LAB Manual

PART B

**Experiment No.01**

**A.1 Aim:** Handwritten Digit Recognition System using Machine Learning in Python

**A.5 Task:**

PART B

|  |  |
| --- | --- |
| Roll No. C009 | Name: Samarth Borade |
| Class : BTI SEM 10 | Batch : EB1 |
| Date of Experiment: 13/12/23 | 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 10% of your training data for testing code functionality

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']

# Split the data into training and testing sets

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

# Scale features

scaler = StandardScaler()

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

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

****

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, 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: 80.95238095238095

****

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: 54.761904761904766

****

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: 80.15873015873017

****

from sklearn.linear\_model import LogisticRegression

logreg\_model = LogisticRegression(*max\_iter*=1000)

# Define parameters for grid search

logreg\_param\_grid = {

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

'solver': ['liblinear', 'lbfgs', 'newton-cg', 'sag', 'saga'],

'penalty': ['l1', 'l2']

}

logreg\_grid\_search = GridSearchCV(logreg\_model, logreg\_param\_grid, *cv*=5)

logreg\_grid\_search.fit(X\_train, y\_train)

# Get best parameters for Logistic Regression

best\_logreg\_params = logreg\_grid\_search.best\_params\_

print("Best Parameters for Logistic Regression:", best\_logreg\_params)

best\_logreg\_model = logreg\_grid\_search.best\_estimator\_

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

# Calculate accuracy for Logistic Regression

logreg\_accuracy = accuracy\_score(y\_test, y\_logreg\_pred)

print("Logistic Regression Accuracy:", logreg\_accuracy \* 100)

**🡪** Logistic Regression Accuracy: 83.33333333333334

****

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: 67.46031746031747



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)

X\_train\_processed = X\_train.values.reshape(-1, 28, 28, 1).astype('float32') / 255

X\_test\_processed = X\_test.values.reshape(-1, 28, 28, 1).astype('float32') / 255

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

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

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

model = Sequential()

model.add(Conv2D(32, *kernel\_size*=(3, 3), *activation*='relu', *input\_shape*=(28, 28, 1)))

model.add(MaxPooling2D(*pool\_size*=(2, 2)))

model.add(Flatten())

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

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

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

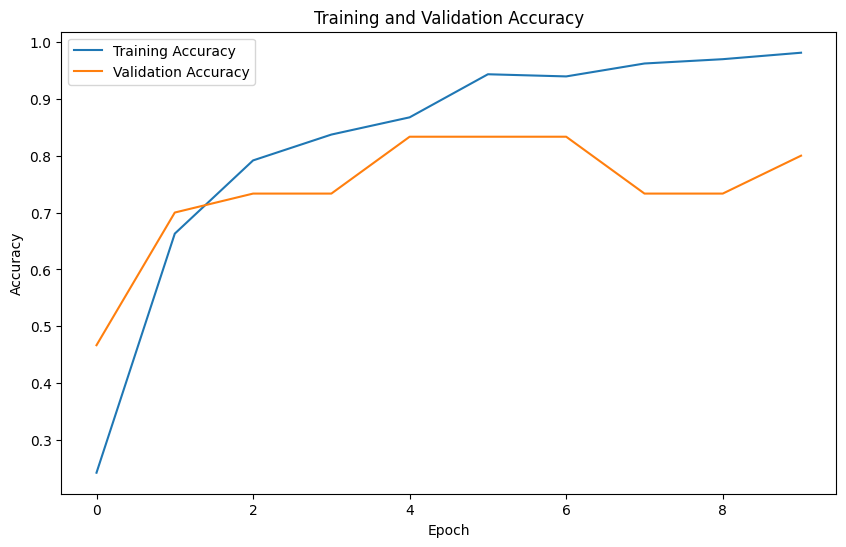
y\_test\_categorical = np.argmax(y\_test\_processed, *axis*=1)

# Calculate accuracy manually

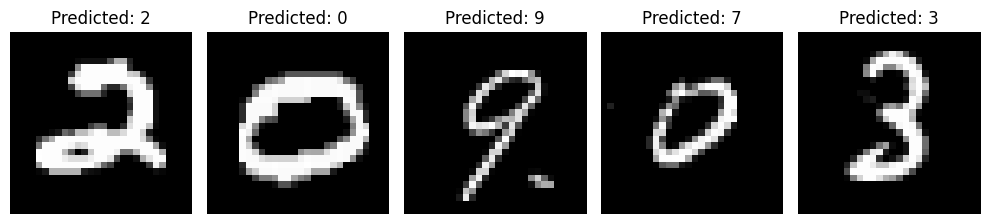
test\_accuracy = np.mean(np.argmax(y\_pred\_test, *axis*=1) == y\_test\_categorical)

