EXP NO: 9	MINI PROJECT - EMAIL SPAM DETECTION USING DECISION TREE AND SVM

AIM:

To develop a web-based email spam detection system using **Decision Tree** and **Support Vector Machine (SVM)** models, integrating **FastAPI** for backend machine learning inference and **Next.js** for the frontend interface. The goal is to classify emails as *Spam* or *Not Spam* based on their textual content.

ALGORITHM:

1. Collect a labeled dataset of emails categorized as spam or not spam.
2. Preprocess the text data by removing punctuation, numbers, and stopwords, and converting all text to lowercase.
3. Convert the cleaned text into numerical form using the TF-IDF vectorizer.
4. Split the dataset into training and testing sets in an 80:20 ratio.
5. Train a Decision Tree classifier and an SVM classifier on the training data.
6. Predict labels on the test data using both models and evaluate performance using accuracy, precision, recall, and F1-score.
7. Develop a FastAPI backend that loads the trained models and provides an API endpoint to classify input text as spam or not spam.
8. Create a Next.js frontend that allows users to enter an email, sends it to the FastAPI API, and displays the prediction result.

CODE:

Page.jsx

```
"use client";
import Image from "next/image";
import DecisionTree from "@/components/comp/dec";
import SVM from "@/components/comp/svm";
import { Tabs, TabsList, TabsTrigger, TabsContent } from "@/components/ui/tabs";
import { Separator } from "@/components/ui/separator";

export default function Home() {
  return (
    <div className="flex justify-center items-start min-h-screen bg-gray-950 text-white
py-12">
     <div className="w-full max-w-3xl">
       <h1 className="text-4xl font-bold text-center mb-8">E-mail spam detection</h1>
  AlertDialogHeader,
  AlertDialogTitle,
  AlertDialogDescription,
  AlertDialogFooter,
  AlertDialogAction,
} from "@/components/ui/alert-dialog";
import { Loader2 } from "lucide-react"; // spinner icon

const DecisionTree = () => {
  const [title, setTitle] = useState('');
  const [description, setDescription] = useState('');
  const [result, setResult] = useState('');
  const [open, setOpen] = useState(false);
  const [loading, setLoading] = useState(false);

  const handleTitle = (e) => setTitle(e.target.value);
  const handleDescription = (e) => setDescription(e.target.value);

  const handleSubmit = async (e) => {
  e.preventDefault();
  setLoading(true);
  const data = title + " " + description;

  const response = await fetch("http://127.0.0.1:8000/decision-tree", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({ text: data }),
  });

  const result = await response.json();
  console.log(result)
  setResult(result?.prediction === 1 ? "Spam" : "Not Spam");
  setLoading(false);
  setOpen(true);
};

  const handleClose = () => {
    setOpen(false);
    setTitle('');
    setDescription('');
    setResult('');
  };

  return (
    <div className="bg-gray-900 p-6 rounded-lg shadow-lg text-white relative">
      <h2 className="text-2xl font-semibold mb-6 text-center">E-mail Input Form</h2>

      {/* Spinner overlay */}
      {loading && (
        <div className="absolute inset-0 bg-black/50 backdrop-blur-sm flex flex-col
items-center justify-center rounded-lg z-10">
          <Loader2 className="w-10 h-10 text-blue-600 animate-spin mb-2" />
          <p className="text-gray-300">Analyzing email...</p>
        </div>
      )}

      <form onSubmit={handleSubmit} className="flex flex-col gap-5">
        <FieldSet className="flex flex-col gap-2">
          <FieldLabel>Title of the E-mail:</FieldLabel>
          <Input
            required
            value={title}
            onChange={handleTitle}
            placeholder="Enter email title"
            className="w-full bg-gray-800 text-white placeholder-gray-400"
            disabled={loading}
          />
        </FieldSet>

        <FieldSet className="flex flex-col gap-2">
          <FieldLabel>Description of the E-mail:</FieldLabel>
          <Textarea
            required
            value={description}
            onChange={handleDescription}
            placeholder="Enter email description"
            className="w-full h-36 bg-gray-800 text-white placeholder-gray-400"
            disabled={loading}
          />
        </FieldSet>

        <Button
          type="submit"
          disabled={loading}
            className="mt-4 bg-blue-600 hover:bg-blue-700 text-white py-2 rounded-md
cursor-pointer"
        >
          {loading ? "Processing..." : "Submit"}
        </Button>
      </form>

      <AlertDialog open={open} onOpenChange={setOpen}>
            <AlertDialogContent className="bg-gray-900/90 backdrop-blur-md border
