

Project Report

Name: Predict Product Purchase Model

Report date: October 27, 2023

Internship Batch: LISUM26

Submitted by: Rupert Tawiah-Quashie

Submitted to: Github

This model takes in a person's age and yearly income and determines if they are likely to purchase the product. This product is promoted on social media platforms, and the data collected was based on the purchases made from social media platforms.

Below are the files I implemented:

Machine Learning Model: K-Nearest Neighbors

The screenshot shows a code editor interface with a dark theme. At the top, there is a tab bar with four tabs: 'model.py' (which is currently selected), 'app.py', 'index.html', and 'main.py'. Below the tabs, the code for 'model.py' is displayed in a syntax-highlighted text area. The code implements a K-Nearest Neighbors classifier for predicting product purchases based on age and income. It includes imports for numpy, matplotlib.pyplot, pandas, pickle, and StandardScaler from sklearn. It reads a CSV dataset, splits it into training and test sets, applies feature scaling, trains a KNN classifier with 5 neighbors and Minkowski metric, saves the model to disk, and finally predicts test results and generates a confusion matrix.

```
Part1 > model.py
Project model.py < app.py < index.html < main.py
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
4 import pickle
5
6 ## Importing the dataset
7 dataset = pd.read_csv("Social_Network_Ads.csv")
8 X = dataset.iloc[:, :-1].values
9 y = dataset.iloc[:, -1].values
10
11 ##Splitting the dataset into the Training set and test set
12 from sklearn.model_selection import train_test_split
13 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10, random_state=0)
14
15 ##Applying feature scaling to normalize the range of features
16 from sklearn.preprocessing import StandardScaler
17 sc = StandardScaler()
18 X_train = sc.fit_transform(X_train)
19 #X_test = sc.transform(X_test)
20
21 ##Training the K-NN model on the training set
22 from sklearn.neighbors import KNeighborsClassifier
23 classifier = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
24 classifier.fit(X_train, y_train)
25
26 ##Saving the model to disk
27 with open('model.pkl', 'wb') as model_file:
28     pickle.dump((classifier, sc), model_file)
29
30
31 #Predicting test results
32 y_pred = classifier.predict(X_test)
33
34 ##Making a confusion matrix to test the model
35 from sklearn.metrics import confusion_matrix, accuracy_score
36 cm = confusion_matrix(y_test, y_pred)
```

Importing necessary libraries:

The code begins by importing several Python libraries, including numpy for numerical operations, matplotlib.pyplot for data visualization, pandas for data manipulation, and pickle for serializing and saving the trained model to disk.

Importing the dataset:

- The code reads a CSV file named "Social_Network_Ads.csv" using the pandas library and stores the data in a variable called dataset. This dataset is made of 2 numerical variables
- It then separates the dataset into two parts:
- X: This variable contains the feature columns (all columns except the last one) of the dataset.
- y: This variable contains the target variable (the last column) of the dataset.

Splitting the dataset:

- The code splits the dataset into training and testing sets using the train_test_split function from sklearn.model_selection. It allocates 10% of the data for testing and uses a random seed (random_state) to ensure reproducibility.
- It results in four datasets:
- X_train: Features of the training set.
- X_test: Features of the test set.
- y_train: Target values of the training set.
- y_test: Target values of the test set.

Feature Scaling:

- Feature scaling is applied to normalize the range of feature values. This is important when using K-NN, as it relies on distance-based calculations.
- The code uses the StandardScaler from sklearn.preprocessing to standardize the feature values of the X_train dataset.

