```
!pip install bayesian-optimization
```

```
Collecting bayesian-optimization
  Downloading bayesian_optimization-1.4.3-py3-none-any.whl (18 kB)
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from bayesian-optimization) (1.25.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from bayesian-optimization) (1.11.4)
Requirement already satisfied: scikit-learn>=0.18.0 in /usr/local/lib/python3.10/dist-packages (from bayesian-optimization) (1.2.2)
Collecting colorama>=0.4.6 (from bayesian-optimization)
  Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18.0->bayesian-optimization) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18.0->bayesian-optimization) (3.
Installing collected packages: colorama, bayesian-optimization
Successfully installed bayesian-optimization-1.4.3 colorama-0.4.6
```

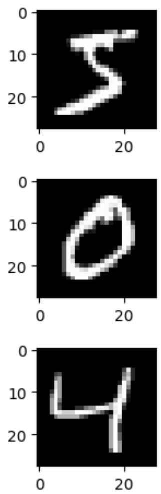
```
import numpy as np
import pandas as pd
from keras.datasets import mnist
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report,confusion_matrix
```

IMPORTING DATASET

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
from matplotlib import pyplot
for i in range(3):
  pyplot.subplot(330 + 1 + i)
  pyplot.imshow(X_train[i], cmap=pyplot.get_cmap('gray'))
  pyplot.show()
```




```
X_train = X_train.reshape(-1, 28*28)
X_test = X_test.reshape(-1, 28*28)
```

Normalizing the pixel values to range [0, 1]

```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

IMPLEMENTING TRAINING MODELS

DECISION TREE CLASSIFIER

```
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_scores = cross_val_score(dt_classifier, X_train, y_train, cv=5)
print("Decision Tree Classifier Accuracy: {:.2f}%".format(dt_scores.mean() * 100))
```

```
Decision Tree Classifier Accuracy: 86.65%
```

RANDOM FOREST CLASSIFIER

```
rf_classifier = RandomForestClassifier(random_state=42)
rf_scores = cross_val_score(rf_classifier, X_train, y_train, cv=5)
print("Random Forest Classifier Accuracy: {:.2f}%".format(rf_scores.mean() * 100))
```

```
Random Forest Classifier Accuracy: 96.64%
```

NAIVE BAYES CLASSIFIER

```
nb_classifier = GaussianNB()
nb_scores = cross_val_score(nb_classifier, X_train, y_train, cv=5)
print("Naïve Bayes Classifier Accuracy: {:.2f}%".format(nb_scores.mean() * 100))
```

```
Naïve Bayes Classifier Accuracy: 56.18%
```

KNN CLASSIFIER

```
knn_classifier = KNeighborsClassifier()
knn_scores = cross_val_score(knn_classifier, X_train, y_train, cv=5)
print("KNN Classifier Accuracy: {:.2f}%".format(knn_scores.mean() * 100))
```

```
KNN Classifier Accuracy: 96.93%
```

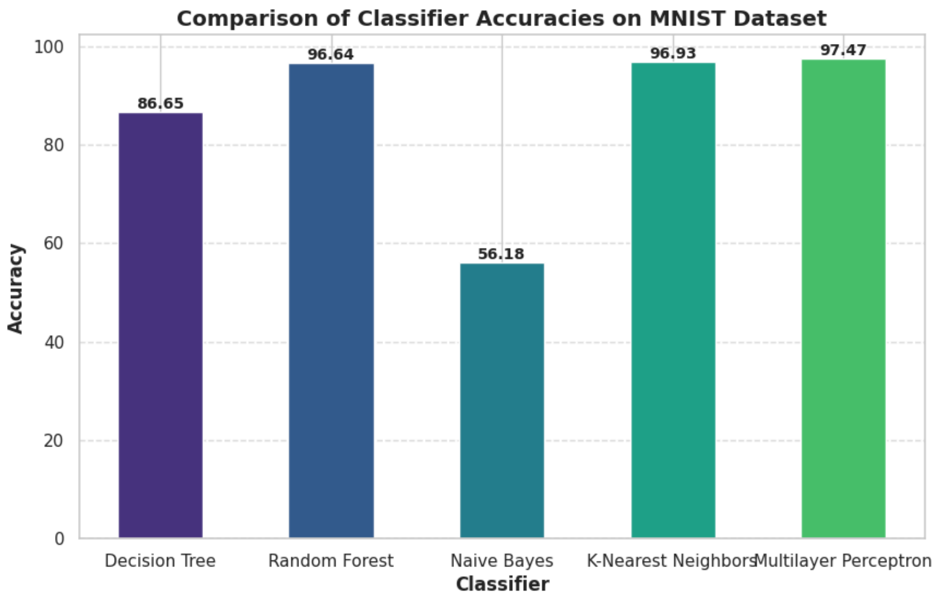
NEURAL NETWORK CLASSIFIER

```
nn_classifier = MLPClassifier(random_state=42)
nn_scores = cross_val_score(nn_classifier, X_train, y_train, cv=5)
print("Neural Network Classifier Accuracy: {:.2f}%".format(nn_scores.mean() * 100))
```

```
Neural Network Classifier Accuracy: 97.47%
```