border-gray-700 rounded-2xl text-white shadow-xl">
          <AlertDialogHeader>
            <AlertDialogTitle className="text-2xl font-semibold text-center mb-3">
              Prediction Result
            </AlertDialogTitle>
            <AlertDialogDescription className="text-lg text-center">
              The model predicts this email is:
              <br />
              <span
                className={`inline-block mt-3 text-3xl font-bold ${
                  result === "Spam" ? "text-red-400" : "text-green-400"
                }`}
              >
                {result}
              </span>
            </AlertDialogDescription>
          </AlertDialogHeader>
          <AlertDialogFooter className="flex justify-center mt-6">
            <AlertDialogAction
              onClick={handleClose}
                  className="bg-blue-600 hover:bg-blue-700 cursor-pointer text-black font-
semibold px-6 py-2 rounded-lg shadow-md transition-all"
            >
              Close
            </AlertDialogAction>
        </AlertDialogFooter>
      </AlertDialogContent>
    </AlertDialog>
  </div>
 );
};

export default DecisionTree;

svm.jsx
"use client";
import React, { useState } from 'react';
import { FieldLabel, FieldSet } from '../ui/field';
import { Input } from '../ui/input';
import { Textarea } from '../ui/textarea';
import { Button } from '../ui/button';
import {
 AlertDialog,
 AlertDialogContent,
 AlertDialogHeader,
 AlertDialogTitle,
 AlertDialogDescription,
 AlertDialogFooter,
 AlertDialogAction,
} from "@/components/ui/alert-dialog";
import { Loader2 } from "lucide-react"; // spinner icon

const SVM = () => {
 const [title, setTitle] = useState('');
 const [description, setDescription] = useState('');
 const [result, setResult] = useState('');
 const [open, setOpen] = useState(false);
 const [loading, setLoading] = useState(false);

 const handleTitle = (e) => setTitle(e.target.value);
 const handleDescription = (e) => setDescription(e.target.value);

 const handleSubmit = async (e) => {
  e.preventDefault();
  setLoading(true);
```

```
    const data = title + " " + description;

    const response = await fetch("http://127.0.0.1:8000/svm", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ text: data }),
    });

    const result = await response.json();
    console.log(result);
    setResult(result?.prediction === 1 ? "Spam" : "Not Spam");
    setLoading(false);
    setOpen(true);
  };

  const handleClose = () => {
    setOpen(false);
    setTitle('');
    setDescription('');
    setResult('');
  };

  return (
    <div className="bg-gray-900 p-6 rounded-lg shadow-lg text-white relative">
      <h2 className="text-2xl font-semibold mb-6 text-center">E-mail Input Form
(SVM)</h2>

      {loading && (
        <div className="absolute inset-0 bg-black/50 backdrop-blur-sm flex flex-col
items-center justify-center rounded-lg z-10">
          <Loader2 className="w-10 h-10 text-blue-600 animate-spin mb-2" />
          <p className="text-gray-300">Analyzing email...</p>
        </div>
      )}

      <form onSubmit={handleSubmit} className="flex flex-col gap-5">
        <FieldSet className="flex flex-col gap-2">
          <FieldLabel>Title of the E-mail:</FieldLabel>
          <Input
            required
            value={title}
            onChange={handleTitle}
            placeholder="Enter email title"
            className="w-full bg-gray-800 text-white placeholder-gray-400"
            disabled={loading}
          />
        </FieldSet>

        <FieldSet className="flex flex-col gap-2">
          <FieldLabel>Description of the E-mail:</FieldLabel>
          <Textarea
            required
            value={description}
            onChange={handleDescription}
            placeholder="Enter email description"
            className="w-full h-36 bg-gray-800 text-white placeholder-gray-400"
            disabled={loading}
          />
        </FieldSet>

        <Button
          type="submit"
          disabled={loading}
          className="mt-4 bg-blue-600 hover:bg-blue-700 text-white py-2 rounded-md
cursor-pointer"
        >
          {loading ? "Processing..." : "Submit"}
        </Button>
      </form>

      <AlertDialog open={open} onOpenChange={setOpen}>
        <AlertDialogContent className="bg-gray-900/90 backdrop-blur-md border
border-gray-700 rounded-2xl text-white shadow-xl">
          <AlertDialogHeader>
            <AlertDialogTitle className="text-2xl font-semibold text-center mb-3">
              Prediction Result (SVM)
            </AlertDialogTitle>
            <AlertDialogDescription className="text-lg text-center">
```

