**EXPERIMENT NO. 2**

1. **A.1  AIM: -** Handwritten Digit Recognition System using PCA
2. **A.2  Prerequisite**

• Different programming language (Python or Java), Understanding of Machine Learning Algorithms, Machine Learning Algorithms

1. **A.3  Outcome**

After successful completion of this experiment students will be able to understand working of Convolutional Neural Networks (CNN) and apply this algorithm wherever required

1. **A.4  Theory**

Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the Principal Components. It is one of the popular tools that is used for exploratory data analysis and predictive modeling. It is a technique to draw strong patterns from the given dataset by reducing the variances.

PCA generally tries to find the lower-dimensional surface to project the high-dimensional data.

PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real- world applications of PCA are image processing, movie recommendation system, optimizing the power allocation in various communication channels. It is a feature extraction technique, so it contains the important variables and drops the least important variable.

**HOW DO YOU DO A PRINCIPAL COMPONENT ANALYSIS?**

1. Standardize the range of continuous initial variables
2. Compute the covariance matrix to identify correlations
3. Compute the eigenvectors and eigenvalues of the covariance matrix to identify the principal components
4. Create a feature vector to decide which principal components to keep
5. Recast the data along the principal components axes

**Steps for PCA algorithm**

1. **Getting the dataset**Firstly, we need to take the input dataset and divide it into two subparts X and Y, where X is the training set, and Y is the validation set.
2. **Representing data into a structure**Now we will represent our dataset into a structure. Such as we will represent the two- dimensional matrix of independent variable X. Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.
3. **Standardizing the data**In this step, we will standardize our dataset. Such as in a particular column, the features with high variance are more important compared to the features with lower variance.  
   If the importance of features is independent of the variance of the feature, then we will divide each data item in a column with the standard deviation of the column. Here we will name the matrix as Z.
4. **Calculating the Covariance of Z**To calculate the covariance of Z, we will take the matrix Z, and will transpose it. After transpose, we will multiply it by Z. The output matrix will be the Covariance matrix of Z.
5. **Calculating the Eigen Values and Eigen Vectors**Now we need to calculate the eigenvalues and eigenvectors for the resultant covariance matrix Z. Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the eigenvalues.
6. **Sorting the Eigen Vectors**In this step, we will take all the eigenvalues and will sort them in decreasing order, which means from largest to smallest. And simultaneously sort the eigenvectors accordingly in matrix P of eigenvalues. The resultant matrix will be named as P\*.
7. **Calculating the new features Or Principal Components**Here we will calculate the new features. To do this, we will multiply the P\* matrix to the Z. In the resultant matrix Z\*, each observation is the linear combination of original features. Each column of the Z\* matrix is independent of each other.
8. **Remove less or unimportant features from the new dataset.**The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

**A5. Task  
Given the MNIST data set your goal is to correctly identify digits from a dataset of tens of thousands of handwritten images. Perform Handwriting detection using PCA.**

**Link: http://yann.lecun.com/exdb/mnist/  
Or  
Link: https://www.kaggle.com/competitions/digit-recognizer**

**Note: Assume necessary Details. Use Exploratory Data Analysis and show details.  
You can use any technique for pre-processing if required.**

PART B

|  |  |
| --- | --- |
| Roll No. C009 | Name: Samarth Borade |
| Class : BTI SEM 10 | Batch : EB1 |
| Date of Experiment: 05/01/24 | Date of Submission |
| Grade : |  |

**B.1 Software Code written by student:**

**Data Pre-Processing:**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

train\_df = pd.read\_csv('train.csv')

test\_df = pd.read\_csv('test.csv')

sample\_train\_df = train\_df.sample(*frac*=0.01, *random\_state*=42)

sample\_train\_df.shape

X = sample\_train\_df.drop('label', *axis*=1)

y = sample\_train\_df['label']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, *test\_size*=0.3, *random\_state*=100)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

**PCA :**

# Assuming X is your dataset

# Mean

X\_mean = X\_train.mean()

# Standard deviation

X\_std = X\_train.std()

# Standardization

Z = (X\_train - X\_mean) / X\_std

# Covariance matrix

c = np.cov(Z, *rowvar*=False)

# Calculate eigenvalues and eigenvectors

eigenvalues, eigenvectors = np.linalg.eig(c)

# Sort eigenvalues in descending order

sorted\_indices = np.argsort(eigenvalues)[::-1]

sorted\_eigenvalues = eigenvalues[sorted\_indices]

# Calculate explained variance ratio

explained\_variance\_ratio = sorted\_eigenvalues / np.sum(sorted\_eigenvalues)

# Calculate cumulative explained variance

cumulative\_variance = np.cumsum(explained\_variance\_ratio)