Comparing through Graph

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style='whitegrid')
palette = sns.color_palette('viridis')

classifiers = ['Decision Tree', 'Random Forest', 'Naive Bayes', 'K-Nearest Neighbors', 'Multilayer Perceptron']
# classifier_accuracies=[86.65, 96.64, 56.18, 96.93, 97.47]
classifier_accuracies=[dt_scores.mean(), rf_scores.mean(), nb_scores.mean(), knn_scores.mean(), nn_scores.mean()]

plt.figure(figsize=(10, 6))
bars = plt.bar(classifiers, classifier_accuracies, color=palette, width=0.5)

plt.xlabel('Classifier', fontsize=12, fontweight='bold')
plt.ylabel('Accuracy', fontsize=12, fontweight='bold')
plt.title('Comparison of Classifier Accuracies on MNIST Dataset', fontsize=14, fontweight='bold')

plt.grid(axis='y', linestyle='--', alpha=0.6)

for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, yval + 0.01, f'{yval:.2f}', ha='center', va='bottom', fontsize=10, fontweight='bold')

plt.show()
```




```
data = {'Classifier': classifiers, 'Accuracy': classifier_accuracies}
df = pd.DataFrame(data)
df_sorted = df.sort_values(by='Accuracy', ascending=False)
df_sorted = df_sorted.reset_index(drop=True)
df_sorted['Rank'] = df_sorted.index + 1
print(df_sorted)

best_classifier = df_sorted.iloc[0]['Classifier']
best_accuracy = df_sorted.iloc[0]['Accuracy']
print(f"\nThe classifier with the highest accuracy ({best_accuracy:.4f}) is {best_classifier}, making it the best for classification.")
```

```
              Classifier  Accuracy  Rank
0  Multilayer Perceptron     97.47     1
1    K-Nearest Neighbors     96.93     2
2          Random Forest     96.64     3
3          Decision Tree     86.65     4
4            Naive Bayes     56.18     5

The classifier with the highest accuracy (97.4700) is Multilayer Perceptron, making it the best for classification.
```

EVALUATION METRICS

```python
classifiers = [dt_classifier, rf_classifier, nb_classifier, knn_classifier, nn_classifier]
classifier_names = ["Decision Tree", "Random Forest", "Naïve Bayes", "KNN", "Neural Network"]
```

```python
for clf, name in zip(classifiers, classifier_names):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    print(f"\n{name} Evaluation Metrics:")
    print(f"Accuracy: {accuracy}")
    print(f"Classification Report:\n{report}")
    print(f"Confusion matrix:\n{conf_matrix}")
    print('-'*150)
```

```
           0       0.96      0.99      0.98       980
           1       0.95      1.00      0.98      1135
           2       0.98      0.96      0.97      1032
           3       0.96      0.97      0.97      1010
           4       0.98      0.96      0.97       982
           5       0.97      0.97      0.97       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.99      0.94      0.96       974
           9       0.96      0.95      0.95      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000

Confusion matrix:
[[ 974    1    1    0    0    1    2    1    0    0]
 [   0 1133    2    0    0    0    0    0    0    0]
 [  11    8  991    2    1    0    1   15    3    0]
 [   0    3    3  976    1   13    1    6    3    4]
 [   3    7    0    0  944    0    4    2    1   21]
 [   5    0    0   12    2  862    4    1    2    4]
 [   5    3    0    0    3    2  945    0    0    0]
 [   0   22    4    0    3    0    0  988    0   11]
 [   8    3    5   13    6   12    5    5  913    4]
 [   5    7    3    9    7    3    1   10    2  962]]

