ArogoAI Task Submission: Shipment Delay Prediction and Image Captioning Web Application

Srijan Anand, IIT Kanpur

December 21, 2024

Abstract

This is a submission for the tasks by ArogoAI. It consists of two tasks:

1. **Shipment Delay Prediction**: This task involved data analysis and binary classification for predicting shipment delays. I performed Exploratory Data Analysis (EDA) to fill in missing values optimally, reduced the dimensionality to two features, and employed techniques such as Artificial Neural Networks (ANN), Random Forest, and XGBoost for classification, achieving an accuracy of 91%. Additionally, I developed the corresponding API using Flask.
2. **Image Captioning Web Application**: For this web application, which accepts an image from the user and provides a written description, I utilized the `blip-image-captioning-base` model locally from HuggingFace. The front end was developed using HTML, CSS, and JavaScript, while Flask was used for the backend.

Contents

1 Shipment Delay Prediction

1.1 Data Analysis

The dataset provided was a dataframe containing the following columns:

- **Shipment ID**: Unique identifier for each shipment.
- **Origin**: Starting location of the shipment.
- **Destination**: Ending location of the shipment.
- **Shipment Date**: Date when the shipment was initiated.
- **Vehicle Type**: Type of vehicle used for the shipment.
- **Distance (km)**: Distance covered during the shipment.
- **Weather Conditions**: Weather conditions during the shipment.
- **Traffic Conditions**: Traffic conditions during the shipment.
- **Delayed**: Binary prediction variable indicating whether the shipment was delayed (1) or on time (0).

```
df = pd.read_excel('AI ML Internship Training Data.xlsx')
df.head(10)
```

Python

	Shipment ID	Origin	Destination	Shipment Date	Planned Delivery Date	Actual Delivery Date	Vehicle Type	Distance (km)	Weather Conditions	Traffic Conditions	Delayed
0	SHIP000000	Jaipur	Mumbai	2023-04-26	2023-05-01	2023-05-02	Trailer	1603	Rain	Light	Yes
1	SHIP000001	Bangalore	Delhi	2023-02-09	2023-02-13	2023-02-17	Trailer	1237	Storm	Moderate	Yes
2	SHIP000002	Mumbai	Chennai	2023-09-19	2023-09-25	2023-09-25	Truck	1863	Clear	Light	No
3	SHIP000003	Hyderabad	Ahmedabad	2023-04-01	2023-04-05	2023-04-05	Container	1374	Clear	Light	No
4	SHIP000004	Chennai	Kolkata	2023-11-24	2023-11-26	2023-11-28	Container	676	Clear	Heavy	Yes
5	SHIP000005	Chennai	Ahmedabad	2024-03-14	2024-03-16	2024-03-19	Container	746	Storm	Light	Yes
6	SHIP000006	Bangalore	Lucknow	2024-05-28	2024-06-03	2024-06-10	Lorry	1988	Fog	Light	Yes
7	SHIP000007	Kolkata	Mumbai	2023-04-01	2023-04-06	2023-04-08	Container	1521	Rain	Moderate	Yes
8	SHIP000008	Kolkata	Bangalore	2023-10-01	2023-10-02	2023-10-03	Truck	390	Rain	Light	Yes
9	SHIP000009	Lucknow	Ahmedabad	2023-03-08	2023-03-09	2023-03-09	Trailer	324	Clear	Light	No

Figure 1: Sample of the Shipment Delay Prediction Dataset

I conducted a null analysis and discovered that 597 out of 20,000 rows were missing values in the `Vehicle Type` column. To address this, I performed various analyses by grouping the data with multiple other features. However, since the distributions were very uniform, it was challenging to handle the missing `Vehicle Type` values effectively. Upon deeper investigation, I hypothesized that vehicle selection is generally influenced by the journey specifics (from origin to destination). Therefore, I grouped the vehicles based on the route and assigned the most frequently used vehicle type for each origin-destination pair to handle the missing values. This approach was adopted because the data was evenly distributed within these groups, making it a viable solution.