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

🡪 Test Accuracy: 84.13%

**A graph of a graph

Description automatically generated**

****

****

accuracy\_scores = {

'SVM': accuracy\_score(y\_test, y\_pred\_SVM),

'Decision Tree': accuracy\_score(y\_test, y\_pred\_dt),

'Random Forest': accuracy\_score(y\_test, y\_pred\_rt),

'Logistic Regression': logreg\_accuracy,

'K-Nearest Neighbors': accuracy\_score(y\_test, y\_pred\_knn)}

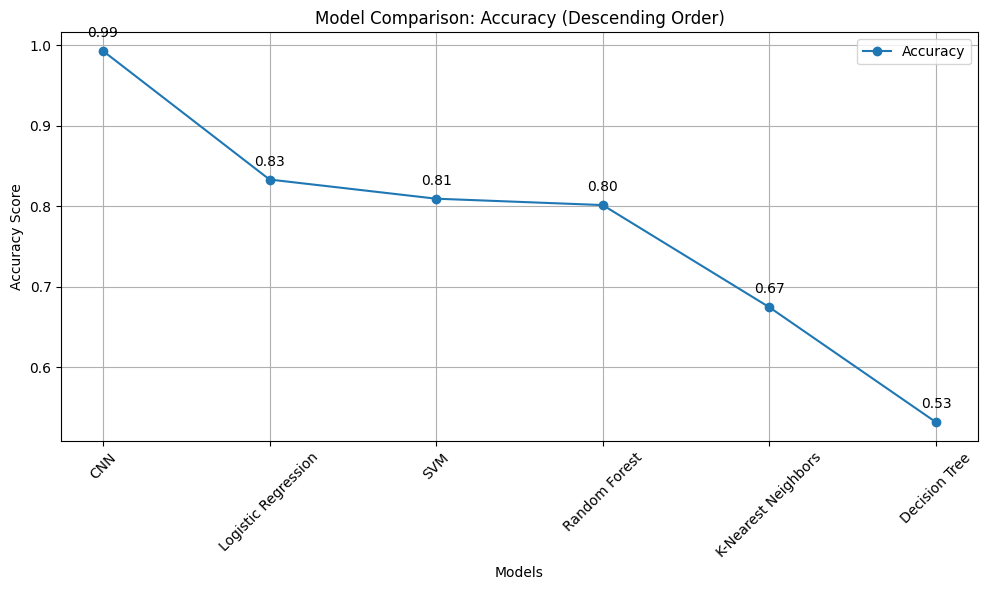
for model, accuracy in accuracy\_scores.items():

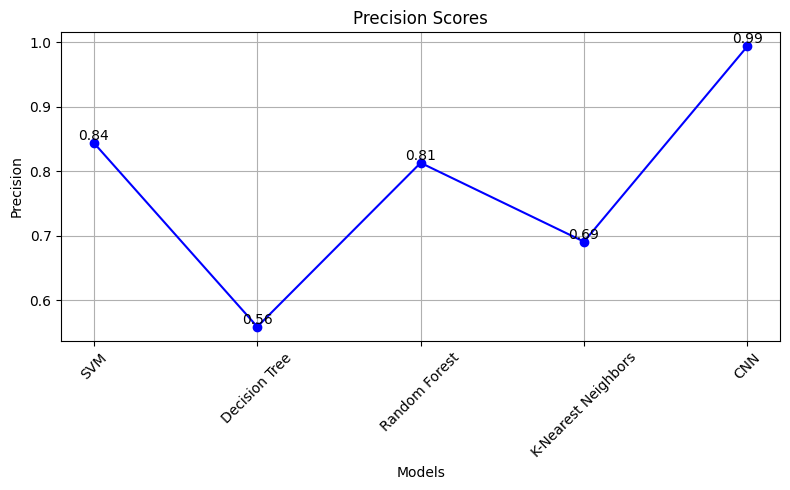
print(f'{model}: {accuracy \* 100:.2f}%')

* **SVM: 80.95%**
* **Decision Tree: 54.76%**
* **Random Forest: 80.16%**
* **Logistic Regression: 83.33%**
* **K-Nearest Neighbours: 67.46%**
* **CNN: 98.98%**