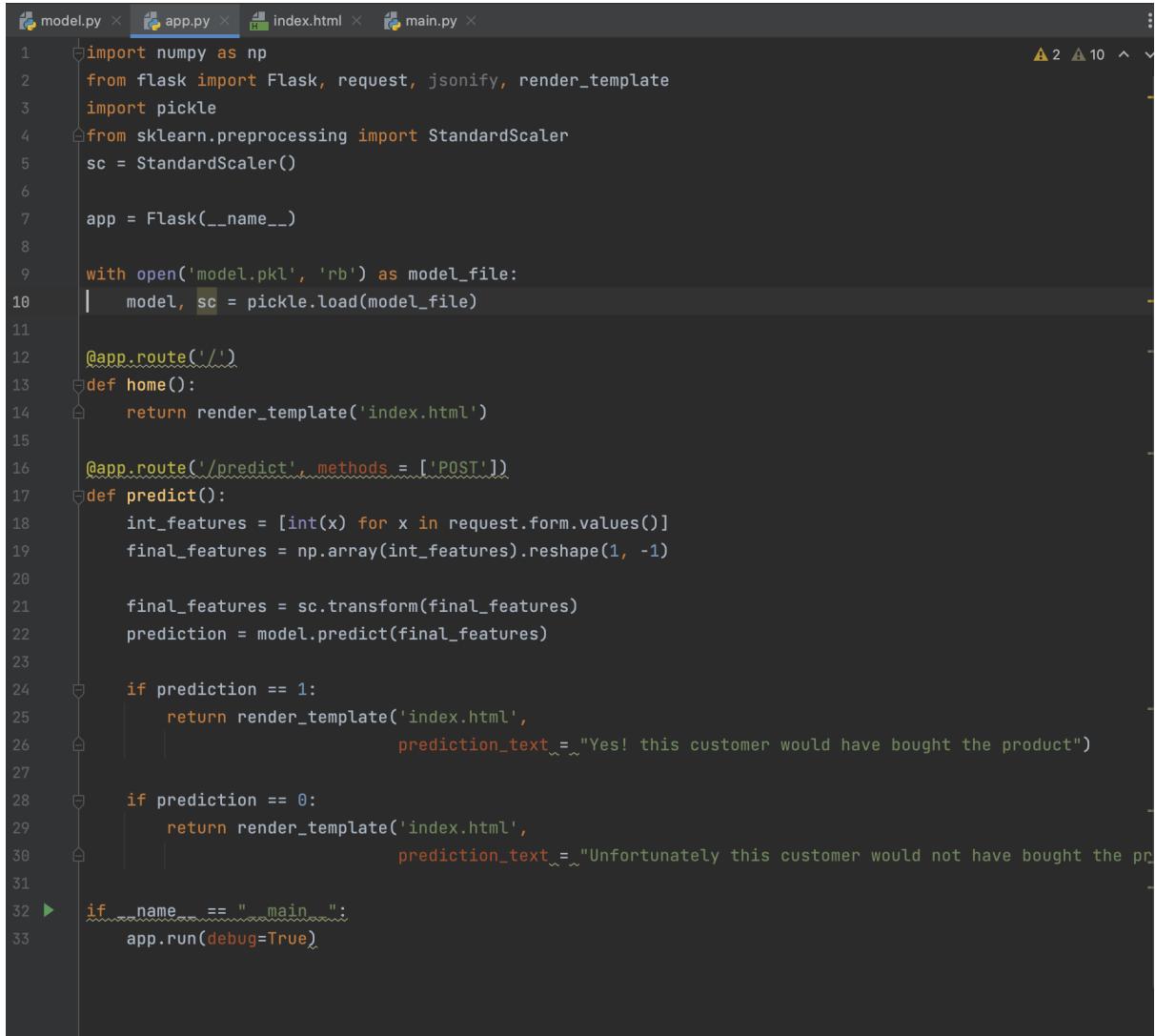
Training the K-NN model:

- The code imports the K-NN classifier (KNeighborsClassifier) from sklearn.neighbors.
- It creates an instance of the classifier with some parameters:
- n_neighbors: Number of neighbors to consider (set to 5 in this case).
- metric: The distance metric used, which is 'minkowski' in this case.
- p: The power parameter for the 'minkowski' metric, set to 2 (Euclidean distance).
- The K-NN model is then trained on the standardized training data (X_train and y_train) using the fitmethod of the classifier object.

Saving the model to disk:

- The trained K-NN model and the StandardScaler object used for feature scaling are saved to disk using the pickle module.
 - They are stored in a file named 'model.pkl' in binary write ('wb') mode.
- Predicting test results:
- The code uses the trained K-NN classifier to make predictions on the test dataset (X_test) and stores the predictions in the y_pred variable.