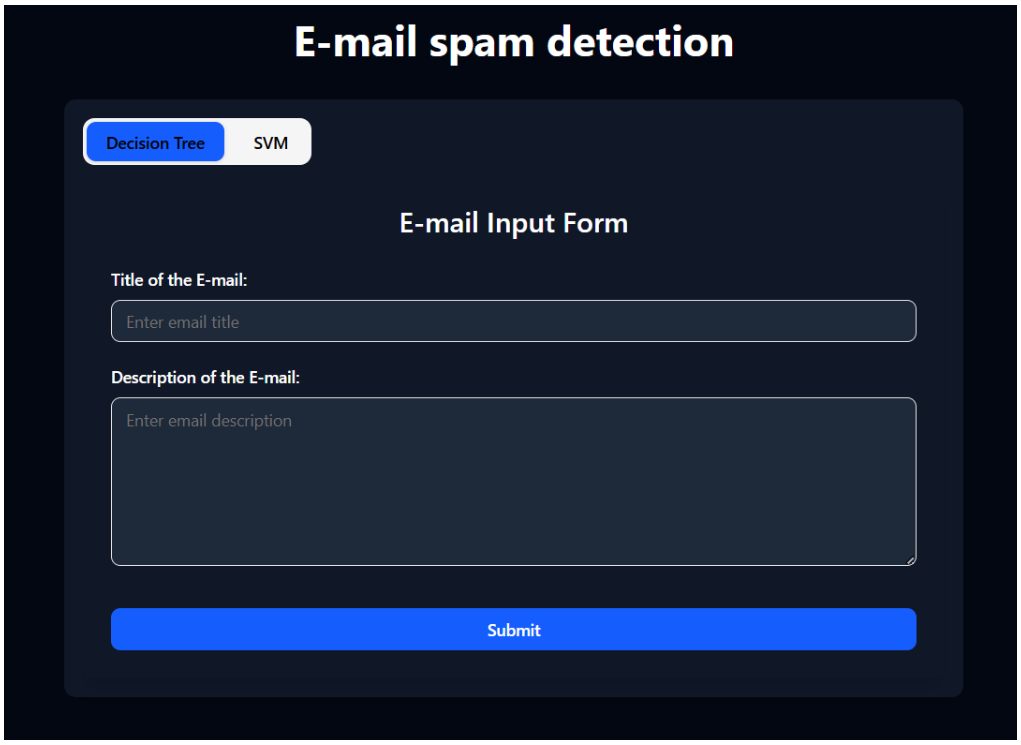
```
              The model predicts this email is:
              <br />
              <span
                className={`inline-block mt-3 text-3xl font-bold ${
                  result === "Spam" ? "text-red-400" : "text-green-400"
                }`}
              >
                {result}
              </span>
            </AlertDialogDescription>
          </AlertDialogHeader>
          <AlertDialogFooter className="flex justify-center mt-6">
            <AlertDialogAction
              onClick={handleClose}
              className="bg-blue-600 hover:bg-blue-700 cursor-pointer text-black font-
semibold px-6 py-2 rounded-lg shadow-md transition-all"
            >
              Close
            </AlertDialogAction>
          </AlertDialogFooter>
        </AlertDialogContent>
      </AlertDialog>
    </div>
  );
};

export default SVM;
```

