Birla Institute of Technology and Science Pilani, Hyderabad Campus

1st Semester 2025-26, BITS F464: Machine Learning: Assignment No: 4

(Gaussian Naïve Bayes, and Multi-Layer Perceptron Model)

Date Given: 10.11.2025 Max. Marks: 15 Submission date: 27.11.2025

Objective: The goal of this assignment is to train two models- Naive Bayes and a Multi-Layer Perceptron (MLP) to predict the type of diabetes based on input features (age, blood sugar level (Glucose), insulin level, and BMI). You will compare the performance of both the models using evaluation metrics such as accuracy, precision, recall, and F1-score. Additionally, you will save both models using the `pickle` library and demonstrate how to use them in a Flask application for real-time predictions.

The Naïve Bayes Generative Classifier is a widely-used algorithm in machine learning, it operates on the principles of Bayes' theorem and assumes independence among features, allowing it to make predictions quickly and with minimal computational resources. It predicts the class for a given instance based on the class probabilities computed using Bayes' theorem. After calculating the posterior probability of each class given the instance's features, the algorithm selects the class with the highest probability as the predicted class for that instance. In other words, it assigns the class that maximizes the posterior probability. As we discussed in the class, the maximum a posteriori (MAP) that selects the best hypothesis for Naïve Bayes classifier is as given below:

$$\text{Max. A Posteriori (MAP)} = argmax\ P(y).\prod_{i=1}^{n} P(x_i|y) \quad (1)$$

In Gaussian Naïve Bayes, the likelihood $P(x_i|y)$ is often modelled using a Gaussian(normal) distribution. The probability density function (PDF) of a Gaussian distribution is:

$$P(x_i|y=c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} \exp\left(-\frac{(x_i-\mu_c)^2}{2\sigma_c^2}\right) \quad (2)$$

In this assignment, you will first create a machine learning-based web application to classify whether a patient is diabetic or not via the **GaussianNB** algorithm. You will use Flask for the backend to serve the model and JavaScript (with fetch API) for the frontend to make asynchronous API calls.

Step 1: Import the required libraries like GaussianNB from sklearn.naive_bayes, Perceptron from sklearn.linear_model, and pickle (for saving ML models to a file and later load them back into your program).

Step 2: Pre-processing: Split the dataset into features(x) and target(y), Perform a train-test split (80% training and 20% testing), if necessary perform feature scaling (Standard or MinMax).

Step 3: Train Naïve Bayes model. Use the diabetes dataset provided to train and test the model. (naive_bayes_model = GaussianNB(),naive_bayes_model.fit(X_train,y_train), y_pred_nb=naive_bayes_model.predict (X_test))

Step 4: Train a MLP model on the same dataset. Use the diabetes dataset provided to train and test the model. Below is a possible usage:

mlp_model = MLPClassifier(hidden_layer_sizes=(50,), activation='relu', solver='adam', max_iter=1000, random_state=42)

Alternatively, you may implement a custom Multilayer multilayer MLPClassifier(MLP) (MLP) by calculating linear combinations using dot products like:

linear_output = np.dot(x_i, self.weights) + self.bias, y_predicted = np.sign(linear_output); an update rule if prediction is incorrect, self.weights += self.learning_rate*y_[idx] * x_i, self.bias += self.learning_rate * y_[idx].

And similarly for predictions. (A custom implementation will attract 2 Marks more than the library one)

Step 5: Compare both models using evaluation metrics such as accuracy, precision, recall, and F1-score. Highlight the strengths and weaknesses of each model based on the results.

Step 6: Save both the models. Use the pickle library to save both the trained models. (naive_bayes_model.pkl and mlp_model.pkl).

Step 7: Set Up the Flask Backend. You are required to create a Flask-based backend that serves the trained Naive Bayes model and exposes an API endpoint to make predictions. The backend will:

- Load the diabetes_model.pkl file containing the trained model.
- While setting up the backend make sure that CORS is enabled so that you are able to make the API calls on that specific port address. CORS (Cross-Origin Resource Sharing) is a mechanism that enables web applications to request resources from a different origin (domain, protocol, or port) than the one from which the application was served.
- Provide a route (/predict) that accepts input data (age, blood sugar level, insulin level, and BMI) via a POST request and returns the predicted diabetes type as a JSON response.