Next, to reduce dimensionality, I identified that not all features significantly contributed to delay prediction. Specifically:

- **Shipment ID**: This was a unique identifier with no predictive power regarding delays.
- **Distance (km)**: The delay was evenly distributed across various distances, making it a less effective predictor.

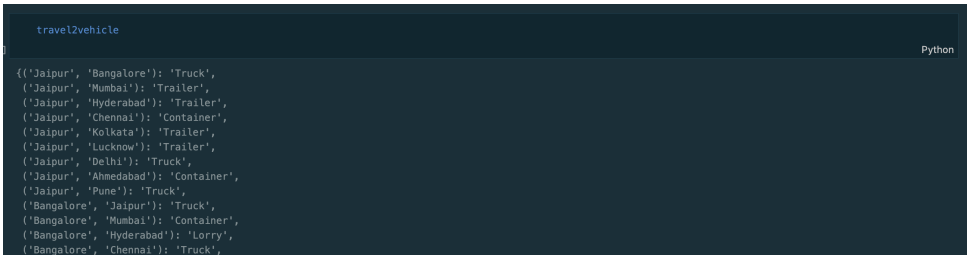
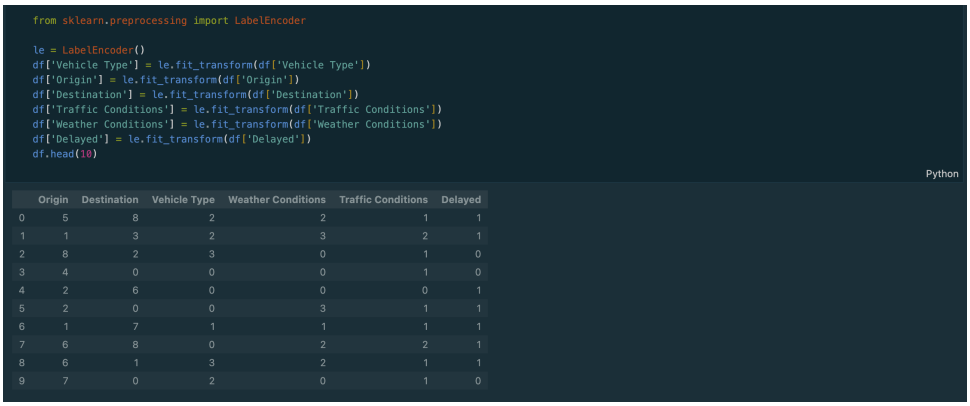


Figure 2: Vehicle Type Imputation Based on Origin-Destination Pair

- **Shipment Date**: Dates did not provide substantial insights into delays.

Consequently, I removed these columns from the dataset.

The remaining categorical features were encoded using Label Encoding to convert textual data into numerical form, facilitating the training of machine learning models.



	Origin	Destination	Vehicle Type	Weather Conditions	Traffic Conditions	Delayed
0	5	8	2	2	1	1
1	1	3	2	3	2	1
2	8	2	3	0	1	0
3	4	0	0	0	1	0
4	2	6	0	0	0	1
5	2	0	0	3	1	1
6	1	7	1	1	1	1
7	6	8	0	2	2	1
8	6	1	3	2	1	1
9	7	0	2	0	1	0

Figure 3: Label Encoding of Categorical Features

Initially, I retained the features **Origin**, **Destination**, and **Vehicle Type** and trained models on them. The results were decent, primarily because this step ensured that the NaN values were appropriately handled. However, further analysis revealed that these features were not strong predictors of shipment delays, as their distribution was also quite uniform. Therefore, I dropped these features and retrained the models using only two features: **Weather Conditions** and **Traffic Conditions**. This simplification led to improved model performance.