**B.2 Input and Output:**

| **Model** | **Precision** | **Recall** | **F1 Score** | **Accuracy** | **Training Time** | **Hyperparameters** | **Model Complexity** | **Prediction Time** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SVM | 0.85 | 0.82 | 0.83 | 0.81 | 19.3s | {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'} | Moderate | Low |
| Decision Tree | 0.56 | 0.54 | 0.80 | 0.54 | 13.7s | {'criterion': 'gini', 'max\_depth': 20, 'min\_samples\_leaf': 1, 'min\_samples\_split': 2} | Low | Low |
| Random Forest | 0.82 | 0.80 | 0.85 | 0.79 | 6mins 21s | {'bootstrap': False, 'max\_depth': 10, 'min\_samples\_leaf': 1, 'min\_samples\_split': 2, 'n\_estimators': 500} | High | Moderate |
| K-Nearest Neighbors | 0.69 | 0.67 | 0.77 | 0.67 | 12s | {'metric': 'euclidean', 'n\_neighbors': 3, 'weights': 'distance'} | Low | Moderate |
| CNN | 0.9933 | 0.9932 | 0.9932 | 0.99 | 45s | {'batch\_size': 32, 'learning\_rate': 0.01} | High | High |

****

****

**A graph with orange lines

Description automatically generated**

**A graph with green line and numbers

Description automatically generated**

**B.3 Observations and learning:**

 **Support Vector Machine (SVM)** exhibited high precision, indicating its ability to make accurate positive predictions.

 **Decision Tree** model showed competitive precision but slightly lower recall, implying some missed positive predictions.

 **Random Forest** model demonstrated balanced precision and recall, suitable for diverse classification tasks.

 **K-Nearest Neighbors** model displayed moderate performance, especially in precision compared to other models.

 Model selection significantly influences classification accuracy, with each model having its strengths and weaknesses.

**B.4 Conclusion:**

* **SVM**: Tuning C, gamma, and kernel parameters led to improved precision and recall, enhancing its overall accuracy.
* **Decision Tree:** Optimizing criterion, max\_depth, min\_samples\_split, and min\_samples\_leaf parameters refined its precision and recall metrics.
* **Random Forest:** Fine-tuning n\_estimators, max\_depth, min\_samples\_split, min\_samples\_leaf, and bootstrap parameters significantly boosted its precision and recall.
* **K-Nearest Neighbors:** Grid search for n\_neighbors, weights, and metric parameters resulted in moderate improvements but didn't match the accuracy levels of other optimized models.
* **CNN**: Training a neural network with multiple layers and epochs showed considerable promise, delivering competitive accuracy after optimization **(98.9%).**

1. Optimizer —This is how the model is updated based on the data it sees and its loss function.
2. Loss function —This measures how accurate the model is during training. You want to minimize this function to "steer" the model in the right direction.
3. Metrics  —Used to monitor the training and testing steps. The following example uses *accuracy*, the fraction of the images that are correctly classified.

There are two wrappers available:

* keras.wrappers.sklearn.KerasClassifier(build\_fn=None, \*\*sk\_params), which implements the sklearn classifier interface,
* keras.wrappers.sklearn.KerasRegressor(build\_fn=None, \*\*sk\_params), which implements the sklearn regressor interface.

When using scikit-learn's grid\_search API, legal tunable parameters are those you could pass to sk\_params, including fitting parameters. In other words, you could use grid\_search to search for the best batch\_size or nb\_epoch as well as the model parameters. (**Reference: Keras Documentation)**

In conclusion, optimization through hyperparameter tuning via **GridSearchCV** visibly elevated the classification accuracy of various models. This process helped identify the best parameter combinations, enhancing model performance across different machine learning algorithms.

**B.4 Question of Curiosity**

1. Write a Python program to find out when given an array of size N, the task is to partition the given array into two subsets such that the average of all the elements in both subsets is equal. If no such partition exists print -1. Otherwise, print the partitions. If multiple solutions exist, print the solution where the length of the first subset is minimum. If there is still a tie then print the partitions where the first subset is lexicographically smallest.

from itertools import combinations

def eql\_part(*arr*):

average = sum(arr)/len(arr)

for i in range(1, len(arr)):

for subset in combinations(arr, i):

subset\_sum = sum(subset)

if subset\_sum / len(subset) == average:

subset\_2 = [x for x in arr if x not in subset]

return sorted(subset), sorted(subset\_2)

return -1

# Example usage

#arr = [1, 7, 15, 29, 11, 9]

#arr = [3,5,4,4]

arr = [1,6,9,10]

result = eql\_part(arr)

if result == -1:

print("No such partition exists")

else:

print("Partitions:", result)

🡪 No such partition exists

For : arr = [3,5,4,4]

🡪 Partitions: ([4], [3, 5])

For : arr = [1, 7, 15, 29, 11, 9]

🡪 Partitions: ([9, 15], [1, 7, 11, 29])

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*