(A Python script (app.py) that: (Suggested)

- Initializes the Flask app.
- Loads the trained model. Extend the Flask API to allow the user to select which model (Naive Bayes or Multilayer multilayer MLPClassifier(MLP) (MLP)) to use for predictions. You will modify the /predict route to accept a model type in the request. Implement a /predict API endpoint that accepts JSON input and returns predictions in JSON format. (part is shown in Fig.1)
- Runs on http://127.0.0.1:5000. (Need not be same can be different as well)

Step 8: Create the Frontend (HTML + JavaScript / React.js)

You need to build a simple frontend using HTML and JavaScript that allows users to input their age, blood sugar level, insulin level, and BMI. When the user clicks a button, the frontend should make an asynchronous API call to the Flask backend (/predict) using the fetch API. The backend's response should be displayed on the same page, showing the predicted diabetes type. Front end interface is as shown in Fig.2.

(An HTML file (index.html) that: (Suggested)

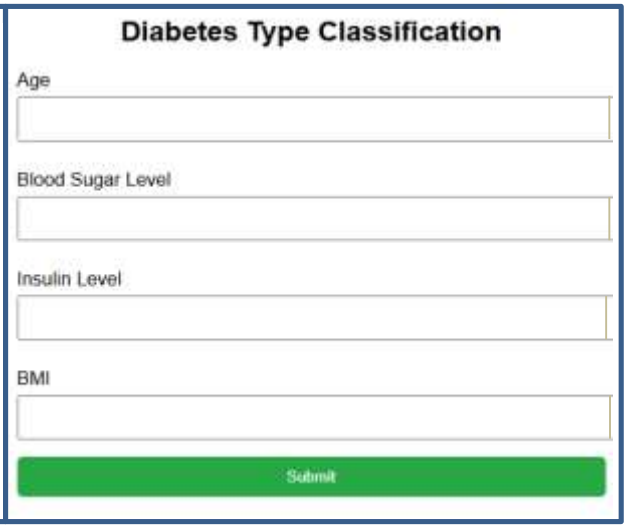
- Has input fields for age, blood sugar level, insulin level, and BMI.
- Includes a button to trigger a JavaScript function that sends a POST request to the Flask API.
- Displays the prediction result on the same page without reload.)

```
@app.route('/predict', methods=['POST'])

def predict():
 data = request.get_json()
 model_type = data.get('model_type', 'naive_bayes')
 input_features = np.array([[ data['age'],
    data['glucose'], data['insulin'], data['bmi'] ]])
 # Choose the model based on input
 if model_type == 'naive_bayes':
  prediction = loaded_nb_model.predict(input_features)
 elif model_type == 'perceptron':
  prediction=loaded_perceptron_model.predict(input_features)
 else:
  return jsonify({'error': 'Invalid model type'}), 400

return jsonify({'diabetes_type': int(prediction[0])})
```

(Fig.1)



(Fig.2)

Step 9: Implement k-fold cross-validation to evaluate both model's performance more robustly. (Similar to the previous assignments)

Step 10: Compare and analyze the performance metrics for both the models.

Deliverables: Python code for generating both the models (Gaussian Naïve Bayes and Multi-Layer Perceptron Classifier), saving them, and comparing their performance. Flask application code to demonstrate the use of both models in real-time predictions.

Submission Instructions: Include all the necessary python files as well as the frontend files in the zip. Any clarification on this coding assignment may be emailed to I/C, Pratyush Bindal (f20220119@hyderabad.bits-pilani.ac.in) or Aryan Shrivastava (f20221653@hyderabad.bits-pilani.ac.in).

References:

- https://www.geeksforgeeks.org/deploy-machine-learning-model-using-flask/
- http://medium.com/swlh/making-use-of-apis-in-your-front-end-c168e343bea3
- https://www.kaggle.com/code/vitorgamalemos/Multilayer multilayer MLPClassifier(MLP) (MLP)-neural-network/notebook