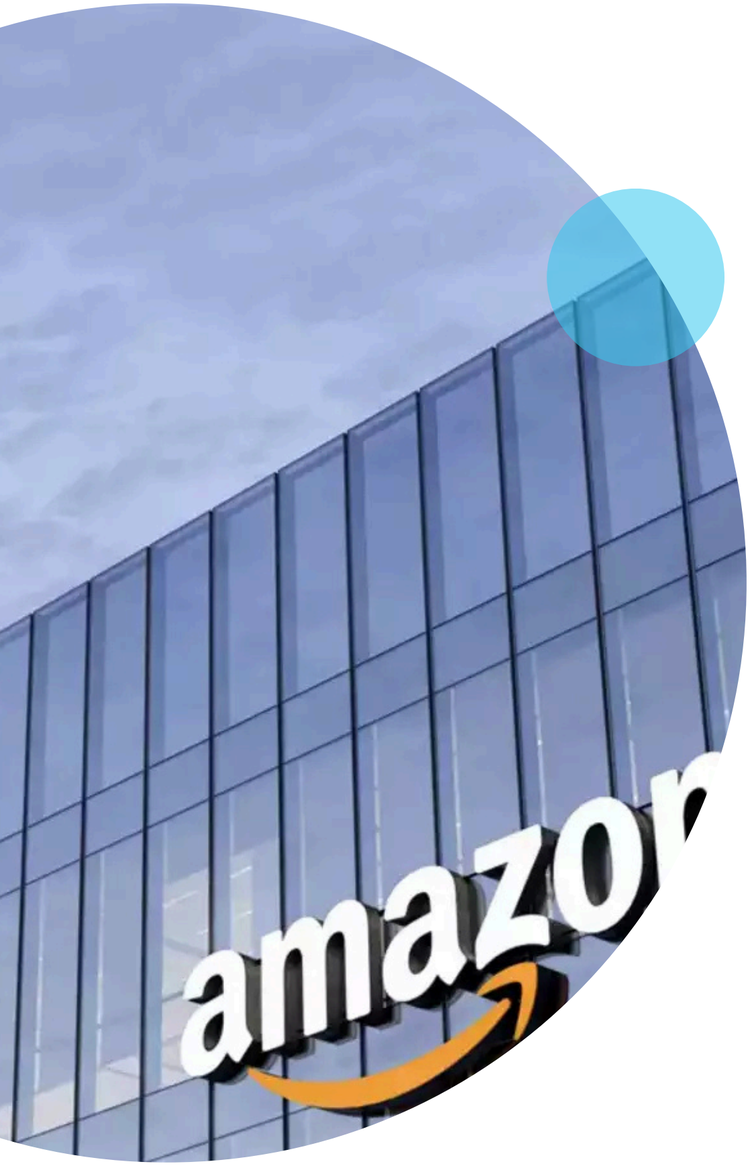


Aspect-Based Sentiment Analysis

Sentiment Analysis with a Twist

Dataset source: *Amazon Product Reviews Dataset*

HOW IS IT DIFFERENT?

Why Aspect-Based Sentiment Analysis?

1

Aspect-Based Analysis
Goes beyond generic "positive" or "negative" sentiment

2

Fine-Grained Sentiment Classification:
Classifies sentiment into detailed levels like star ratings.

3

Handling Imbalanced Data:
Uses oversampling to address class imbalance effectively.

4

State-of-the-Art Model (RoBERTa): Leverages RoBERTa for contextual understanding of reviews.

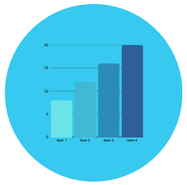
5

Custom Weighting for Class Imbalance: Applies class weights to improve minority class predictions.

6

End-to-End Pipeline: Covers preprocessing, fine-tuning, and advanced metric evaluation.

Dataset Overview



reviews.text

Customer review text

reviews.rating

Customer rating (1–5)

Preprocessing Steps:

a. Removed missing values
b. Encoded ratings into numeric labels
c. Resolved class imbalance using RandomOverSampler

Makes an impact

Use visual charts to communicate info more effectively.