1.2 Modeling Approach

I experimented with three different models: an Artificial Neural Network (ANN), a Random Forest, and XGBoost.

- **Random Forest**: Selected for its robustness and common usage in classification problems.
- **XGBoost**: Chosen to incorporate gradient boosting techniques, enhancing model performance.
- **Artificial Neural Network (ANN)**: Implemented to leverage deep learning capabilities for classification.

All three models achieved similar results, demonstrating the effectiveness of the feature selection process.

The final results using only two features (**Weather Conditions** and **Traffic Conditions**) with a 20% test-train split are presented in Table 1.

Model	Test Accuracy	F1-Score (Macro Avg)
Neural Network	0.91	0.90
Random Forest	0.91	0.90
XGBoost	0.91	0.90

Table 1: Classification Results of Different Models

1.2.1 Neural Network

```
Test Accuracy: 0.91

Classification Report:
              precision    recall  f1-score   support

           0       0.75      1.00      0.86      1059
           1       1.00      0.88      0.94      2941

    accuracy                           0.91      4000
   macro avg       0.87      0.94      0.90      4000
weighted avg       0.93      0.91      0.92      4000
```

Listing 1: Neural Network Classification Report

1.2.2 Random Forest

```
Test Accuracy: 0.91

Classification Report:
              precision    recall  f1-score   support

           0       0.75      1.00      0.86      1059
           1       1.00      0.88      0.94      2941

    accuracy                           0.91      4000
   macro avg       0.87      0.94      0.90      4000
weighted avg       0.93      0.91      0.92      4000
```

Listing 2: Random Forest Classification Report

1.2.3 XGBoost

```
Test Accuracy: 0.91

Classification Report:
              precision    recall  f1-score   support

           0       0.75      1.00      0.86      1059
```

```
           1       1.00      0.88      0.94      2941

    accuracy                           0.91      4000
   macro avg       0.87      0.94      0.90      4000
weighted avg       0.93      0.91      0.92      4000
```

Listing 3: XGBoost Classification Report

1.3 API Development

For deploying the shipment delay prediction models, an API was developed using Flask. The API accepts input data in JSON format, processes it, and returns predictions from all three models.

```
app = Flask(__name__)

mp = {
    "Traffic Conditions": {'Heavy': 0, 'Light': 1, 'Moderate': 2},
    "Weather Conditions": {'Rain': 2, 'Storm': 3, 'Clear': 0, 'Fog': 1}
}

rf_model = joblib.load('rf_model.pkl')
xgb_model = joblib.load('xgb_model.pkl')
nn_model = load_model('nn_model.h5')

@app.route('/predict', methods=['POST'])
def predict():
    try:
        data = request.json
        features = ['Weather Conditions', 'Traffic Conditions']
        for field in features:
            if field not in data:
                return jsonify({'error': f'Missing field: {field}'}),
    400
        input_data = pd.DataFrame([{field: mp[field][data[field]] for
    field in features}])

        rf_prediction = rf_model.predict(input_data)[0]
        xgb_prediction = xgb_model.predict(input_data)[0]
        nn_prediction = (nn_model.predict(input_data)[0][0] > 0.5).
    astype(int)

        response = {
            'RandomForest': 'Delayed' if rf_prediction == 1 else 'On
    Time',
            'XGBoost': 'Delayed' if xgb_prediction == 1 else 'On Time',
            'NeuralNetwork': 'Delayed' if nn_prediction == 1 else 'On
    Time'
        }
        return jsonify(response)

    except Exception as e:
        return jsonify({'error': str(e)})

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

Listing 4: Flask API Implementation

This API accepts a JSON payload and returns the prediction results from all three models. To use this API, while it is running, you can perform a POST request using another Python program as shown below:

```python
import requests

url = "http://127.0.0.1:5000/predict"
payload = {
    "Weather Conditions": "Clear",
    "Traffic Conditions": "Moderate"
}

response = requests.post(url, json=payload)
print(response.json())
```

Listing 5: Example of Using the Flask API

Executing the above code will send a POST request to the API and receive a JSON response indicating the delay predictions from each model.

Figure 4: Example API Response

2 Image Captioning Web Application

2.1 Introduction

The problem statement for this task was to create a web application that allows users to upload an image and receive a text description of its content. I utilized an open-source `blip-image-captioning-base` model by Hugging Face, which generates descriptive text based on the provided image. The backend was developed using Python with Flask, while the frontend was built using a combination of HTML, CSS, and JavaScript to ensure a user-friendly interface.

2.2 Model Selection

Selecting an appropriate model was crucial for this task. I aimed to choose a lightweight model that not only functions as an image classifier but also provides generative AI output to deliver comprehensive descriptions of images. After evaluating several options, I selected the `blip-image-captioning-base` model. This model is approximately 900MB in size and effectively generates detailed captions, making it an optimal choice for balancing performance and resource requirements.

2.3 Frontend Development

As I had limited experience with frontend development, possessing only basic knowledge of HTML and CSS, I sought assistance from online resources and ChatGPT to create a

user-friendly webpage. The goal was to design an intuitive interface where users can easily upload images and view the generated descriptions. The frontend consists of a simple button for image upload, and a generate caption button, it also previews the uploaded image and generates caption below it (figure 6).

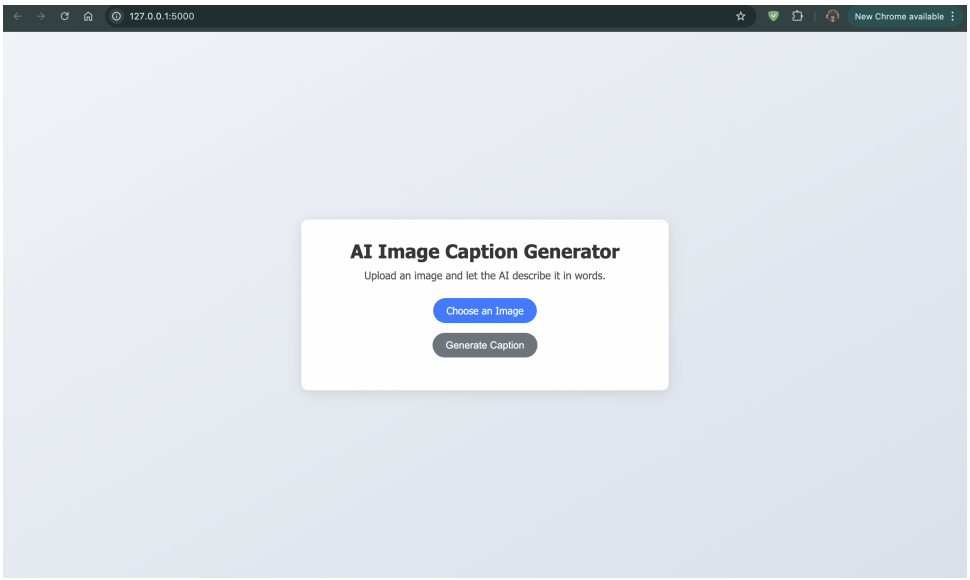


Figure 5: User-Friendly Frontend Interface for Image Captioning Web Application

2.4 Backend Development

The backend of the application was developed using Flask in Python. It handles image uploads, processes the images using the BLIP model, and returns the generated captions. Below is a snippet of the backend code implemented in ‘app.py‘:

```
import os
import io
from PIL import Image
from flask import Flask, request, jsonify, render_template

import torch
from transformers import BlipProcessor, BlipForConditionalGeneration

app = Flask(__name__)

device = "cuda" if torch.cuda.is_available() else "cpu"

model_name = "salesforce/blip-image-captioning-base"
processor = BlipProcessor.from_pretrained(model_name)
model = BlipForConditionalGeneration.from_pretrained(model_name).to(
    device)


def generate_caption(image_bytes):
    """
    Generate a descriptive caption for the given image bytes using the
    BLIP model.
    """
    image = Image.open(io.BytesIO(image_bytes)).convert('RGB')
```

