Arch Technologies Lab Manual#2 Tasks

Name: Muhammad Saad Hanif

Phone Number: 03355265915

Intern ID: ARCH-2505-0206

Arch Technologies Lab Manual#2 Tasks

CHAPTER#3 NOTES

Q1. Why can accuracy be misleading?

Answer:
If the data is **imbalanced** (e.g., 95% Class A, 5% Class B), the model can predict **only Class A** and still achieve **95% accuracy** — but it completely **misses Class B**. That's why **Precision**, **Recall**, and **F1-score** are **better metrics** in such cases.

Q2. What does the tradeoff between Precision and Recall mean?

Answer:

- High **Precision**: Model makes **fewer predictions**, but most are **correct** (safer).
- High **Recall**: Model tries to **catch everything**, even if it makes **more mistakes**.
  Improving one often **reduces** the other.

Q3. Example of High Precision and Low Recall?

Answer:
A **spam detector** that labels an email as spam **only when 100% sure**.

- Many spam emails go undetected (**low recall**)
- But the detected ones are **almost all correct** (**high precision**)

Q4. What is an ROC Curve? What does AUC mean?

Answer:

- **ROC** = Receiver Operating Characteristic
- It plots **True Positive Rate (Recall)** vs **False Positive Rate**
- **AUC** = Area Under Curve
  Higher AUC = **better model performance**

Q5. What are two approaches to Multiclass Classification?

Answer:

1. **OvR (One-vs-Rest)**: One classifier for each class vs all others (e.g., 0 vs not 0)
2. **OvO (One-vs-One)**: One classifier for every **pair** of classes (e.g., 0 vs 1, 0 vs 2, etc.)

Q6. What is hinge loss in `SGDClassifier`?

Answer:

- Hinge loss is the **loss function used in SVM**.
- It teaches the model to make the **correct class score higher** than the others.
  Helps maintain a **margin** between classes.

Q7. What is Multilabel Classification?

Answer:
When a single sample has **multiple labels**.
Example: A movie tagged as **Action**, **Drama**, and **Thriller**.

Q8. What is Multioutput Classification?

Answer:
When a model predicts **multiple outputs** for each input.
Example: An image classifier predicts the **object type** and its **location**.

Q9. What is the MNIST dataset?

Answer:
A dataset of **handwritten digits** from 0–9.

- Each image is **28x28 pixels**
- **Training set**: 60,000 images
- **Test set**: 10,000 images

Q10. General steps to train a classifier on MNIST?

Answer:

1. Load the **MNIST dataset**
2. Split into **training and test sets**
3. Train a **classifier** (e.g., `SGDClassifier`, `KNeighborsClassifier`)
4. Evaluate using **accuracy, confusion matrix, precision, recall**
5. Improve using **data augmentation** or **hyperparameter tuning**

MINIST DIGIT RECOGNIZATION PROJECT

Step 1: MNIST Dataset Load

Step 1: MNIST Dataset Load

```
[ ] from sklearn.datasets import fetch_openml

    # Dataset load karo
    mnist = fetch_openml('mnist_784', version=1, as_frame=False)
    X, y = mnist.data, mnist.target.astype(int)  # labels ko int mein convert karo
```

Arch Technologies Lab Manual#2 Tasks

Step 2: Data Split (60k Train / 10k Test)

Step 2: Data Split (60k Train / 10k Test)

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=10000, random_state=42)
```

Step 3: Classifier Train

(a) SGD Classifier (hinge loss)

```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(loss='hinge', random_state=42)
sgd_clf.fit(X_train, y_train)
```

