

Activity Test

Understanding Classification, Training, and Testing a Classifier:

**1. What is classification?**

Classification is a machine learning task where the goal is to assign predefined labels or categories to input data based on its features. It is a type of supervised learning, meaning that the algorithm learns from a labeled dataset to make predictions on new, unseen data.

**2. What is training classifier?**

Training a classifier involves teaching a machine learning model to recognize patterns and relationships in the training data. The process typically includes the following steps:

1. **Data Collection**: Gather a dataset with labeled examples. Each example consists of input features and corresponding output labels.
2. **Data Preprocessing**: Clean and preprocess the data to ensure it is suitable for training. This may involve handling missing values, normalizing features, or encoding categorical variables.
3. **Model Selection**: Choose a classification algorithm or model based on the nature of the problem and the characteristics of the data.
4. **Feature Extraction**: Identify and select relevant features that contribute to the model's ability to make accurate predictions.
5. **Training the Model**: Use the labeled training data to train the model. During training, the algorithm adjusts its parameters to minimize the difference between predicted and actual labels.
6. **Model Evaluation**: Assess the model's performance on a separate validation set to ensure it generalizes well to new, unseen data.
7. **Hyperparameter Tuning**: Fine-tune the model's hyperparameters to optimize its performance.

**3. What is testing classifier?**

Testing a classifier involves evaluating its performance on a separate dataset that it has not seen during training. This is crucial for assessing how well the model is expected to perform on new, real-world data. The testing process typically includes the following steps:

1. **Data Splitting**: Divide the dataset into training and testing sets. The training set is used to train the model, while the testing set is reserved for evaluating its performance.
2. **Model Prediction**: Apply the trained model to the testing set to obtain predictions for the input data.
3. **Performance Metrics**: Calculate various performance metrics such as accuracy, precision, recall, and F1 score to measure how well the model performs on the testing data.
4. **Analysis and Improvement**: Analyze the results and, if necessary, make adjustments to the model or data preprocessing steps to improve performance.

By following these steps, users can effectively develop, train, and test classifiers for various applications, ranging from image recognition to spam detection.

**SOURCE CODE**

# Function to load and preprocess images

def load\_and\_preprocess\_image(image\_path):

    image = io.imread(image\_path, as\_gray=True)

    image = transform.resize(image, (28, 28))  # Resize image to a common size (you may adjust this size)

    image = exposure.equalize\_adapthist(image)

    return image.flatten()

# Function to load images and labels from a folder

def load\_images\_from\_folder(folder\_path):

    images = []

    labels = []

    for filename in os.listdir(folder\_path):

        img\_path = os.path.join(folder\_path, filename)

        if os.path.isfile(img\_path):

            images.append(load\_and\_preprocess\_image(img\_path))

            labels.append(int(os.path.basename(folder\_path)))

    return images, labels

# Function to load dataset

def load\_dataset(root\_folder):

    all\_images = []

    all\_labels = []

    for digit\_folder in os.listdir(root\_folder):

        digit\_folder\_path = os.path.join(root\_folder, digit\_folder)

        if os.path.isdir(digit\_folder\_path):

            images, labels = load\_images\_from\_folder(digit\_folder\_path)

            all\_images.extend(images)

            all\_labels.extend(labels)

    return all\_images, all\_labels

# Load training dataset

train\_root\_folder = 'Train'

X\_train, y\_train = load\_dataset(train\_root\_folder)

# Load testing dataset

test\_root\_folder = 'Test'

X\_test, y\_test = load\_dataset(test\_root\_folder)

**RANDOM FOREST CLASSIFIER**

**SOURCE CODE**

# Create a Random Forest classifier

clf = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Train the classifier

clf.fit(X\_train, y\_train)

# Make predictions on the test set

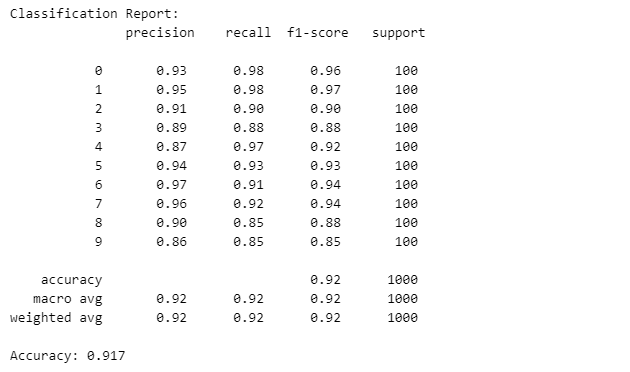
y\_pred = clf.predict(X\_test)

# Print classification report and accuracy

print("Classification Report:\n", metrics.classification\_report(y\_test, y\_pred))

print("Accuracy:", metrics.accuracy\_score(y\_test, y\_pred))

