Name : Varshit Manepalli

Batch Code : LISUM32

Submission Date : 26/04/2024

Submitted to : Data Glacier

Introduction

In this document, we outline the process of developing and deploying a machine learning model to predict the species of iris flowers based on their physical characteristics. The iris dataset, a classic in the field of machine learning, consists of 150 samples across three species, with four features each: sepal length, sepal width, petal length, and petal width. A logistic regression model, chosen for its efficacy in binary and multiclass classification problems, was trained using this dataset.

The goal of this project is not only to train a model with high predictive accuracy but also to deploy this model into a production-like environment using a Flask web application. This enables end-users to make predictions through a simple web interface. The Flask framework was chosen for its simplicity and efficiency in creating lightweight web services. By the end of this document, the reader will be familiar with the complete workflow which includes data preprocessing, model training, serialization of the trained model, and deployment using Flask within a virtual environment in Visual Studio Code.

The Model

Logistic regression is a statistical method for predicting binary classes. The outcome or target variable is dichotomous in nature (binary). Despite its name, logistic regression is used in classification problems, not regression problems. For the Iris dataset, which includes three species, logistic regression is extended to multinomial logistic regression to handle the multiple classes (multinomial logistic regression).

The model outputs probabilities of class memberships based on logistic function of the predictors. This is particularly useful in the Iris dataset, where the task is to classify an observation into one of three possible species (setosa, versicolor, or virginica) based on the physical measurements of the flower's sepals and petals. By training the logistic regression model on known examples of iris measurements, it learns the relationship between these measurements and the species, providing a mathematical framework for predicting the species of new iris observations based on these measurements.

```
data
```

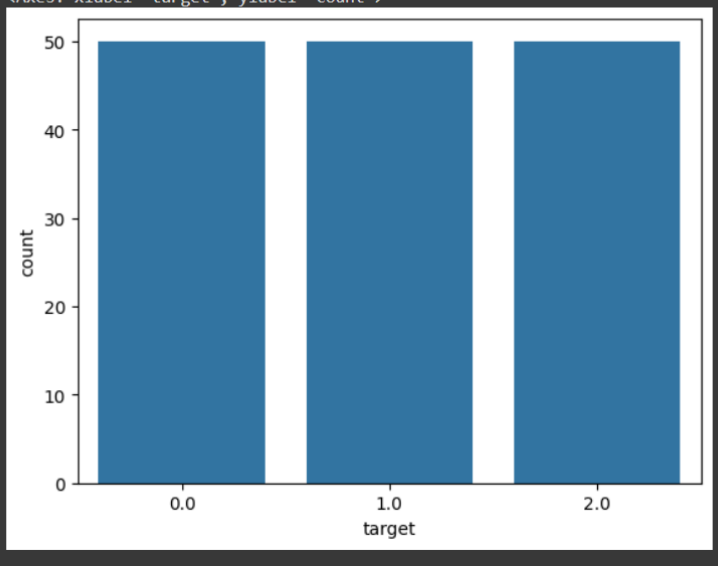
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2.0
146	6.3	2.5	5.0	1.9	2.0
147	6.5	3.0	5.2	2.0	2.0
148	6.2	3.4	5.4	2.3	2.0
149	5.9	3.0	5.1	1.8	2.0

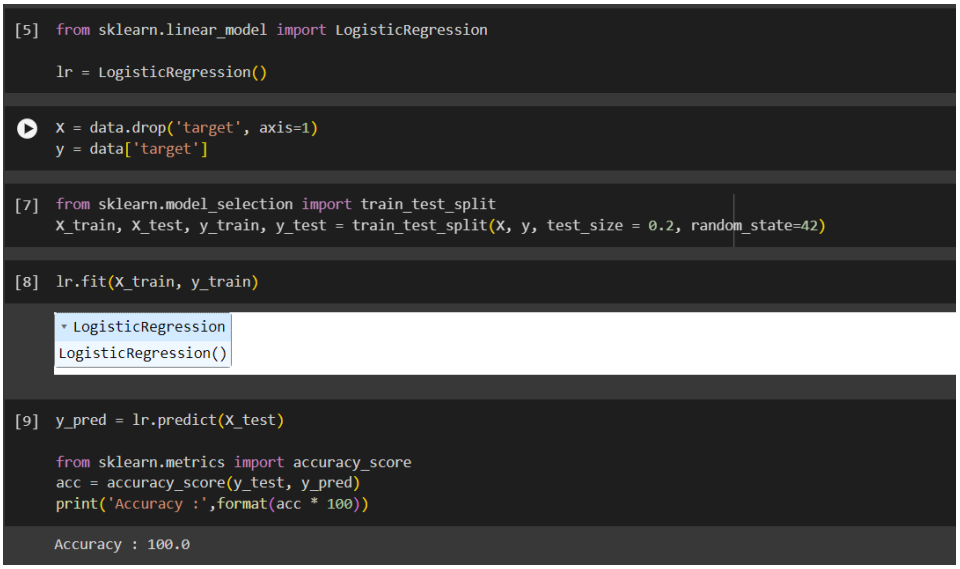
150 rows × 5 columns

The above picture is the sample from the iris data and the below picture shows the distribution of the target variable in the iris dataset.

```
import seaborn as sns
sns.countplot(x = 'target', data = data)
```

```
<Axes: xlabel='target', ylabel='count'>
```