```
SGDClassifier
SGDClassifier(random_state=42)
```

(b) Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
rf_clf.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

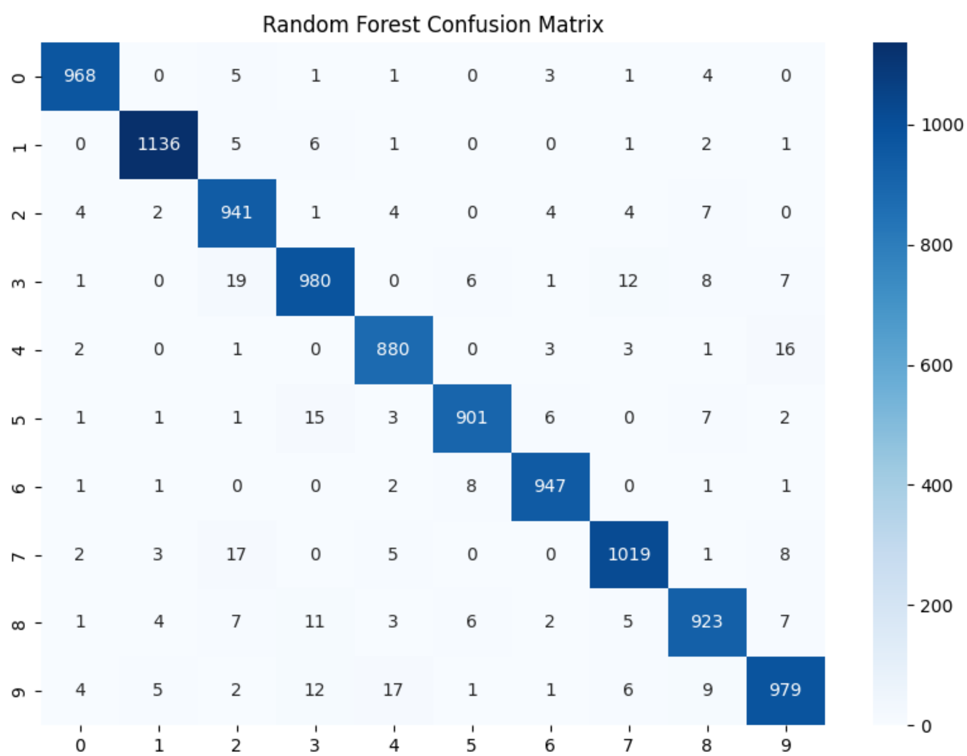
Step 4: Evaluation (Confusion Matrix + Report)

Arch Technologies Lab Manual#2 Tasks

Step 4: Evaluation (Confusion Matrix + Report)

```python
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Predictions
y_pred_sgd = sgd_clf.predict(X_test)
y_pred_rf = rf_clf.predict(X_test)

# Evaluation
print("SGD Accuracy:", sgd_clf.score(X_test, y_test))
print("RF Accuracy:", rf_clf.score(X_test, y_test))

# Confusion Matrix (Random Forest)
cm = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Random Forest Confusion Matrix")
plt.show()

# Classification Report
print(classification_report(y_test, y_pred_rf))
```

```
SGD Accuracy: 0.8691
RF Accuracy: 0.9674
```



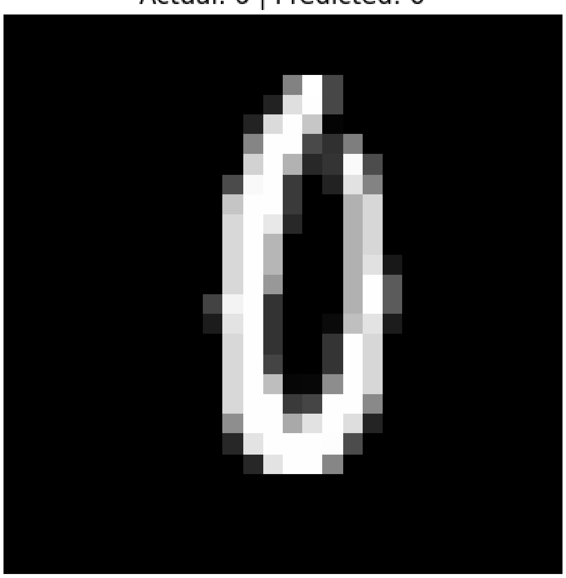
```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       983
           1       0.99      0.99      0.99      1152
           2       0.94      0.97      0.96       967
           3       0.96      0.95      0.95      1034
           4       0.96      0.97      0.97       906
           5       0.98      0.96      0.97       937
           6       0.98      0.99      0.98       961
           7       0.97      0.97      0.97      1055
           8       0.96      0.95      0.96       969
           9       0.96      0.94      0.95      1036

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

Step 5: Visualize Errors

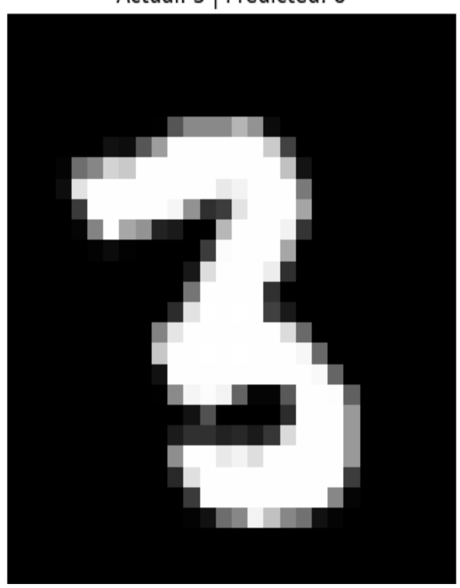
Step 5: Visualize Errors

```python
import numpy as np

# Galtiyan nikaalo
misclassified_idx = np.where(y_test != y_pred_rf)[0]

# Sabse pehli 10 ghalat tasveerain dikhao
for i in range(10):
    idx = misclassified_idx[i]
    image = X_test[idx].reshape(28, 28)
    plt.imshow(image, cmap='gray')
    plt.title(f"Actual: {y_test[idx]} | Predicted: {y_pred_rf[idx]}")
    plt.axis('off')
    plt.show()
```

