature(indep_X, dep_Y, n):
    pca = PCA(n_components=n)
    fit = pca.fit(indep_X)
    pca_features = fit.transform(indep_X)
    return pca_features
```

- pca = PCA(n_components=n) **- n principal components** (the most important features).
- PCA will compress the data into n new dimensions while retaining most of the important information.
- fit = pca.fit(indep_X) - PCA **learn the structure of the data**
- pca_features = fit.transform(indep_X) **- transforms the original data** into new PCA features (the principal components).

```
result
#3
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
PCA	0.82	0.83	0.81	0.8	0.77	0.87	0.84

```
result
#4
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
PCA	0.83	0.8	0.84	0.81	0.77	0.87	0.85

```
result
#5
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
PCA	0.83	0.83	0.87	0.87	0.81	0.86	0.86

```
result
#6
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
PCA	0.86	0.87	0.91	0.88	0.83	0.87	0.88

Summary:

- As the number of PCA components increases from **3 to 6**, the accuracy of all classifiers consistently improves.
- **SVM (non-linear)** shows the highest accuracy with 6 components.
- The best results are achieved with **6 components.**

Linear Discriminent Analysis

LDA (Linear Discriminant Analysis) is a supervised dimensionality reduction technique.

It reduces the number of features using class labels (target variable) to keep the most important information for classification.

```
def ldaFeature(indep_X, dep_Y, n):
    lda = LDA(n_components=n)
    fit = lda.fit(indep_X, dep_Y)
    lda_features = fit.transform(indep_X)
    return lda_features
```

- lda = LDA(n_components=n) - how many new features (components) to be extracted
- fit = lda.fit(indep_X, dep_Y) – learn how to separate the data based on dep_Y.

```python
lda_data = ldaFeature(indep_X, dep_Y, 1)
```

If our target variable dep_Y has:

- **2 classes** (e.g. yes/no or 0/1),
  → then n_classes - 1 = 1
  ✅ So **n_components can only be 1**.

If we try to set n=3, we will get error. So here we are giving only one value.

```python
result
#1
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
LDA	0.98	0.98	0.98	0.98	0.98	0.99	0.99

Summary:

- All models performed exceptionally well after applying **LDA**, achieving **98–99%** accuracy.
- **Decision Tree and Random Forest** shows **0.99** accuracy, making them the top-performing classifiers.
- This means that **LDA successfully simplified the data** by reducing the number of features, **while still keeping the important differences between the classes clear and easy to separate**.

Kernel Principal Component Analysis

Kernel Principal Component Analysis **(Kernel PCA)** is an extension of standard Principal Component Analysis (PCA) that uses kernel methods to perform **non-linear dimensionality reduction**.

Traditional PCA can only capture linear relationships in data, Kernel PCA can capture complex, non-linear patterns.

```python
def kpcaFeature(indep_X, dep_Y, n):
    kpca = KernelPCA(n_components=n, kernel='rbf')
    kpca_features = kpca.fit_transform(indep_X)
    return kpca_features
```

- n_components=n: Specifies how many principal components you want to keep.
- kernel='rbf': Chooses the Radial Basis Function (Gaussian kernel) for non-linear mapping.

```
result
#3
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
KPCA	0.64	0.64	0.64	0.58	0.39	0.68	0.68

```
result
#4
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
KPCA	0.64	0.64	0.64	0.51	0.39	0.68	0.67

```
result
#5
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
KPCA	0.64	0.64	0.64	0.53	0.39	0.68	0.65

```
result
#6
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
KPCA	0.64	0.64	0.64	0.58	0.39	0.69	0.68

Summary:

- **Logistic Regression, SVM** (linear and nonlinear), and **Decision Tree** models have consistent scores around 0.64–0.69 across runs.
- **Naive Bayes and KNN** performed very low.
- **Decision Tree and Random Forest** shows the highest performance reaching up to **0.68 or 0.69.**