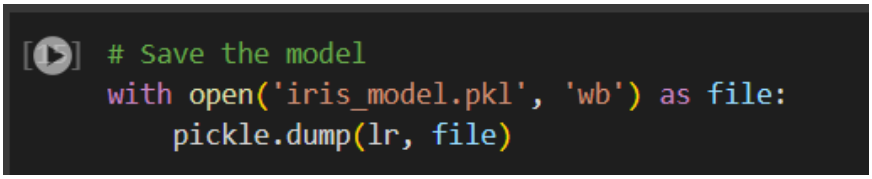


In the above picture it shows the process of training the logistic regression model for predicting the class in using the iris dataset.

Deployment

Deployment of the logistic regression model within a Flask web application involves a series of steps aimed at transforming a static machine learning model into a dynamic, interactive web service. Flask, a lightweight and powerful Python web framework, serves as the backbone for our application, providing the necessary tools to create and manage web routes. The deployment process begins with encapsulating the trained logistic regression model into a Pickle file, which serializes the Python object into a byte stream. This file can then be easily loaded within our Flask application, enabling it to make predictions in real-time based on user input.

Once the model is serialized and integrated into the Flask environment, the next step involves setting up the web server to handle incoming prediction requests. The application is designed with two primary routes: a homepage that provides general information about the service and a prediction endpoint where users can submit iris measurements via a JSON payload. When data is sent to the prediction endpoint, the Flask application processes this data, feeds it into the logistic regression model, and returns the predicted iris species. This setup not only demonstrates the practical application of machine learning models but also highlights the ease with which such models can be integrated into web applications, thereby making them accessible to a broader audience through standard web technologies.



We saved the model as a pickle file names ‘iris_model.pkl’. We use this model for deployment in the Flask.

```
PS D:\DataGlacier Internship\Week4> #Creating a new virtual environment
PS D:\DataGlacier Internship\Week4> python -m venv venv
PS D:\DataGlacier Internship\Week4> # Activate the virtual environment
>> .\venv\Scripts\activate
```

We set up a Python virtual environment for your project. This keeps our project dependencies separate from your global Python installation.

```
(venv) PS D:\DataGlacier Internship\Week4> #Installing Flask
(venv) PS D:\DataGlacier Internship\Week4> pip install flask scikit-learn numpy
Collecting flask
  Downloading flask-3.0.3-py3-none-any.whl.metadata (3.2 kB)
Collecting scikit-learn
  Downloading scikit_learn-1.4.2-cp312-cp312-win_amd64.whl.metadata (11 kB)
Collecting numpy
  Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl.metadata (61 kB)
     ---------------------------------------- 61.0/61.0 kB 806.3 kB/s eta 0:00:00
Collecting Werkzeug>=3.0.0 (from flask)
  Using cached werkzeug-3.0.2-py3-none-any.whl.metadata (4.1 kB)
Collecting Jinja2>=3.1.2 (from flask)
  Using cached Jinja2-3.1.3-py3-none-any.whl.metadata (3.3 kB)
Collecting itsdangerous>=2.1.2 (from flask)
  Downloading itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Collecting click>=8.1.3 (from flask)
  Using cached click-8.1.7-py3-none-any.whl.metadata (3.0 kB)
Collecting blinker>=1.6.2 (from flask)
  Using cached blinker-1.7.0-py3-none-any.whl.metadata (1.9 kB)
Collecting scipy>=1.6.0 (from scikit-learn)
  Downloading scipy-1.13.0-cp312-cp312-win_amd64.whl.metadata (60 kB)
     ---------------------------------------- 60.6/60.6 kB 3.1 MB/s eta 0:00:00
Collecting joblib>=1.2.0 (from scikit-learn)
  Downloading joblib-1.4.0-py3-none-any.whl.metadata (5.4 kB)
Collecting threadpoolctl>=2.0.0 (from scikit-learn)
  Downloading threadpoolctl-3.4.0-py3-none-any.whl.metadata (13 kB)
Collecting colorama (from click>=8.1.3->flask)
  Using cached colorama-0.4.6-py2.py3-none-any.whl.metadata (17 kB)
Collecting MarkupSafe>=2.0 (from Jinja2>=3.1.2->flask)
  Using cached MarkupSafe-2.1.5-cp312-cp312-win_amd64.whl.metadata (3.1 kB)
Downloading flask-3.0.3-py3-none-any.whl (101 kB)
   ---------------------------------------- 101.7/101.7 kB 3.0 MB/s eta 0:00:00
Downloading scikit_learn-1.4.2-cp312-cp312-win_amd64.whl (10.6 MB)
   ---------------------------------------- 10.6/10.6 MB 8.7 MB/s eta 0:00:00
Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl (15.5 MB)
   ---------------------------------------- 15.5/15.5 MB 10.2 MB/s eta 0:00:00
Using cached blinker-1.7.0-py3-none-any.whl (13 kB)
Using cached click-8.1.7-py3-none-any.whl (97 kB)
Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Using cached Jinja2-3.1.3-py3-none-any.whl (133 kB)
```