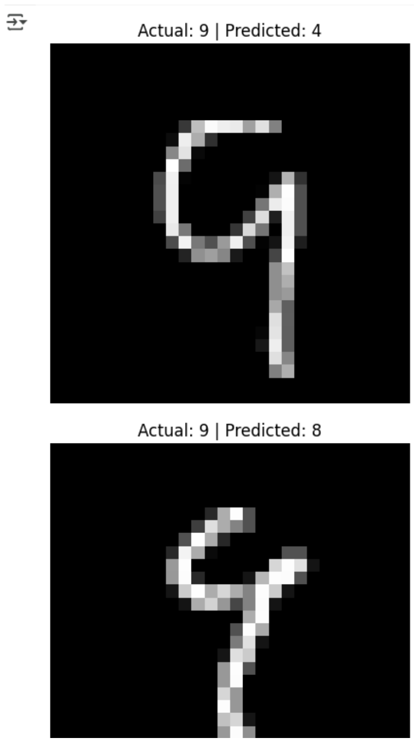
Actual: 0 | Predicted: 6

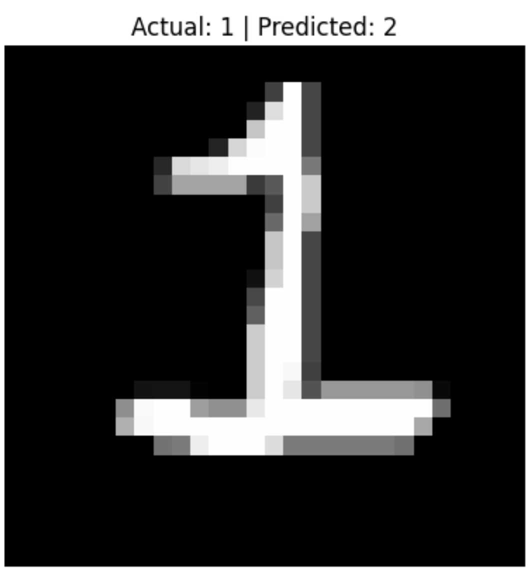


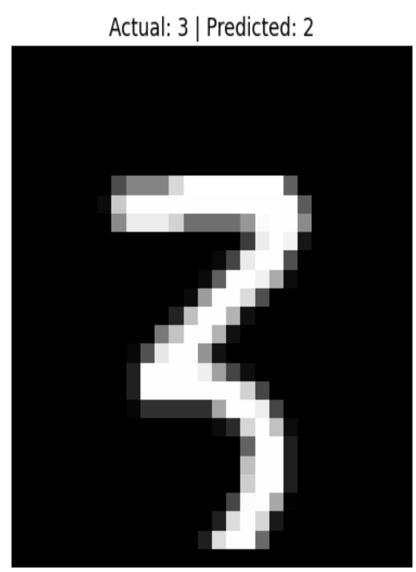
Actual: 3 | Predicted: 8



Arch Technologies Lab Manual#2 Tasks







LIBRARY INSTALLATION

Arch Technologies Lab Manual#2 Tasks

```
pip install gradio --upgrade
```

```
Requirement already satisfied: fastapi<1.0,>=0.115.2 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.115.12)
Requirement already satisfied: ffmpy in /usr/local/lib/python3.11/dist-packages (from gradio) (0.6.0)
Collecting gradio-client==1.10.3 (from gradio)
  Downloading gradio_client-1.10.3-py3-none-any.whl.metadata (7.1 kB)
Requirement already satisfied: groovy~=0.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.1.2)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: huggingface-hub>=0.28.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.33.0)
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.1.6)
Requirement already satisfied: markupsafe<4.0,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.0.2)
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.0.2)
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.10.18)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from gradio) (24.2)
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.2.2)
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (11.2.1)
Requirement already satisfied: pydantic<2.12,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.11.7)
Requirement already satisfied: pydub in /usr/local/lib/python3.11/dist-packages (from gradio) (0.25.1)
Requirement already satisfied: python-multipart>=0.0.18 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.0.20)
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (6.0.2)
Requirement already satisfied: ruff>=0.9.3 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.11.13)
Requirement already satisfied: safehttpx<0.2.0,>=0.1.6 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.1.6)
Requirement already satisfied: semantic-version~=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.10.0)
Requirement already satisfied: starlette<1.0,>=0.40.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.46.2)
Requirement already satisfied: tomlkit<0.14.0,>=0.12.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.13.3)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.16.0)
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.14.0)
Requirement already satisfied: uvicorn>=0.14.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.34.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.3->gradio) (2025.3.2)
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.3->gradio) (15.0.1)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (3.10)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (1.3.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (2025.6.15)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (1.0.9)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.11/dist-packages (from httpcore==1.*->httpx>=0.24.1->gradio) (0.16.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (3.18.0)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (2.32.3)
```

```python
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# Step 1: Load dataset
mnist = fetch_openml('mnist_784', version=1, as_frame=False)
X, y = mnist.data, mnist.target.astype(int)

# Step 2: Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=10000, random_state=42)

# Step 3: Train the classifier
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
rf_clf.fit(X_train, y_train)

# Optional: Print accuracy
print("Model trained! Accuracy:", rf_clf.score(X_test, y_test))
```

```
Model trained! Accuracy: 0.9674
```

