To implement a **Naïve Bayes classifier** from scratch in the **MERN stack**, you will need to follow these steps, ensuring each part integrates into your project. I'll guide you through a **basic** Naïve Bayes implementation using the following structure:

1. **Frontend** (React) for user interactions and displaying results.
2. **Backend** (Node.js/Express) for handling logic, receiving input (e.g., resumes or job descriptions), and training/testing the Naïve Bayes model.
3. **Database** (MongoDB) for storing training data (job descriptions and categories).

Let's break down the process.

**Step 1: Set Up the MERN Stack Environment**

Make sure you have the basic **MERN stack** set up:

1. **Backend**: Express server with Node.js.
2. **Frontend**: React for displaying the user interface.
3. **Database**: MongoDB to store the job categories and descriptions.

**Step 2: Set Up MongoDB to Store Training Data**

We’ll use MongoDB to store job categories and job descriptions. Each job description will be mapped to a category.

**MongoDB Schema (Mongoose Model)**

Create a file called jobCategory.js inside the models folder for storing job categories and job descriptions.

// models/jobCategory.js

const mongoose = require('mongoose');

const jobCategorySchema = new mongoose.Schema({

category: { type: String, required: true }, // e.g., "Frontend Developer"

text: { type: String, required: true } // e.g., "React JavaScript Redux"

});

module.exports = mongoose.model('JobCategory', jobCategorySchema);

**Step 3: Set Up Express API for Adding Training Data**

You will need an API to add job categories and their descriptions to the database. This API will allow you to update the training data dynamically.

// routes/jobCategory.js

const express = require('express');

const JobCategory = require('../models/jobCategory');

const router = express.Router();

// Add a job category and description

router.post('/add', async (req, res) => {

const { category, text } = req.body;

try {

const newCategory = new JobCategory({ category, text });

await newCategory.save();

res.status(201).json({ message: 'Job category added successfully!' });

} catch (error) {

res.status(500).json({ error: error.message });

}

});

module.exports = router;

In your server.js or app.js, add this route:

const express = require('express');

const mongoose = require('mongoose');

const bodyParser = require('body-parser');

const jobCategoryRoutes = require('./routes/jobCategory');

const app = express();

app.use(bodyParser.json());

// Connect to MongoDB

mongoose.connect('mongodb://localhost:27017/jobportal', { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log('Connected to MongoDB'))

.catch(err => console.log(err));

// Routes

app.use('/api/job-category', jobCategoryRoutes);

const PORT = 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

**Step 4: Implement the Naïve Bayes Classifier Logic (Backend)**

You need to implement the Naïve Bayes classifier in a separate file to handle the logic of training and classifying job descriptions. Here's a basic implementation of the Naïve Bayes algorithm:

// utils/naiveBayes.js

const \_ = require('lodash');

let categoryWordCounts = {};

let categoryCounts = {};

let totalWords = 0;

async function trainNaiveBayes() {

// Fetch training data from MongoDB

const JobCategory = require('../models/jobCategory');

const trainingData = await JobCategory.find();

categoryWordCounts = {};

categoryCounts = {};

totalWords = 0;

// Loop through training data to calculate word frequencies

trainingData.forEach(({ text, category }) => {

const words = text.toLowerCase().split(/\s+/);

if (!categoryWordCounts[category]) {

categoryWordCounts[category] = {};

categoryCounts[category] = 0;

}

categoryCounts[category] += 1;

words.forEach(word => {

categoryWordCounts[category][word] = (categoryWordCounts[category][word] || 0) + 1;

totalWords += 1;

});

});

console.log('Training complete!');

}

// Classify a new job description

function classifyJob(text) {

const words = text.toLowerCase().split(/\s+/);

let maxCategory = null;

let maxProbability = -Infinity;

Object.keys(categoryCounts).forEach(category => {

let probability = Math.log(categoryCounts[category] / totalWords); // Prior probability

words.forEach(word => {

const wordCount = categoryWordCounts[category][word] || 0;

const wordProbability = Math.log((wordCount + 1) / (totalWords + 1)); // Laplace smoothing

probability += wordProbability;

});

if (probability > maxProbability) {

maxProbability = probability;

maxCategory = category;

}

});

return maxCategory;

}

module.exports = { trainNaiveBayes, classifyJob };

**Step 5: Integrate Naïve Bayes Logic with the API**

Now, you can integrate this logic into your backend to classify new job descriptions or resumes that users submit.

// routes/classification.js

const express = require('express');

const { classifyJob, trainNaiveBayes } = require('../utils/naiveBayes');

const router = express.Router();

// Endpoint to classify a job description

router.post('/classify', (req, res) => {

const { text } = req.body;

// Classify the job description

const category = classifyJob(text);

res.json({ category });

});

module.exports = router;

In your server.js:

const classificationRoutes = require('./routes/classification');

app.use('/api/classify', classificationRoutes);

**Step 6: Frontend (React) for User Input and Results**

On the frontend, create a simple form where users can input job descriptions or upload their resumes.

// App.js (Frontend)

import React, { useState } from 'react';

import axios from 'axios';

const App = () => {

const [text, setText] = useState('');

const [category, setCategory] = useState('');

const handleSubmit = async (e) => {

e.preventDefault();

const response = await axios.post('http://localhost:5000/api/classify', { text });

setCategory(response.data.category);

};

return (

<div>

<h1>Job Role Classifier</h1>

<form onSubmit={handleSubmit}>

<textarea

value={text}

onChange={(e) => setText(e.target.value)}

rows="6"

cols="50"

placeholder="Paste job description or resume here..."

/>

<br />

<button type="submit">Classify</button>

</form>

{category && <h2>Predicted Job Role: {category}</h2>}

</div>

);

};

export default App;

**Step 7: Run the Project**

1. **Start the Backend (Express server):**
2. node server.js
3. **Start the Frontend (React app):**
4. npm start
5. Open the React app in the browser and test classifying job descriptions.

**Conclusion**

* **Backend**: Handles Naïve Bayes model training and classification.
* **Frontend**: Displays the UI where users can input job descriptions or resumes.
* **MongoDB**: Stores job categories and descriptions for dynamic learning.

This basic setup allows you to **classify job descriptions** based on the Naïve Bayes algorithm in your **MERN stack** application. You can expand this by adding features like resume upload, more sophisticated text extraction, and a better front-end UI for managing categories.