Amazon Customer Reviews

TOOLS & TECHNOLOGY:

01

Model & Frameworks:

Hugging Face Transformers: RoBERTa-base, PyTorch for model fine-tuning

02

Data Handling:

pandas, numpy, imblearn (oversampling)

03

Evaluation Metrics::

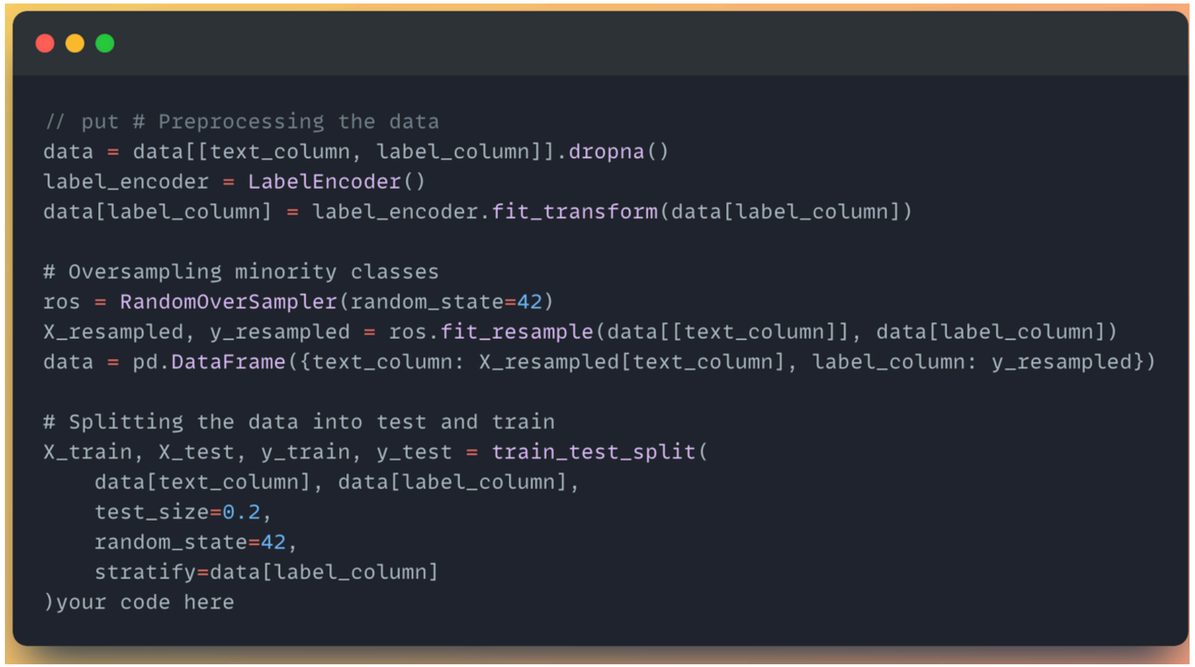
F1 Score, Precision, Recall

Dataset Preprocessing

Objective: Prepare the dataset for effective training and evaluation.

Key Steps:

- **Remove null values and retain relevant columns.**
- **Encode labels using LabelEncoder.**
- **Handle class imbalance using oversampling (RandomOverSampler).**
- **Split the dataset into training and test sets using train_test_split.**



```
// put # Preprocessing the data
data = data[[text_column, label_column]].dropna()
label_encoder = LabelEncoder()
data[label_column] = label_encoder.fit_transform(data[label_column])

# Oversampling minority classes
ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(data[[text_column]], data[label_column])
data = pd.DataFrame({text_column: X_resampled[text_column], label_column: y_resampled})

# Splitting the data into test and train
X_train, X_test, y_train, y_test = train_test_split(
    data[text_column], data[label_column],
    test_size=0.2,
    random_state=42,
    stratify=data[label_column]
)your code here
```

Model Selection & Tokenization

Objective: Leverage RoBERTa for aspect-based sentiment analysis.

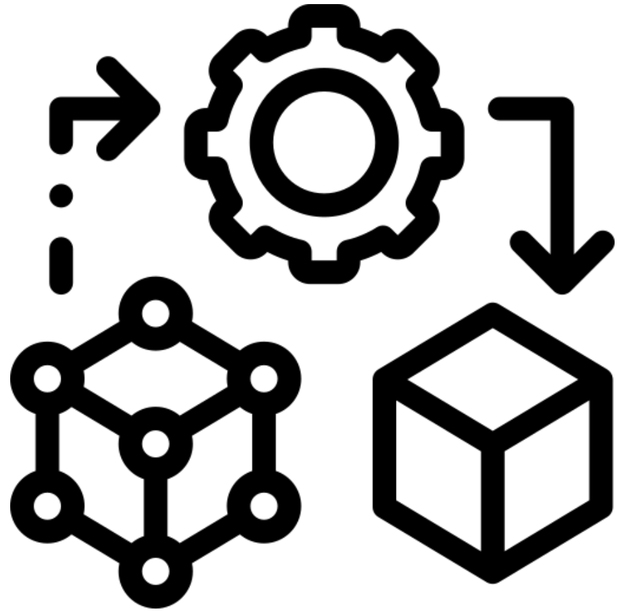
Key Steps:

- **Load the pre-trained RoBERTa model and tokenizer.**
- **Define a tokenization function to prepare text inputs for the model.**

Tool: Hugging Face Transformers.

```
# Load model and tokenizer
model_name = "roberta-base"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(
    model_name,
    num_labels=len(label_encoder.classes_)
)

# Tokenization function
def tokenize_function(examples):
    return tokenizer(
        examples['text'],
        padding="max_length",
        truncation=True,
        max_length=256,
        return_tensors="pt"
    )
```



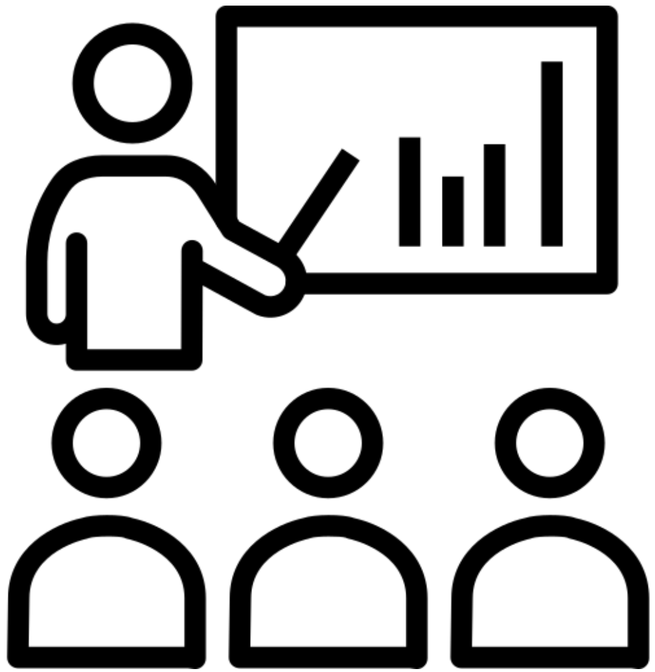
Custom Training with Weighted Loss

Objective: Address class imbalance with a weighted loss function.

Key Steps:

- Implement a custom **Trainer** class.
- Define a weighted cross-entropy loss function.
- Integrate early stopping for optimized training.

```python
class WeightedTrainer(Trainer):
    def compute_loss(self, model, inputs, return_outputs=False, **kwargs):
        labels = inputs.get("labels")
        outputs = model(**inputs)
        logits = outputs.get("logits")
        loss_fct = torch.nn.CrossEntropyLoss(weight=class_weights.to(model.device))
        loss = loss_fct(logits.view(-1, self.model.config.num_labels), labels.view(-1))
        return (loss, outputs) if return_outputs else loss
```

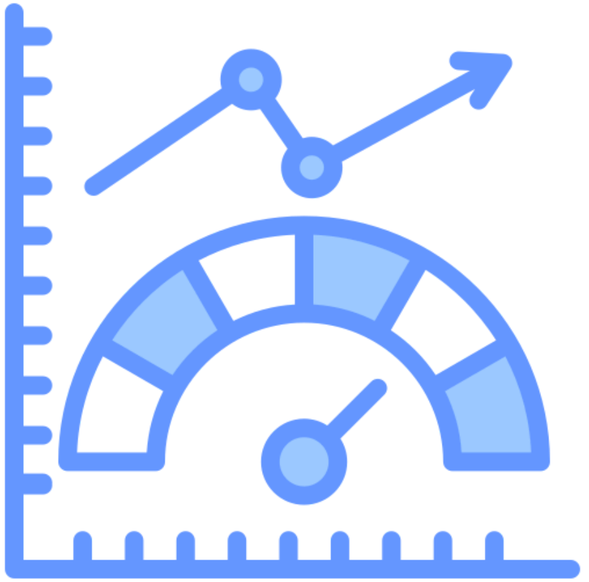



Evaluation & Metrics

Objective: Measure model performance on unseen data.