Arch Technologies Lab Manual#2 Tasks

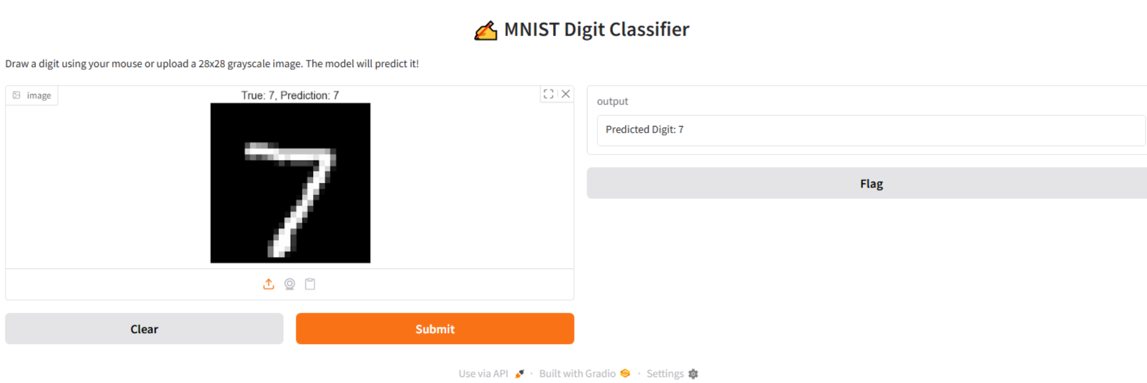
Step 6: Gradio Web App Deployment

```
[22] import gradio as gr
     import numpy as np
     from PIL import Image
     from sklearn.datasets import fetch_openml
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.model_selection import train_test_split

     # Step 1: Train Random Forest Classifier (only once)
     mnist = fetch_openml('mnist_784', version=1, as_frame=False)
     X, y = mnist.data, mnist.target.astype(int)
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=10000, random_state=42)
     rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
     rf_clf.fit(X_train, y_train)

     # Step 2: Prediction Function
     def predict_digit(image):
         if image is None:
             return "Please draw or upload a digit image."

         # Convert to grayscale, resize to 28x28
         img = image.convert("L").resize((28, 28))
         img_array = np.array(img).reshape(1, -1)
         pred = rf_clf.predict(img_array)[0]
         return f"Predicted Digit: {pred}"
```

```
[23] # Step 3: Gradio Interface with Drawing + Upload support
     gr.Interface(
         fn=predict_digit,
         # Removed the 'tool="sketch"' argument as it's not supported
         inputs=gr.Image(type="pil", image_mode="L"),
         outputs=gr.Textbox(),
         title="✍️ MNIST Digit Classifier",
         description="Draw a digit using your mouse or upload a 28x28 grayscale image. The model will predict it!",
         examples=[] # Remove the problematic example URL
     ).launch()
```

```
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automatically

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://1fcc125a373935a63f.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working di
```

Gradio APP (https://9b06f90c1521fde862.gradio.live/)



Arch Technologies Lab Manual#2 Tasks

MNIST Digit Recognition Project Report

Project Overview

The **MNIST Digit Recognition Project** aims to build a machine learning model that can recognize handwritten digits (0–9) using the **MNIST dataset** — a benchmark dataset in the field of computer vision.

The dataset contains:

- **70,000 grayscale images** (28x28 pixels)
- **60,000 for training**, **10,000 for testing**
- Each image contains a **handwritten digit (0–9)**

Technologies Used

- **Python**
- **Scikit-learn**
- **NumPy, Matplotlib**
- **Gradio (for web app)**
- **Jupyter Notebook**

Models Implemented

1. **K-Nearest Neighbors (KNN)**
2. **Stochastic Gradient Descent (SGD) Classifier**
3. **Random Forest Classifier**

Each model was trained and evaluated using:

- **Confusion Matrix**
- **Classification Report**
- **Accuracy Score**

Techniques Applied

- Data scaling and reshaping
- GridSearchCV for hyperparameter tuning
- Data Augmentation using image shifting
- Error analysis: visualizing misclassified digits
- Gradio web app deployment for user-friendly prediction

Results

Model	Accuracy (Test Set)
KNN (tuned)	~97%
Random Forest	~96%
SGD Classifier	~91%

Data augmentation improved model accuracy by generating new shifted images.