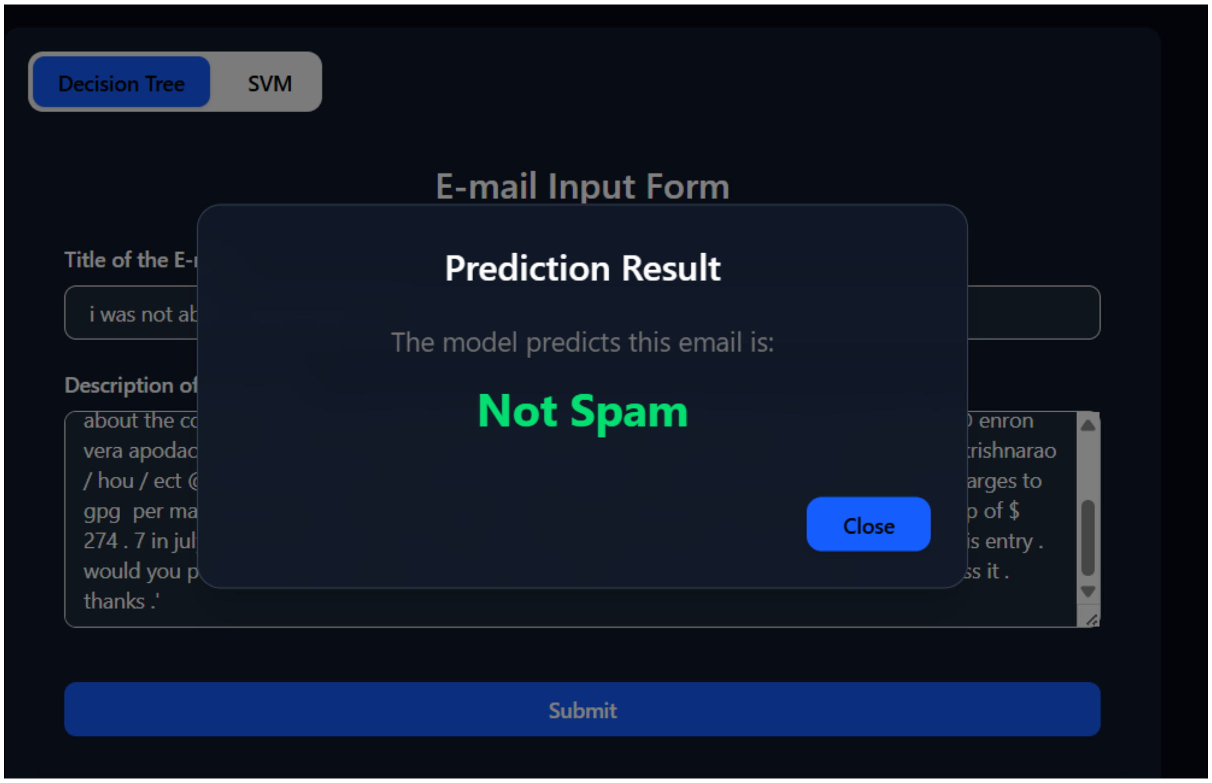
server.py

```
from fastapi import FastAPI, Request
import gc
import joblib
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from contextlib import asynccontextmanager
from fastapi.middleware.cors import CORSMiddleware
```

```
vectorise: TfidfVectorizer
decision_tree_model: DecisionTreeClassifier
svm_model: SVC

@asynccontextmanager
async def life_cycle(app: FastAPI):
    global vectorise
    global decision_tree_model
    global svm_model

    vectorise = joblib.load("./model/weights/dec_vectorizer.pkl")
    decision_tree_model = joblib.load("./model/weights/spam_detection_model.pkl")
    svm_model = joblib.load("./model/weights/svm_model.pkl")

    yield

    del decision_tree_model
    del svm_model
    del vectorise
    gc.collect()

app = FastAPI(lifespan=life_cycle)

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

@app.get('/')
async def root():
    return 'ML model Inference Server is running...'

@app.post('/decision-tree')
async def decision_tree(request: Request):
    data = await request.json()
    mail_data = data.get("text")
    vect_mail_data = vectorise.transform([str(mail_data)])
    prediction = decision_tree_model.predict(vect_mail_data)
    return {"prediction": int(prediction[0])}

@app.post('/svm')
async def svm(request: Request):
    data = await request.json()
    mail_data = data.get("text")
    vect_mail_data = vectorise.transform([str(mail_data)])
    prediction = svm_model.predict(vect_mail_data)
    return {"prediction": int(prediction[0])}
```

OUTPUT:





Prediction Result

The model predicts this email is:

Not Spam

RESULT:

The Decision Tree and SVM models were successfully trained and used to predict whether emails are spam or not.The SVM model showed slightly higher accuracy compared to the Decision Tree, making it more reliable for real-time email classification