# Set desired variance threshold

desired\_variance\_retained = 0.95

n\_components = np.argmax(cumulative\_variance >= desired\_variance\_retained) + 1

print("Number of components to retain {}% variance: {}".format(desired\_variance\_retained \* 100, n\_components))

**🡪** Number of components to retain 95.0% variance: 115

from sklearn.decomposition import PCA

# Initialize PCA

pca = PCA(*n\_components*=115) # You can set the explained variance threshold here

# Fit PCA on the scaled training data

pca.fit(X\_train)

# Transform both the training and test sets

X\_train\_pca = pca.transform(X\_train)

X\_test\_pca = pca.transform(X\_test)

selected\_features = pca.components\_

# Print the selected features

print("Selected Features by PCA:")

for i, component in enumerate(selected\_features):

print(f"Principal Component {i+1}:")

for j, feature in enumerate(X.columns):

print(f"Feature {feature}: {component[j]}")

print()

****

svm\_model = SVC()

# Define parameters for grid search

param\_grid = {

'C': [0.1, 1, 10, 100],

'gamma': [0.01, 0.1, 1, 'auto'],

'kernel': ['linear', 'rbf', 'poly', 'sigmoid']

}

grid\_search = GridSearchCV(svm\_model, param\_grid, *cv*=5)

grid\_search.fit(X\_train\_pca, y\_train)

# Get best parameters

best\_params = grid\_search.best\_params\_

print("Best Parameters:", best\_params)

🡪 Best Parameters: {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}

best\_model\_SVM = grid\_search.best\_estimator\_

y\_pred\_SVM = best\_model\_SVM.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred\_SVM)

print("Accuracy:", accuracy \* 100)

🡪 Accuracy: 85.714

****

from sklearn.tree import DecisionTreeClassifier

tree\_model = DecisionTreeClassifier()

# Define parameters for grid search

param\_grid = {

'criterion': ['gini', 'entropy'],

'max\_depth': [None, 10, 20, 30, 40, 50],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4]

}

grid\_search = GridSearchCV(tree\_model, param\_grid, *cv*=5)

grid\_search.fit(X\_train, y\_train)

best\_params = grid\_search.best\_params\_

print("Best Parameters:", best\_params)

🡪Best Parameters: {'criterion': 'gini', 'max\_depth': None, 'min\_samples\_leaf': 1, 'min\_samples\_split': 5}

best\_model\_dt = grid\_search.best\_estimator\_

y\_pred\_dt = best\_model\_dt.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred\_dt)

print("Accuracy:", accuracy \* 100)

🡪 Accuracy: 42.85

****

from sklearn.ensemble import RandomForestClassifier

forest\_model = RandomForestClassifier()

param\_grid = {

'n\_estimators': [100, 300, 500],

'max\_depth': [None, 10, 20, 30],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4],

'bootstrap': [True, False]

}

grid\_search = GridSearchCV(forest\_model, param\_grid, *cv*=5)

grid\_search.fit(X\_train, y\_train)

best\_params = grid\_search.best\_params\_

print("Best Parameters:", best\_params)

🡪Best Parameters: {'bootstrap': False, 'max\_depth': 10, 'min\_samples\_leaf': 1, 'min\_samples\_split': 2, 'n\_estimators': 300}

best\_model\_rt = grid\_search.best\_estimator\_

y\_pred\_rt = best\_model\_rt.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred\_rt)

print("Accuracy:", accuracy \* 100)

🡪 Accuracy: 74.64

****

from sklearn.neighbors import KNeighborsClassifier

knn\_model = KNeighborsClassifier()

# Define parameters for grid search

param\_grid = {

'n\_neighbors': [3, 5, 7, 9], # Test different values for K

'weights': ['uniform', 'distance'],

'metric': ['euclidean', 'manhattan'] # Different distance metrics

}

grid\_search = GridSearchCV(knn\_model, param\_grid, *cv*=5)

grid\_search.fit(X\_train, y\_train)

best\_params = grid\_search.best\_params\_

print("Best Parameters:", best\_params)

**🡪** Best Parameters: {'metric': 'euclidean', 'n\_neighbors': 3, 'weights': 'distance'}

best\_model\_knn = grid\_search.best\_estimator\_

y\_pred\_knn = best\_model\_knn.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred\_knn)

print("Accuracy:", accuracy \* 100)

🡪 Accuracy: 68.25



import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from keras.models import Sequential

from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D

from keras.utils import to\_categorical

from scikeras.wrappers import KerasClassifier

from sklearn.model\_selection import GridSearchCV

train\_df = pd.read\_csv('train.csv')

test\_df = pd.read\_csv('test.csv')

# Sample 10% of your training data for testing code functionality

sample\_train\_df = train\_df.sample(*frac*=0.01, *random\_state*=42)