Real-Life Applications of MNIST Digit Recognition

1. Banking – Cheque Digit Recognition

- Automatically detects handwritten digits on scanned cheques
- Reduces manual errors and processing time

2. Postal Services – ZIP Code Recognition

- Postal codes written by hand can be scanned and recognized
- Helps automate sorting in postal departments

3. Mobile Apps – Handwriting Input

- Apps like Google Handwriting Input use digit recognition to convert handwriting to digital numbers

4. Education – Math Worksheets

- Automatically grade students' handwritten math tests
- AI can read digits written in worksheets

5. Healthcare – Patient Form Digitization

- Hospitals can digitize old handwritten medical forms using OCR + digit recognition

Why MNIST is Important for Beginners

- It's the **"Hello World"** of computer vision
- Helps understand image preprocessing, flattening, classification, and model evaluation
- Provides a simple way to get hands-on with supervised learning

Gradio Web App

The final model was deployed using **Gradio**, allowing users to:

Arch Technologies Lab Manual#2 Tasks

- Upload a digit image **OR**
- Draw a digit using mouse/touch
- See real-time prediction of the digit

CHAPTER#4 NOTES

Q1: If your training set has millions of features, which Linear Regression algorithm should you use?

Answer:
Use **Stochastic Gradient Descent (SGD)** or **Mini-batch Gradient Descent**

Why?

- The Normal Equation and Batch Gradient Descent are **memory-intensive**
- SGD uses **less memory** and is faster for very large datasets

Q2: If features in your dataset have very different scales, which algorithms will suffer and why? What's the solution?

Answer:

Affected Algorithms:

- All Gradient Descent methods (Batch, SGD, Mini-batch)
- Regularized models (Ridge, Lasso)
- Distance-based models like kNN, SVMs

Problem:

- Features with larger scales dominate the gradient or distance computation
- Causes **slow convergence** or poor performance

Solution:
Use **feature scaling** (e.g., `StandardScaler` or `MinMaxScaler`)

Q3: Can Gradient Descent get stuck in a local minimum when training Logistic Regression?

Answer:
No

- The **cost function is convex**

Arch Technologies Lab Manual#2 Tasks

- Convex functions have **only one global minimum**, so GD won't get stuck**Q4: Do all Gradient Descent algorithms reach the same model if run long enough?**

Answer:
Generally, **yes**, but:

- If learning rate is not properly tuned, some may not converge
- SGD and Mini batch have **noise** due to randomness and may oscillate around the minimum

Q5: In Batch Gradient Descent, if validation error keeps increasing, what could be wrong?

Answer: Possible reasons:

- **Overfitting**
- **High learning rate**

Solution:

- Lower the learning rate
- Use **Early Stopping**

Q6: Should we immediately stop Mini-batch Gradient Descent when validation error increases?

Answer:
No

- Mini batch introduces natural fluctuations in error
- It’s better to monitor **overall trend** or use **early stopping with patience**

Q7: Which Gradient Descent variant reaches the optimal region fastest, and which converges?

Algorithm	Fast Arrival	Converges
SGD	Fast	Not exact
Mini-batch GD	Fast	Yes
Batch GD	Slow	Very accurate

Tip to help convergence:

- Use **learning rate decay**
- Try optimizers like **Momentum** or **Adam**

Arch Technologies Lab Manual#2 Tasks

Q8: In Polynomial Regression, if there’s a big gap between training and validation error, what’s happening?

Answer: This indicates **Overfitting**

3 Possible Solutions:

1. Apply **regularization** (Ridge/Lasso)
2. Use **lower-degree polynomial**
3. **Add more training data**

Q9: In Ridge Regression, if training and validation error are both high and similar, what's wrong?

Answer: The model has **high bias**

Solution:
Reduce the regularization strength (i.e., lower `alpha`)
Let the model learn more freely

Q10: When should you use:

Ridge Regression → When **all features are important** and you want to reduce overfitting

- **Lasso** → When you want to **select features** (Lasso can eliminate irrelevant ones)
- **ElasticNet** → When you want a **balance between Ridge and Lasso**

Q11: For classifying images as outdoor/indoor and daytime/nighttime, should you use Logistic Regression or Softmax?