Metrics:

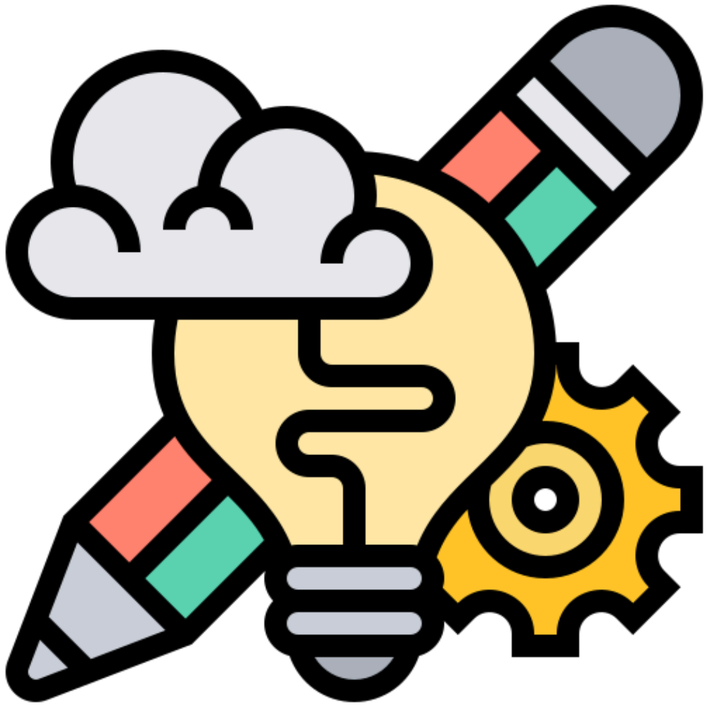
- Macro F1 Score, Precision, Recall.
- Weighted F1 Score.
- Classification Report.



```python
# Metrics calculation
def compute_metrics(p):
    preds = np.argmax(p.predictions, axis=1)
    return {
        'macro_f1': f1_score(p.label_ids, preds, average='macro'),
        'macro_precision': precision_score(p.label_ids, preds, average='macro'),
        'macro_recall': recall_score(p.label_ids, preds, average='macro'),
        'weighted_f1': f1_score(p.label_ids, preds, average='weighted')
    }

# Final evaluation
predictions = trainer.predict(tokenized_test)
preds = np.argmax(predictions.predictions, axis=1)
print(classification_report(y_test, preds, digits=4))
```

CHALLENGES & SOLUTIONS



Challenges:

Imbalanced Dataset: Applied oversampling using RandomOverSampler.

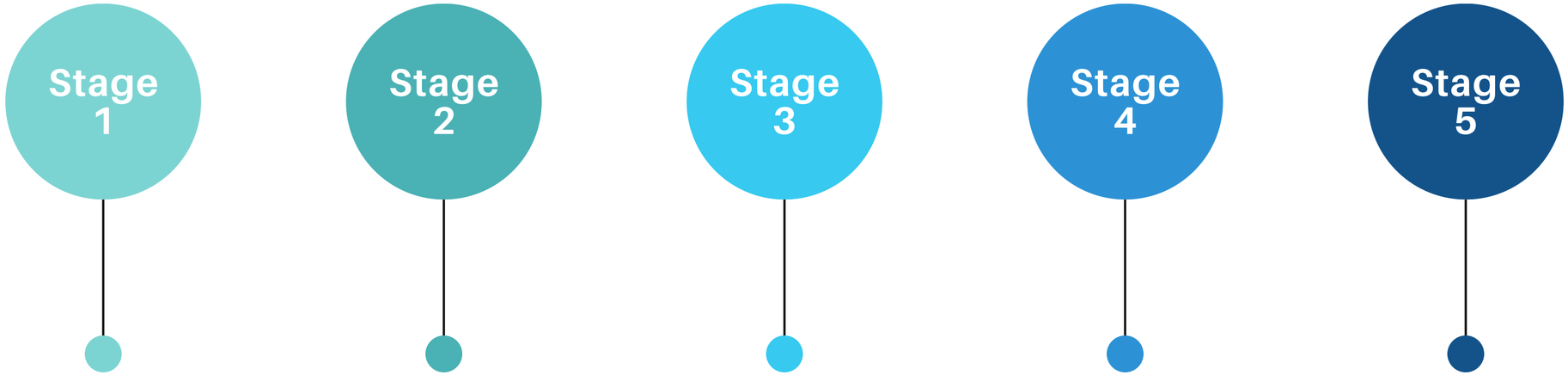
Ambiguity in Text Reviews

Solution: Used RoBERTa for contextual understanding.

Computational Overhead

Optimized batch sizes and used early stopping.

CONCLUSION & FUTURE WORK



Successfully implemented sentiment analysis using RoBERTa.

Improved classification performance with custom loss and resampling.

Expand to Aspect-Based Sentiment Analysis (ABSA) for feature-level insights.

Add multilingual support using models like mBERT.

Deploy a real-time Streamlit/Flask app for business use.