```
    inputs = processor(images=image, return_tensors="pt").to(device)
    with torch.no_grad():
        out = model.generate(**inputs, max_length=50, num_beams=5,
    early_stopping=True)
    caption = processor.tokenizer.decode(out[0], skip_special_tokens=
    True)
    return caption


@app.route("/", methods=["GET"])
def index():
    return render_template("index.html")


@app.route("/api/describe", methods=["POST"])
def describe_image():
    if 'image' not in request.files:
        return jsonify({"error": "No image provided"}), 400

    file = request.files['image']
    image_bytes = file.read()

    try:
        _ = Image.open(io.BytesIO(image_bytes))
    except Exception:
        return jsonify({"error": "Invalid image file."}), 400

    try:
        caption = generate_caption(image_bytes)
    except Exception as e:
        print(e)
        return jsonify({"error": "Error generating description."}), 500

    return jsonify({"description": caption}), 200


if __name__ == "__main__":
    app.run(host='0.0.0.0', port=int(os.environ.get('PORT', 5000)),
    debug=True)
```

Listing 6: Flask Backend Implementation for Image Captioning

2.5 Deployment and Testing

To deploy and test the application, I organized the project directory with the following structure:

- `templates/`: Contains the HTML files.
- `static/`: Contains CSS and JavaScript files.
- `app.py`: The main Flask application script.

I ran the 'app.py' script to start the Flask server locally. The application interface is intuitive, allowing users to upload images and receive accurate textual descriptions. Below is an example of the application's interface and its functionality.

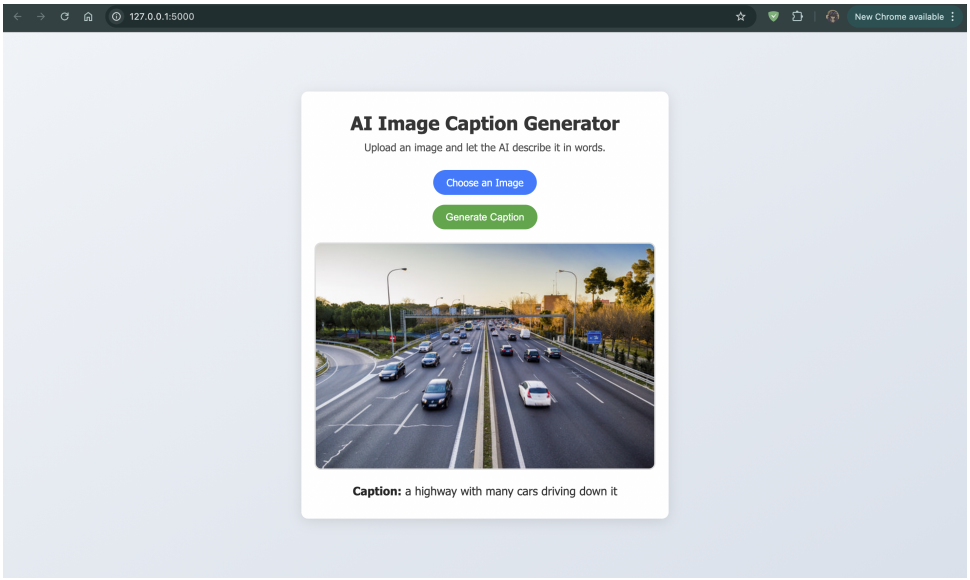


Figure 6: Deployment Interface of the Image Captioning Web Application

The application successfully generates descriptive captions for uploaded images, demonstrating both functionality and user-friendliness.

3 Thank You

It was a pleasure and a rewarding experience to work on these tasks. Thank You.