Neural Network Evaluation Metrics:
Accuracy: 0.9782
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.98      0.98      1032
           3       0.98      0.98      0.98      1010
           4       0.97      0.97      0.97       982
           5       0.98      0.98      0.98       892
           6       0.98      0.98      0.98       958
           7       0.98      0.98      0.98      1028
           8       0.97      0.97      0.97       974
           9       0.97      0.97      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000

Confusion matrix:
[[ 972    0    1    0    1    0    2    1    3    0]
 [   0 1125    3    0    0    0    2    1    4    0]
 [   4    1 1007    2    1    0    4    5    7    1]
 [   0    0    1  989    0    6    0    3    4    7]
 [   3    0    3    1  956    0    4    4    2    9]
 [   2    0    0    6    3  871    4    0    4    2]
 [   4    2    2    1    4    5  938    0    2    0]
 [   1    2   11    2    0    0    0 1003    3    6]
 [   4    1    3    5    8    2    2    2  943    4]
 [   3    2    0    5    9    4    1    4    3  978]]
```

PARAMETER TUNING

Through Grid search

Defining parameter grid for each classifier

```python
dt_classifier = DecisionTreeClassifier(random_state=42)
rf_classifier = RandomForestClassifier(random_state=42)
knn_classifier = KNeighborsClassifier()
nn_classifier = MLPClassifier(random_state=42)

param_grid_dt = {
```

```
     'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
    'max_depth': [10, 20, 30,],
    'min_samples_split': [2, 5,],
     'min_samples_leaf': [1, 2, 4]
}

param_grid_rf = {
    'n_estimators': [50, 100],
     'criterion': ['gini', 'entropy'],
    'max_depth': [10, 20, 30],
     'min_samples_split': [2, 5, 10],
     'min_samples_leaf': [1, 2, 4]
}

param_grid_knn = {
    'n_neighbors': [3, 5, 7],
     'weights': ['uniform', 'distance'],
     'metric': ['euclidean', 'manhattan']
}

param_grid_nn = {
    'hidden_layer_sizes': [(50,), (100,)],
    'activation': ['logistic', 'relu'],
     'solver': ['adam', 'sgd'],
     'alpha': [0.0001, 0.001, 0.01]
}
```

Grid search for each classifier

```
grid_search_dt = GridSearchCV(dt_classifier, param_grid_dt, cv=5)
grid_search_rf = GridSearchCV(rf_classifier, param_grid_rf, cv=5)
grid_search_knn = GridSearchCV(knn_classifier, param_grid_knn, cv=5)
grid_search_nn = GridSearchCV(nn_classifier, param_grid_nn, cv=5)
```

Fit the grid search objects

```
grid_search_dt.fit(X_train, y_train)
grid_search_rf.fit(X_train, y_train)
grid_search_knn.fit(X_train, y_train)
grid_search_nn.fit(X_train, y_train)
```

Best parameters for each classifier

```
print("Best parameters for Decision Tree:", grid_search_dt.best_params_)
print("Best parameters for Random Forest:", grid_search_rf.best_params_)
print("Best parameters for KNN:", grid_search_knn.best_params_)
print("Best parameters for Neural Network:", grid_search_nn.best_params_)
```

```
Best parameters for Decision Tree: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

Through Parameter Search

Random Forest

Random Search

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform
```

```
# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)


param_dist = {
    'n_estimators': randint(10, 200),
    'max_depth': randint(1, 20),
    'min_samples_split': randint(2, 10),
    'min_samples_leaf': randint(1, 10),
    'max_features': ['auto', 'sqrt', 'log2']
}

rf_classifier = RandomForestClassifier(random_state=42)

random_search = RandomizedSearchCV(rf_classifier, param_distributions=param_dist, n_iter=20, cv=5, random_state=42)

random_search.fit(X_train, y_train);


print(f"Best parameters: {random_search.best_params_}")
print(f"Best cross-validation score: {random_search.best_score_:.4f}")
```

```
Best parameters: {'max_depth': 1, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 4, 'n_estimators': 10}
Best cross-validation score: 0.4660
```

Bayesian Optimization