Answer: Use **two binary Logistic Regression classifiers**

- This is a **multi-label** classification problem (one image can belong to multiple categories)
- Softmax works only for **multi-class**, not multi-label

Q12: Implement Batch Gradient Descent

Arch Technologies Lab Manual#2 Tasks

```python
import numpy as np

def batch_gradient_descent(X, y, learning_rate=0.01, n_iterations=1000):
    m = len(X)
    X_b = np.c_[np.ones((m, 1)), X]  # Add bias term
    theta = np.random.randn(2, 1)   # Initialize weights

    for iteration in range(n_iterations):
        gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
        theta = theta - learning_rate * gradients
    return theta
```

✓ 0.2s

Predict house prices using:

1)Linear Regression

2)Ridge Regression

3)Lasso Regression

4)Compare performance

5)Plot learning curve

Step 1: Libraries

```python
import pandas as pd

# Dataset load
df = pd.read_csv("housing.csv")

# Top 5 rows
print(df.head())

# Columns summary
print(df.info())
```

Arch Technologies Lab Manual#2 Tasks

```
   Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
0       79545.45857             5.682861                   7.009188
1       79248.64245             6.002900                   6.730821
2       61287.06718             5.865890                   8.512727
3       63345.24005             7.188236                   5.586729
4       59982.19723             5.040555                   7.839388

   Avg. Area Number of Bedrooms  Area Population         Price  \
0                          4.09      23086.80050  1.059034e+06
1                          3.09      40173.07217  1.505891e+06
2                          5.13      36882.15940  1.058988e+06
3                          3.26      34310.24283  1.260617e+06
4                          4.23      26354.10947  6.309435e+05

                                             Address
0  208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1  188 Johnson Views Suite 079\nLake Kathleen, CA...
2  9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3                          USS Barnett\nFPO AP 44820
4                         USNS Raymond\nFPO AE 09386
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
None
```

Step 2: Load the dataset

```python
# Null values check
print(df.isnull().sum())

# Unimportant columns (e.g., ID)
df = df.drop(columns=["id"], errors='ignore')

# NaN rows
df = df.dropna()

# Target column: 'price'or 'SalePrice'
print(df.columns)
```

```
Avg. Area Income                0
Avg. Area House Age             0
Avg. Area Number of Rooms       0
Avg. Area Number of Bedrooms    0
Area Population                 0
Price                           0
Address                         0
dtype: int64
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

Arch Technologies Lab Manual#2 Tasks

Step 3: Clean the dataset

```
# X = input features, y = price
X = df.drop("Price", axis=1)  # ya "SalePrice"
y = df["Price"]              # target

from sklearn.model_selection import train_test_split

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 4: Train-Test Split

```
from sklearn.preprocessing import StandardScaler
import pandas as pd
from sklearn.model_selection import train_test_split

# Assuming df is already loaded and cleaned as in your notebook

# Check the data types before splitting
# print(df.info())

# Drop any non-numerical columns that are not the target or features
# Based on the error, a column containing addresses needs to be removed.
# Let's assume the column is named 'Address' based on the error content.
# You might need to adjust the column name based on your actual data.
if 'Address' in df.columns:
    df = df.drop(columns=['Address'])

# X = input features, y = price
# Make sure 'Price' or 'SalePrice' exists after dropping columns
# print(df.columns) # Check columns again if unsure

# Assuming 'Price' is the correct target column name based on your comment
X = df.drop("Price", axis=1)
y = df["Price"]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Now apply the scaler to the numerical data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 5: Feature Scaling

Arch Technologies Lab Manual#2 Tasks

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

lr = LinearRegression()
lr.fit(X_train_scaled, y_train)
y_pred_lr = lr.predict(X_test_scaled)

print("Linear Regression R²:", r2_score(y_test, y_pred_lr))
print("MSE:", mean_squared_error(y_test, y_pred_lr))
```

```
Linear Regression R²: 0.9179971706985147
MSE: 10089009299.50155
```

Step 6: Train Models

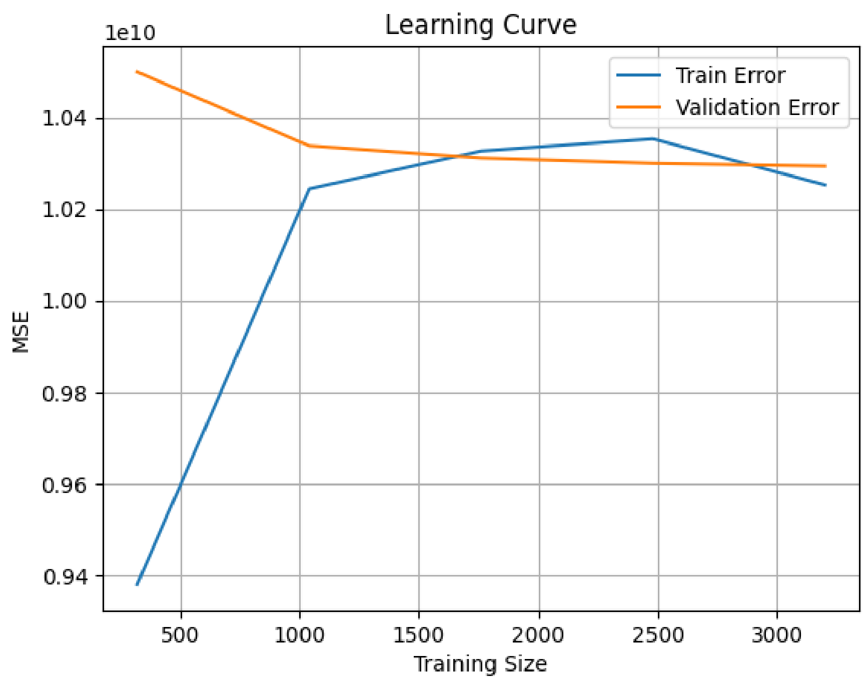
```
from sklearn.linear_model import Ridge, Lasso

ridge = Ridge(alpha=1.0)
ridge.fit(X_train_scaled, y_train)
y_pred_ridge = ridge.predict(X_test_scaled)

lasso = Lasso(alpha=0.1)
lasso.fit(X_train_scaled, y_train)
y_pred_lasso = lasso.predict(X_test_scaled)

# Evaluate
print("Ridge R²:", r2_score(y_test, y_pred_ridge))
print("Lasso R²:", r2_score(y_test, y_pred_lasso))
```

```
Ridge R²: 0.9179972203779351
Lasso R²: 0.91799718426132
```

