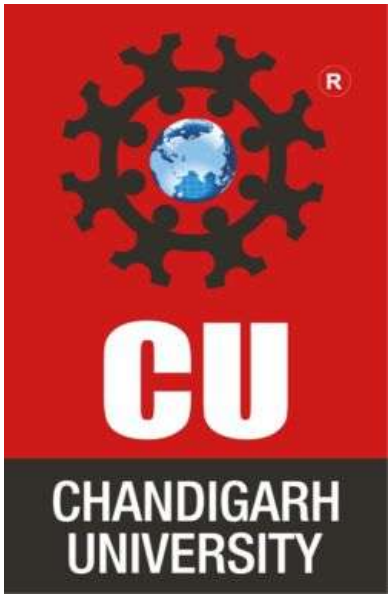
UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+

A blog on

Privacy-Preserving Federated Learning for Healthcare Data



Name: Sharad Pratap Singh, Radhe Shyam, Naga Santosh,

Semester: 3rd

Branch: MCA (General)

Section/Group:(2A)

Machine learning can seem like magic. You hear about algorithms that can diagnose diseases, drive cars, and recommend your next favourite song. But what does the code that powers this "magic" actually look like?

You might be surprised to learn that the basic structure of many machine learning projects is remarkably logical and consistent.

In this post, we'll strip away the complex theory and walk you through the essential, six-step recipe behind a simple machine learning model. We'll use Python, the most popular language for ML, and its powerful libraries to build a model that can predict whether a patient has diabetes based on their medical measurements.

Let's get started!

Step 1: The Setup - Importing Your Tools

Every project starts with gathering your tools. In Python, this means importing the necessary libraries. For a basic ML task, you almost always need these three:

- **Pandas:** The ultimate tool for loading and manipulating data (think of it as Excel on steroids).
- **Scikit-learn (sklearn):** The Swiss Army knife of machine learning. It has everything you need to build, train, and evaluate models.
- **NumPy:** The fundamental package for numerical computation in Python. Pandas and Scikit-learn are built on top of it.

Python

```
# Import the necessary libraries

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score
```

Step 2: Loading the Data

You can't do machine learning without data. Our goal is to train a model to find patterns in a dataset. We'll use a publicly available diabetes dataset. In a real project, this would be your .csv file or a database.

Using Pandas, loading data is a one-liner:

Python

```
# Load the dataset from a CSV file
```

```
df = pd.read_csv('diabetes.csv')


# Take a quick look at the first 5 rows

print(df.head())
```

This will show you a table with columns like Glucose, BloodPressure, BMI, and our target column, Outcome (where 1 means the patient has diabetes and 0 means they don't).

Step 3: Preparing the Data - Features and Target

Now, we need to tell our model what we're trying to predict and what information it should use to do so.

- **Features (X):** These are the inputs or the "clues" the model will learn from. In our case, these are all the medical measurement columns (Glucose, BMI, etc.).
- **Target (y):** This is the output or the "answer" we want the model to predict. For us, it's the Outcome column.

We separate our data into X and y:

Python

```
# Define the features (all columns except 'Outcome')

X = df.drop('Outcome', axis=1)


# Define the target variable (the 'Outcome' column)

y = df['Outcome']
```

Step 4: Splitting the Data - Training and Testing

This is one of the most important concepts in machine learning. We need to know if our model is actually learning or just memorizing the data. To do this, we split our dataset into two parts:

1. **Training Set (~80%):** The model will look at this data to learn the patterns. This is the "study material."
2. **Testing Set (~20%):** We hold this data back. The model will never see it during training. We use it at the very end to evaluate how well the model performs on new, unseen data. This is the "final exam."

Scikit-learn makes this incredibly easy:

Python

```
# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 5: Choosing and Training the Model

Now for the exciting part—the actual "learning." We need to choose a model. Since we're predicting one of two outcomes (diabetes or not), a **Logistic Regression** is a great, simple choice.

Training the model is just two lines of code:

1. Create an instance of the model.
2. "Fit" the model to the training data. The .fit() method is where the model looks at the X_train and y_train and learns the relationship between the medical features and the diabetes outcome.

Python

```
# Create an instance of the Logistic Regression model

model = LogisticRegression(max_iter=1000) # max_iter helps the model converge


# Train the model on the training data

model.fit(X_train, y_train)
```

That's it! The model has now "learned" the patterns from the training data.

Step 6: Making Predictions & Evaluating the Model

The training is done. But how good is our model? It's time to take our unseen test data (X_test) and ask the model to make predictions.

Python

```
# Make predictions on the test data

predictions = model.predict(X_test)
```

The predictions variable now holds the model's best guess for each patient in our test set. To see how well it did, we compare these predictions to the actual answers (y_test). The most straightforward metric is **accuracy**.

Python

```
# Calculate the accuracy of the model

accuracy = accuracy_score(y_test, predictions)
```

print(f"Model Accuracy: {accuracy * 100:.2f}%")

If you run this, you'll likely see an accuracy of around 75-78%, which is a great start! It means our model correctly predicted the outcome for about 3 out of every 4 patients in the unseen test set.

Putting It All Together

Here is the complete, basic script from start to finish.

Python

```
# 1. Import libraries

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score


# 2. Load the data

df = pd.read_csv('diabetes.csv')


# 3. Prepare data into features (X) and target (y)

X = df.drop('Outcome', axis=1)

y = df['Outcome']


# 4. Split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# 5. Create and train the model

model = LogisticRegression(max_iter=1000)

model.fit(X_train, y_train)


# 6. Make predictions and evaluate
```

```
predictions = model.predict(X_test)

accuracy = accuracy_score(y_test, predictions)


print(f"Model Accuracy: {accuracy * 100:.2f}%")
```

Conclusion

And there you have it! You've just walked through the fundamental workflow of a machine learning project. While real-world projects can get much more complex, this six-step process—**Load, Prepare, Split, Train, Predict, and Evaluate**—is the backbone of almost everything you'll do in machine learning.