```
from bayes_opt import BayesianOptimization
```

```
def rf_cv(n_estimators, max_depth, min_samples_split, min_samples_leaf):
    rf = RandomForestClassifier(
        n_estimators=int(n_estimators),
        max_depth=int(max_depth),
        min_samples_split=int(min_samples_split),
        min_samples_leaf=int(min_samples_leaf),
        random_state=42
    )
    cv_scores = cross_val_score(rf, X_train, y_train, cv=5)
    return cv_scores.mean()

pbounds = {
    'n_estimators': (10, 200),
    'max_depth': (1, 20),
    'min_samples_split': (2, 10),
    'min_samples_leaf': (1, 10),
}

# Initialize the Bayesian optimizer
optimizer = BayesianOptimization(
    f=rf_cv,
    pbounds=pbounds,
    random_state=42,
)

# Optimize
optimizer.maximize(init_points=5, n_iter=15)

# Print the best parameters and the corresponding best score
print(f"Best parameters: {optimizer.max['params']}")
print(f"Best cross-validation score: {optimizer.max['target']:.4f}")
```

Decision Tree

Random Search

```
param_dist_dt = {
'max_depth': randint(1, 20),
'min_samples_split': randint(2, 10),
'min_samples_leaf': randint(1, 10)
}

dt_classifier = DecisionTreeClassifier(random_state=42)

random_search_dt = RandomizedSearchCV(
dt_classifier, param_distributions=param_dist_dt, n_iter=1, cv=5, random_state=42
)
random_search_dt.fit(X_train, y_train)

print(f"Decision Tree Random Search - Best Parameters: {random_search_dt.best_params_}")
print(f"Decision Tree Random Search - Best CV Score: {random_search_dt.best_score_:.4f}")
```

```
Decision Tree Random Search - Best Parameters: {'max_depth': 7, 'min_samples_leaf': 4, 'min_samples_split': 6}
Decision Tree Random Search - Best CV Score: 0.7746
```

Bayesian Optimization

```
# Define function to optimize
def dt_cv(max_depth, min_samples_split, min_samples_leaf):
    dt = DecisionTreeClassifier(
        max_depth=int(max_depth),
        min_samples_split=int(min_samples_split),
        min_samples_leaf=int(min_samples_leaf),
        random_state=42
    )
    cv_scores = cross_val_score(dt, X_train, y_train, cv=5)
    return cv_scores.mean()

# Define parameter bounds for Bayesian optimization
pbounds_dt = {
    'max_depth': (1, 20),
    'min_samples_split': (2, 10),
    'min_samples_leaf': (1, 10)
}

# Initialize Bayesian optimizer
optimizer_dt = BayesianOptimization(
    f=dt_cv,
    pbounds=pbounds_dt,
    random_state=42
)

# Optimize
optimizer_dt.maximize(init_points=5, n_iter=15)

# Print the best parameters and score
print(f"Decision Tree Bayesian Optimization - Best Parameters: {optimizer_dt.max['params']}")
print(f"Decision Tree Bayesian Optimization - Best CV Score: {optimizer_dt.max['target']:.4f}")
```

```
|   iter    |  target   | max_depth | min_sa... | min_sa... |
-------------------------------------------------------------
```

```
| 1         | 0.8051    | 8.116     | 9.556     | 7.856     |
| 2         | 0.8667    | 12.37     | 2.404     | 3.248     |
| 3         | 0.3404    | 2.104     | 8.796     | 6.809     |
| 4         | 0.869     | 14.45     | 1.185     | 9.759     |
| 5         | 0.8685    | 16.82     | 2.911     | 3.455     |
| 6         | 0.866     | 14.05     | 6.92      | 7.359     |
=============================================================
Decision Tree Bayesian Optimization - Best Parameters: {'max_depth': 14.453378978124864, 'min_samples_leaf': 1.185260448662222, 'min_samples_split':
Decision Tree Bayesian Optimization - Best CV Score: 0.8690
```

KNN

Random Search

```
# Define parameter distributions for KNN
param_dist_knn = {
    'n_neighbors': randint(1, 20),
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'leaf_size': randint(10, 50),
    'p': randint(1, 2)
}