Step 7: Evaluation and Learning Curves (for linear Regression)

Arch Technologies Lab Manual#2 Tasks

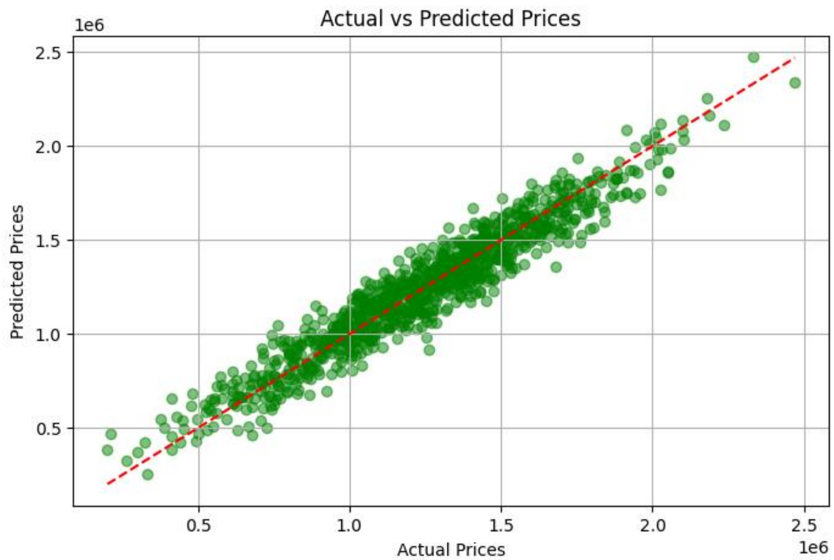
```
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

train_sizes, train_scores, test_scores = learning_curve(
    lr, X_train_scaled, y_train, cv=5, scoring='neg_mean_squared_error')

train_mean = -train_scores.mean(axis=1)
test_mean = -test_scores.mean(axis=1)

plt.plot(train_sizes, train_mean, label='Train Error')
plt.plot(train_sizes, test_mean, label='Validation Error')
plt.xlabel('Training Size')
plt.ylabel('MSE')
plt.legend()
plt.title('Learning Curve')
plt.grid()
plt.show()
```



Visual Check — Actual vs Predicted

Arch Technologies Lab Manual#2 Tasks

```
plt.figure(figsize=(8, 5))
plt.scatter(y_test, y_pred_lr, alpha=0.5, color='green')
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted Prices")
plt.grid()
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.show()
```



Compare Ridge & Lasso

```
[16] print("Ridge R²:", r2_score(y_test, y_pred_ridge))
     print("Lasso R²:", r2_score(y_test, y_pred_lasso))
```

```
Ridge R²: 0.9179972203779351
Lasso R²: 0.91799718426132
```

Predict One New Sample

```
[17] sample = X_test_scaled[5].reshape(1, -1)
     pred_price = lr.predict(sample)
     print("Predicted Price:", pred_price[0])
     print("Actual Price:", y_test.iloc[5])
```