Deploying the model on Flask



The screenshot shows a code editor with several tabs at the top: 'model.py' (active), 'app.py', 'index.html', and 'main.py'. The code in 'model.py' is as follows:

```
1 import numpy as np
2 from flask import Flask, request, jsonify, render_template
3 import pickle
4 from sklearn.preprocessing import StandardScaler
5 sc = StandardScaler()
6
7 app = Flask(__name__)
8
9 with open('model.pkl', 'rb') as model_file:
10     model, sc = pickle.load(model_file)
11
12 @app.route('/')
13 def home():
14     return render_template('index.html')
15
16 @app.route('/predict', methods=['POST'])
17 def predict():
18     int_features = [int(x) for x in request.form.values()]
19     final_features = np.array(int_features).reshape(1, -1)
20
21     final_features = sc.transform(final_features)
22     prediction = model.predict(final_features)
23
24     if prediction == 1:
25         return render_template('index.html',
26                               prediction_text="Yes! this customer would have bought the product")
27
28     if prediction == 0:
29         return render_template('index.html',
30                               prediction_text="Unfortunately this customer would not have bought the product")
31
32 if __name__ == "__main__":
33     app.run(debug=True)
```

This Python code is creating a simple web application for making predictions using a trained machine learning model.

- It imports the necessary libraries like Flask for creating the web app, numpy for numerical processing, pickle for loading the trained model, and StandardScaler for data preprocessing.
- It creates a Flask app instance and loads the pretrained model and StandardScaler object from a pickle file called 'model.pkl'. This file contains the trained model object and fitted StandardScaler object we need to reuse.
- The '/' route renders a simple HTML template called 'index.html' to allow submitting input data.
- The '/predict' route handles making predictions. It takes in user input data, converts it to a numpy array, reshapes it to 2D for the model, applies the loaded StandardScaler to preprocess the data, and passes it to the model to make a prediction.
- Based on the prediction, it renders the same 'index.html' template but with an appropriate message for whether the customer would have bought the product or not.
- This allows creating a simple web interface where a user can submit data, it is preprocessed with the fitted StandardScaler, fed to the pretrained model, and a prediction is returned to the user.

HTML file for my Flask Webpage

```
model.py x app.py x index.html x main.py x
2   <html>
3     <!-- From https://codepen.io/frytyler/pen/EGdtg-->
4   <head>
5     <meta charset="UTF-8">
6     <title>Machine Learning API</title>
7     <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>
8     <link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>
9     <link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>
10    <link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' type='text/css'>
11    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}>
12
13  </head>
14
15  <body>
16    <div class="login">
17      <h1>Predict Product Purchase Analysis</h1>
18
19      <!-- Main Input For Receiving Query to our ML -->
20      <form action="{{ url_for('predict') }}" method="post">
21        <input type="text" name="Age" placeholder="Enter Age" required="required" />
22        <input type="text" name="EstimatedSalary" placeholder="Enter Estimated Salary" required="required" />
23
24        <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
25      </form>
26
27      <br>
28      <br>
29      {{ prediction_text }}
30
31    </div>
32
33  </body>
```

This HTML code creates a simple web form to allow users to input data and make predictions from the Machine Learning API created in the Python Flask code.

- It defines a basic HTML layout with title, CSS styling and font imports.
- The `<div>` with class "login" contains the main content.
- An `<h1>` tag displays the title "Predict Product Purchase Analysis".
- A HTML `<form>` is defined to send user input data to the "/predict" route of the Flask app on submit.
- There are two `<input>` fields to allow entering Age and EstimatedSalary values. These names match what the Flask code is expecting as input.
- A "Predict" button submits the form data.
- Below the form, the `{{prediction_text}}` variable will display the prediction result sent back from the Flask app.

So in summary, this simple HTML form allows a user to enter data, submit it to the API, and see the prediction result, creating an interactive web interface for the ML model. The Python Flask code handles the actual ML predictions based on the submitted data.

CSS File to beautify my webpage

This CSS code creates the styling and layout for the web form to make predictions. The CSS imports the Open Sans font for styling the text.

- It defines styling for the submit button - colors, shadows, hover effects etc. This will style the "Predict" button.
 - The .btn-primary class sets the main blue color and styling for the submit button.
 - It sets box-sizing to border-box for all elements for easier styling.
 - The html and body tags set up the overall page with dark background and centered text.
 - The .login class centers the form container on the page using positioning and negative margins.
 - The h1 tag displays the main heading "Predict Product Purchase Analysis".
 - The input tags create the form fields for Age and EstimatedSalary. They have styles for width, fonts, shadows etc.
 - The :focus pseudo-class adds effect when focusing on an input.

So in summary, the HTML constructs the basic form structure while the CSS stylizes and centers it to look nice on the page, creating the front-end for users to input data and see predictions.

Final Outcome