# Initialize K-Nearest Neighbors Classifier
knn_classifier = KNeighborsClassifier()

# Perform Random Search
random_search_knn = RandomizedSearchCV(
    knn_classifier, param_distributions=param_dist_knn, n_iter=20, cv=5, random_state=42
)
random_search_knn.fit(X_train, y_train)

# Print the best parameters and score
print(f"K-Nearest Neighbors Random Search - Best Parameters: {random_search_knn.best_params_}")
print(f"K-Nearest Neighbors Random Search - Best CV Score: {random_search_knn.best_score_:.4f}")
```

Bayesian Optimization

```
# Define function to optimize
def knn_cv(n_neighbors, leaf_size, p):
    knn = KNeighborsClassifier(
        n_neighbors=int(n_neighbors),
        leaf_size=int(leaf_size),
        p=int(p),
        weights='uniform',
        algorithm='auto'
    )
    cv_scores = cross_val_score(knn, X_train, y_train, cv=5)
    return cv_scores.mean()

# Define parameter bounds for Bayesian optimization
pbounds_knn = {
    'n_neighbors': (1, 20),
    'leaf_size': (10, 50),
    'p': (1, 2)
}

# Initialize Bayesian optimizer
optimizer_knn = BayesianOptimization(
    f=knn_cv,
    pbounds=pbounds_knn,
    random_state=42
)

# Optimize
optimizer_knn.maximize(init_points=5, n_iter=15)

# Print the best parameters and score
print(f"K-Nearest Neighbors Bayesian Optimization - Best Parameters: {optimizer_knn.max['params']}")
print(f"K-Nearest Neighbors Bayesian Optimization - Best CV Score: {optimizer_knn.max['target']:.4f}")
```

Neural Networks

Random Search

```
# Define parameter distributions for MLP
param_dist_mlp = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 100)],
    'activation': ['relu', 'tanh', 'logistic'],
    'solver': ['adam', 'sgd'],
    'alpha': uniform(0.0001, 0.1),
    'learning_rate': ['constant', 'invscaling', 'adaptive'],
    'learning_rate_init': uniform(0.001, 0.1),
    'max_iter': randint(100, 1000)
}

# Initialize MLPClassifier
mlp_classifier = MLPClassifier(random_state=42)

# Perform Random Search
random_search_mlp = RandomizedSearchCV(
```

```
    mlp_classifier, param_distributions=param_dist_mlp, n_iter=20, cv=5, random_state=42
)
random_search_mlp.fit(X_train, y_train)

# Print the best parameters and score
print(f"MLP Random Search - Best Parameters: {random_search_mlp.best_params_}")
print(f"MLP Random Search - Best CV Score: {random_search_mlp.best_score_:.4f}")
```

Bayesian Optimization

```
# Define function to optimize
def mlp_cv(hidden_layer_sizes, activation, solver, alpha, learning_rate, learning_rate_init, max_iter):
    mlp = MLPClassifier(
        hidden_layer_sizes=eval(hidden_layer_sizes),
        activation=activation,
        solver=solver,
        alpha=alpha,
        learning_rate=learning_rate,
        learning_rate_init=learning_rate_init,
        max_iter=max_iter,
        random_state=42
    )
    cv_scores = cross_val_score(mlp, X_train, y_train, cv=5)
    return cv_scores.mean()

# Define parameter bounds for Bayesian optimization
pbounds_mlp = {
    'hidden_layer_sizes': ['(50,)', '(100,)', '(50,50)', '(100,100)'],
    'activation': ['relu', 'tanh', 'logistic'],
    'solver': ['adam', 'sgd'],
    'alpha': (0.0001, 0.1),
    'learning_rate': ['constant', 'invscaling', 'adaptive'],
    'learning_rate_init': (0.001, 0.1),
    'max_iter': (100, 1000)
}

# Initialize Bayesian optimizer
optimizer_mlp = BayesianOptimization(
    f=mlp_cv,
    pbounds=pbounds_mlp,
    random_state=42
)

# Optimize
optimizer_mlp.maximize(init_points=5, n_iter=15)

# Print the best parameters and score
print(f"MLP Bayesian Optimization - Best Parameters: {optimizer_mlp.max['params']}")
print(f"MLP Bayesian Optimization - Best CV Score: {optimizer_mlp.max['target']:.4f}")
```