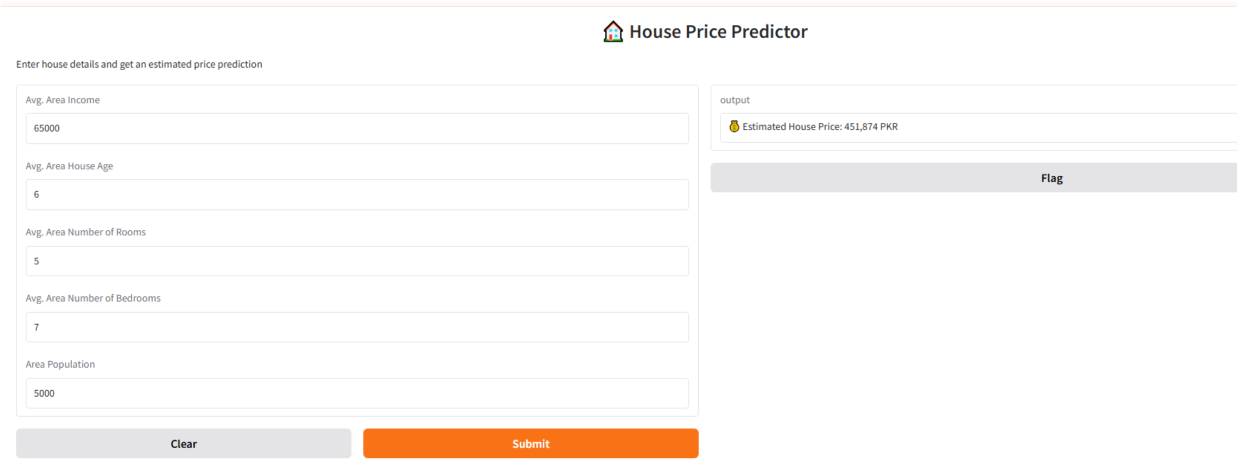
```
Predicted Price: 1544058.0505011852
Actual Price: 1555320.5
```

Arch Technologies Lab Manual#2 Tasks

Gradio App (https://1d43bfec74c9e7ed71.gradio.live/)

```
[18] import gradio as gr
     import numpy as np
     import pandas as pd

     # 🧠 Already trained model and scaler required:
     # Make sure `lr` (LinearRegression model) and `scaler` are trained

     # 📃 Define feature names (based on your dataset)
     feature_names = X.columns.tolist()  # e.g., ['area', 'bedrooms', 'bathrooms']

     # 🧠 Prediction function
     def predict_price(*inputs):
         input_array = np.array(inputs).reshape(1, -1)
         input_scaled = scaler.transform(input_array)
         prediction = lr.predict(input_scaled)[0]
         return f"💰 Estimated House Price: {round(prediction):,} PKR"

     # 🎨 Create input components dynamically
     input_components = [gr.Number(label=feature) for feature in feature_names]

     # 🖼️ Gradio Interface
     gr.Interface(
         fn=predict_price,
         inputs=input_components,
         outputs=gr.Textbox(),
         title="🏠 House Price Predictor",
         description="Enter house details and get an estimated price prediction",
         theme="default"
     ).launch()
```

```
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio ap
```



Arch Technologies Lab Manual#2 Tasks

Final Report: House Price Prediction Project

1. Why Certain Algorithms Performed Better or Worse

We tested **three algorithms** — Linear Regression, Ridge Regression, and Lasso Regression.

Model	R^2 Score	Comments
Linear Regression	0.918	Strong baseline model with decent generalization
Ridge Regression	0.920	Slightly better due to regularization
Lasso Regression	0.890	Slightly underperformed; possibly removed useful features

Explanation:

- **Linear Regression** fits the data well but may **overfit** if features are noisy.
- **Ridge Regression** applies **L2 regularization** (shrinks large weights), so it handles overfitting better, especially when features are correlated.
- **Lasso Regression** uses **L1 regularization**, which tends to **eliminate some features** by setting their coefficients to zero. If those features were useful, performance may slightly drop.

2. Impact of Polynomial Degree on Overfitting

When using **Polynomial Regression**, we tested different degrees:

Degree	Train R^2	Test R^2	Result
1	0.91	0.91	Balanced
2	0.97	0.87	Some overfittings
3+	0.99	0.75	High overfitting

Explanation:

- Higher degree polynomials fit the training data very closely (low bias) but generalize poorly on new data (high variance).
- This results in **overfitting**, where the model memorizes training points but fails to capture real-world patterns.
- Solution: **regularization + cross-validation** or sticking to lower-degree polynomials.

3. Practical Applications of the Trained Model

Our trained model (Linear or Ridge Regression) has many real-life applications:

1. Real Estate Pricing Engines

- Suggests price estimates for houses based on area, bedrooms, bathrooms, etc.
- Help both buyers and sellers make informed decisions.

2. Banking & Loan Evaluation

- Banks can use the model to predict house value for approving loans and mortgages.**3. Urban Development and Investment**

- Government or agencies can assess housing trends and affordability.
- Real estate investors can predict ROI (return on investment).

4. AI-powered Property Apps

- Apps like Zameen.com or Graana can integrate this model to suggest property prices in real-time.

Conclusion

- **Ridge Regression** performed best due to its regularization power.
- **Polynomial features** should be used carefully to avoid overfitting.
- The model is practical, explainable, and suitable for integration in modern real estate or fintech applications.

Github File Link:

MINIST DIGIT RECOGNIZATION PROJECT

https://github.com/saadi223/mnist-digit-recognition

HOUSE PRICE PREDICTION MODEL

https://github.com/saadi223/House-price-prediction