**OUTPUT**

****

**SUPPORT VECTOR MACHINES (SVM)**

**SOURCE CODE**

# Create an SVM classifier

clf = svm.SVC(kernel='linear', C=1)

# Train the classifier

clf.fit(X\_train, y\_train)

# Make predictions on the test set

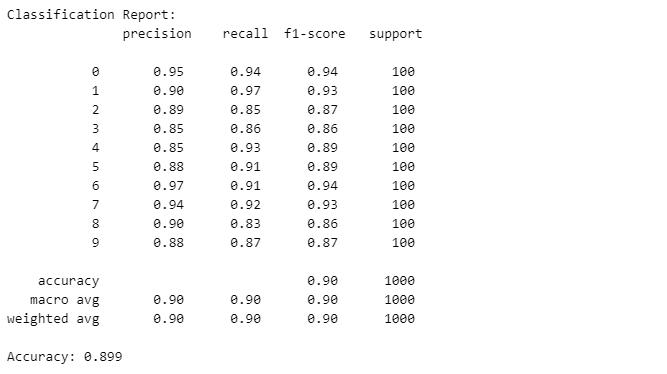
y\_pred = clf.predict(X\_test)

# Print classification report and accuracy

print("Classification Report:\n", metrics.classification\_report(y\_test, y\_pred))

print("Accuracy:", metrics.accuracy\_score(y\_test, y\_pred))

**OUTPUT**

****

**K-NEAREST NEIGHBORS (KNN)**

**SOURCE CODE**

# Create a KNN classifier with k=3 (you can adjust k as needed)

knn = KNeighborsClassifier(n\_neighbors=3)

# Train the classifier

knn.fit(X\_train, y\_train)

# Flatten the input images for prediction

X\_test\_flattened = [image.flatten() for image in X\_test]

# Make predictions on the test set

y\_pred = knn.predict(X\_test\_flattened)

# Print classification report and accuracy

print("Classification Report:\n", metrics.classification\_report(y\_test, y\_pred))

print("Accuracy:", metrics.accuracy\_score(y\_test, y\_pred))

**NEURAL NETWORK CLASSIFIER USING TENSORFLOW/KERAS**

**SOURCE CODE**

# Convert images and labels to NumPy arrays

X\_train = np.array(X\_train)

y\_train = np.array(y\_train)

X\_test = np.array(X\_test)

y\_test = np.array(y\_test)

# Convert labels to one-hot encoding

lb = LabelBinarizer()

y\_train = lb.fit\_transform(y\_train)

y\_test = lb.transform(y\_test)

# Reshape the input data to match the expected input shape

X\_train = X\_train.reshape((-1, 28, 28))

X\_test = X\_test.reshape((-1, 28, 28))

# Create a simple neural network model

model = Sequential()

model.add(Flatten(input\_shape=(28, 28)))  # Flatten the 28x28 images to a 1D array

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

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

# Compile the model

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

# Train the model

model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_split=0.2)

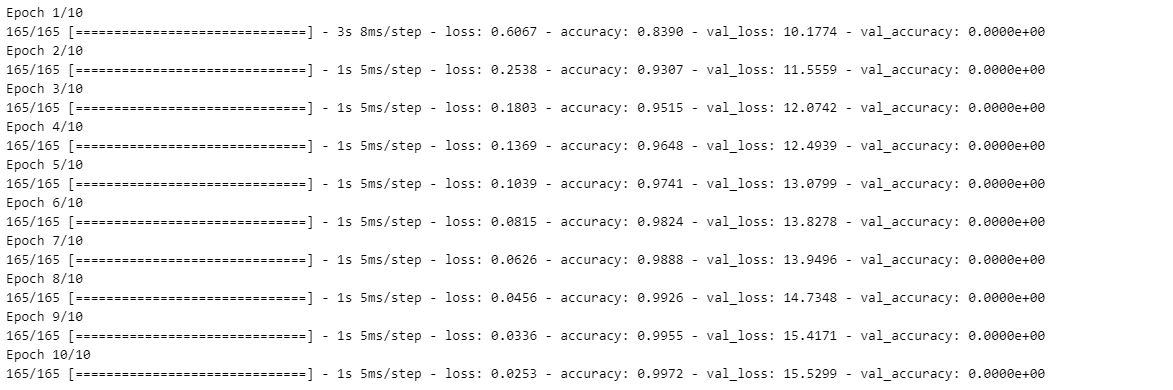
# Evaluate the model on the test set

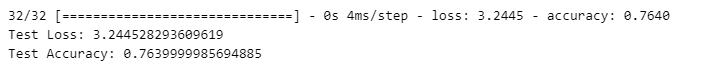
loss, accuracy = model.evaluate(X\_test, y\_test)

print("Test Loss:", loss)

print("Test Accuracy:", accuracy)

**OUTPUT**

****

****