X = sample\_train\_df.drop('label', *axis*=1)

y = sample\_train\_df['label']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, *test\_size*=0.3, *random\_state*=100)

# Preprocess the data

X\_train\_processed = X\_train\_pca.reshape(-1, 115, 1).astype('float32') / 255 # Adjusting the shape here

X\_test\_processed = X\_test\_pca.reshape(-1, 115, 1).astype('float32') / 255

y\_train\_processed = to\_categorical(y\_train)

y\_test\_processed = to\_categorical(y\_test)

from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense

# Define a function to create the CNN model

def create\_model(*learning\_rate*=0.001):

model = Sequential()

model.add(Conv1D(32, *kernel\_size*=3, *activation*='relu', *input\_shape*=(115, 1))) # Using Conv1D for 1D convolution

model.add(MaxPooling1D(*pool\_size*=2))

model.add(Flatten())

model.add(Dense(128, *activation*='relu'))

model.add(Dense(10, *activation*='softmax'))

# Compile the model with the given learning rate

model.compile(*optimizer*='adam', *loss*='categorical\_crossentropy', *metrics*=['accuracy'])

return model

model = KerasClassifier(*build\_fn*=create\_model, *learning\_rate*=0.001, *epochs*=10, *batch\_size*=64, *verbose*=0)

param\_grid = {'learning\_rate': [0.001, 0.01, 0.1],

'batch\_size': [32, 64, 128]}

grid = GridSearchCV(*estimator*=model, *param\_grid*=param\_grid, *cv*=3)

grid\_result = grid.fit(X\_train\_processed, y\_train\_processed)

print("Best Parameters: ", grid\_result.best\_params\_)

🡪 Best Parameters: {'batch\_size': 32, 'learning\_rate': 0.01}

best\_model = grid\_result.best\_estimator\_

y\_pred\_train = best\_model.predict(X\_train\_processed)

# Convert one-hot encoded y\_pred\_train to categorical labels

y\_pred\_categorical = np.argmax(y\_pred\_train, *axis*=1)

# Convert y\_train\_processed back to categorical labels

y\_train\_categorical = np.argmax(y\_train\_processed, *axis*=1)

train\_accuracy = np.mean(y\_pred\_categorical == y\_train\_categorical)

print(f"Train Accuracy: {train\_accuracy \* 100:.2f}%")

🡪 Train Accuracy: 98.98%

# Use the best model to make predictions on test data

y\_pred\_test = best\_model.predict(X\_test\_processed)

# Convert one-hot encoded labels to categorical labels

y\_test\_categorical = np.argmax(y\_test\_processed, *axis*=1)

# Calculate accuracy manually

test\_accuracy = np.mean(np.argmax(y\_pred\_test, *axis*=1) == y\_test\_categorical)

print(f"Test Accuracy: {test\_accuracy \* 100:.2f}%")

🡪 Test Accuracy: 73.13%

Observation:

| **Model** | **Accuracy** | **Training Time** | **Hyperparameters** | **Model Complexity** | **Prediction Time** |
| --- | --- | --- | --- | --- | --- |
| SVM | 0.85 | 1.7s | {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'} | Moderate | Low |
| Decision Tree | 0.42 | 15.7s | {'criterion': 'gini', 'max\_depth': 10, 'min\_samples\_leaf': 1, 'min\_samples\_split': 2} | Low | Low |
| Random Forest | 0.74 | 10mins 41s | {'bootstrap': False, 'max\_depth': 10, 'min\_samples\_leaf': 1, 'min\_samples\_split': 2, 'n\_estimators': 500} | High | Moderate |
| K-Nearest Neighbors | 0.68 | 6s | {'metric': 'euclidean', 'n\_neighbors': 3, 'weights': 'distance'} | Low | Moderate |
| CNN | 0.98 | 41s | {'batch\_size': 32, 'learning\_rate': 0.01} | High | High |

**Variance Retention Method:**

* Explained Variance Ratio: Calculate the variance each component explains. Choose components until a desired total variance is reached (e.g., 95%).

**1) Standardization:**

* Computes mean and standard deviation, standardizes the data.

**2) Covariance Matrix:**

* Calculates the covariance matrix.

**3) Eigenvalues and Eigenvectors:**

* Obtains eigenvalues and eigenvectors of the covariance matrix.

**4) Explained Variance Ratio:**

* Determines the variance explained by each component.

**5) Cumulative Explained Variance:**

* Computes cumulative explained variance.

**6) Threshold Selection:**

* Sets a desired variance retention threshold (95% in this case).
* Determines the number of components required to retain at least this threshold.

**Conclusion:**

I compared to previous experiment , after applying PCA I got increased accuracy in SVM , K-nearest and CNN . I shall make changes to the code and check on the optimization part because even after using GridSearchCV I feel the accuracy is comparatively low.