With our virtual environment activated, we now install Flask and any other necessary libraries, like numpy, scikit-learn, or others we have used in our model.

```
Downloading joblib-1.4.0-py3-none-any.whl (301 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 301.2/301.2 kB 9.4 MB/s eta 0:00:00
Downloading scipy-1.13.0-cp312-cp312-win_amd64.whl (45.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 45.9/45.9 MB 9.9 MB/s eta 0:00:00
Downloading threadpoolctl-3.4.0-py3-none-any.whl (17 kB)
Using cached werkzeug-3.0.2-py3-none-any.whl (226 kB)
Using cached MarkupSafe-2.1.5-cp312-cp312-win_amd64.whl (17 kB)
Using cached colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Installing collected packages: threadpoolctl, numpy, MarkupSafe, joblib, itsdangerous, colorama, blinker, Werkzeug, scipy, Jinja2, click, scikit-learn, flask
Successfully installed Jinja2-3.1.3 MarkupSafe-2.1.5 Werkzeug-3.0.2 blinker-1.7.0 click-8.1.7 colorama-0.4.6 flask-3.0.3 itsdangerous-2.2.0 joblib-1.4.0 numpy-1.26.4 scikit-learn-1.4.2
 scipy-1.13.0 threadpoolctl-3.4.0
```

In our app.py, we set up a simple Flask app. Below is the

code look like:

app.py

Week4 > app.py > ...

```python
from flask import Flask, request, jsonify
import pickle
import numpy as np

app = Flask(__name__)

# Load your trained model
model = pickle.load(open('iris_model.pkl', 'rb'))

@app.route('/')
def home():
    return "Welcome to the Iris Model API!"

@app.route('/predict', methods=['POST'])
def predict():
    # Expects data to be JSON with features in a list
    data = request.get_json(force=True)
    prediction = model.predict([np.array(data['features'])])
    return jsonify(prediction=int(prediction[0]))

if __name__ == '__main__':
    app.run(debug=True)
```

This code sets up two routes: one as a home page and another to make predictions using POST requests.

```
(venv) PS D:\DataGlacier Internship\Week4> # Set environment variables
>> export FLASK_APP=app.py
>> export FLASK_ENV=development
>>
>> # Run the Flask application
>> flask run
>>
```

We can run our Flask application directly from the command line in Visual Studio Code's terminal.

```
D:\DataGlacier Internship\Week4\venv\Lib\site-packages\sklearn\base.py:376: InconsistentVersionWarning: Trying to unpickle estimator LogisticRegression from version 1.2.2 when using ve
rsion 1.4.2. This might lead to breaking code or invalid results. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
D:\DataGlacier Internship\Week4\venv\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
127.0.0.1 - - [26/Apr/2024 23:12:19] "POST /predict HTTP/1.1" 200 -
D:\DataGlacier Internship\Week4\venv\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
127.0.0.1 - - [26/Apr/2024 23:12:23] "POST /predict HTTP/1.1" 200 -
D:\DataGlacier Internship\Week4\venv\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
127.0.0.1 - - [26/Apr/2024 23:12:56] "POST /predict HTTP/1.1" 200 -
D:\DataGlacier Internship\Week4\venv\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
127.0.0.1 - - [26/Apr/2024 23:14:49] "POST /predict HTTP/1.1" 200 -
```

This will start a development server, typically accessible at http://127.0.0.1:5000/ on your local machine.

Now we can test our Flask application using a tool like Postman or by using curl from the command line. Here's an example curl command to send a POST request:

```
D:\DataGlacier Internship\Week4>curl -X POST -H "Content-Type: application/json" -d "{\"features\":[1.2, 2.5, 3.2, 1.6]}" http://127.0.0.1:5000/predict
{"prediction":0}

D:\DataGlacier Internship\Week4>curl -X POST -H "Content-Type: application/json" -d "{\"features\":[6.7,3,5.2,2.3]}" http://127.0.0.1:5000/predict
{"prediction":2}
```

Conclusion

In conclusion, the deployment of the logistic regression model using Flask demonstrates a seamless integration of machine learning and web technology, enabling the practical application of predictive analytics. By converting the Iris dataset into a dynamic web service, we have provided a user-friendly interface that allows users to make real-time predictions. This project not only underscores the versatility of logistic regression in handling classification problems but also showcases Flask's capability to serve as an effective platform for deploying machine learning models. Future enhancements could include implementing more robust input validation, expanding the application's functionality with additional endpoints, or optimizing the model for better performance. The experience gained from this deployment provides a valuable foundation for future projects aiming to leverage machine learning in web environments, thereby bridging the gap between data science and application development.