Employee's Sentiment Analysis Chatbot Report

RoBERTa Model Trainning

RoBERTa (Robustly optimized BERT approach) is a advanced version of the BERT model developed by Facebook AI. It is a pre-trained transformer-based language model designed for natural language understanding tasks such as sentiment analysis, text classification, question-answering and many more. It is excellent in classifying positive, Negative and Neutral sentiments with high quality contextual understanding in short and informal text. Further it can be improved with fine-tuning to learn domain-specific patterns. But other than that this model needs large and balanced dataset for under labels (no-match). Fine-tuning can lead to accuracy drop slightly, if dataset is noisy or imbalanced.

Attempt – 1 : Using Pre-trained RoBERTa (No fine-tuning)

This model show high performance on traditional sentiment classes (Positive, Neutral, Negative). I implemented a sentiment classification model trained on 3 sentiment classes. Although it was not trained for the “No-Match” category, but I still tested its performance across all 4-classes. After that this model achieve an excellent overall accuracy of 97.98%, but it fail to predict any of the “No-Match” class. This means that while the model is highly accurate for sentiment feedback, it lacks the ability to differentiate feedback that does not carry any sentiment, emotions or is irrelevant. This failure is likely because the pre-trained model was originally designed for 3-class sentiments task.

	Predicted Negative	Predicted Neutral	Predicted Positive	Predicted No Match
Actual Negative	282	0	0	0
Actual Neutral	0	157	0	0
Actual Positive	0	0	290	0
Actual No Match	0	0	0	0

The model meet 5 out of the 6 requirements, it completely failed the “No-Match” criteria that is too bad for classification without further adaptation. This inability can reduce the precision analysis and may be misguide the decision making in practical. Therefore it, is not recommended to use this model for 4-class classification problem , instead I fine-tuned this model on my domain-specific dataset that includes label examples of all 4-classes. This may be give better contextual understanding and improve performance on all categories, until I give me second attempt, The model still cannot be the considered best solution for full sentiment and relevance detection.

Criterion Description	Expected Value	Actual Value	Status
Positive Recall ≥ 0.96	≥ 0.96	1.000	Passed
Neutral Recall ≥ 0.56	≥ 0.56	1.000	Passed
Negative Recall ≥ 0.646	≥ 0.646	1.000	Passed
No Match Recall = 1.0	= 1.0	0.000	Failed
Positive → Negative Misclassification = 0	= 0	0	Passed
Negative → Positive Misclassification ≤ 8	≤ 8	0	Passed

CODE OUTPUT :

Calculating the overall accuracy of model

```
[ ]  # calculate overall accuracy
     accuracy = accuracy_score(df["Label"], df["Predicted_Sentiment"])
     print(f"OVERALL MODEL ACCURACY: {accuracy * 100:.2f}%")
```

```
OVERALL MODEL ACCURACY: 97.98%
```

```
[ ]      showscale=True
     )

     fig.update_layout(
         title="Multi-Class Sentiment Detection ",
         xaxis_title="Predicted Label",
         yaxis_title="Label"
     )

     fig.show()
```

```
-------------------------CONFUSION MATRIX HEATMAP----------------------

[[290   0   0   0]
 [  0 157   0   0]
 [  0   0 282   0]
 [  0   0   0   0]]
```

```
[ ]     print(f"{label:10s}: {recall:.3f}")

    # misclassification checks
    pos_as_neg = cm[labels.index("Positive")][labels.index("Negative")]
    neg_as_pos = cm[labels.index("Negative")][labels.index("Positive")]

    print("Positive → Negative Misclassifications:", pos_as_neg)
    print("Negative → Positive Misclassifications:", neg_as_pos)
```

```
-------------------------PER CLASS BREAKDOWN--------------------------

Positive  : 1.000
Neutral   : 1.000
Negative  : 1.000
No Match  : 0.000
Positive → Negative Misclassifications: 0
Negative → Positive Misclassifications: 0
```

```
    "Positive→Negative = 0": pos_as_neg == 0,
    "Negative→Positive ≤ 8": neg_as_pos <= 8
}

print("\n-------------------------EVALUATION CRITERIA CHECK---------------------\n")
for k, passed in criteria.items():
    status = " Passed" if passed else  "Failed"
    print(f"{k:<30} {status}")
```

```
-------------------------EVALUATION CRITERIA CHECK---------------------

Positive ≥ 0.96                Passed
Neutral ≥ 0.56                 Passed
Negative ≥ 0.646               Passed
No Match = 1.0                 Failed
Positive→Negative = 0          Passed
Negative→Positive ≤ 8          Passed
```

Attempt – 2 : Fine-Tuned RoBERTa on 4-Class Sentiment Task

In this attempt my main goal is to accurately classify the feedback into 4-categories using LLM. To solve the above issue , I fine-tune the same model using domain-specific data that all 4-labels so that it can learn “No Match” class and perform better than other. I train the model over 5 epochs and with each epoch , validation accuracy improve consistently. In despite of any fluctuations, training was stable and the model show strong learning process. After completing training , I evaluate the model on the test data and result showed some improvements as compare to first attempt. Now the model meet 4 out of 6 criteria and give a strong 4-class solution overall. The overall accuracy of this model is lower than the previous attempt(97.98%) but the previous model entirely failed to detect one class. This model show real learning for all 4-categories and a small drop in accuracy(89.39%) for much more complete classification.

Epoch	Training Loss	Validation Loss	Accuracy	Positive Recall	Negative Recall	Neutral Recall	No Match Recall
1	0.673	0.634	75.17%	0.964	0.980	0.216	0.000
2	0.428	0.506	82.55%	0.946	0.960	0.567	0.000
3	0.381	0.598	78.52%	0.946	1.000	0.351	0.000
4	0.245	0.562	81.87%	0.946	1.000	0.486	0.000
5	0.213	0.641	81.21%	0.946	1.000	0.459	0.000

Final Class-wise Recall Breakdown

Label	Recall	Passed Threshold
Positive	0.972	(≥ 0.96)
Negative	0.947	(≥ 0.646)
Neutral	0.707	(≥ 0.56)
No Match	0.333	(= 1.0)

Evaluation Criteria Result

Criterion	Status
Positive Recall ≥ 0.96	Passed
Neutral Recall ≥ 0.56	Passed
Negative Recall ≥ 0.646	Passed
No Match Recall = 1.0	Failed
Positive → Negative Misclassifications = 0	Failed (3 found)
Negative → Positive Misclassifications ≤ 8	Passed (1 found)

CODE OUTPUT :

```
[ ]     args=training_args,
        train_dataset=train_dataset,
        eval_dataset=eval_dataset,
        compute_metrics=compute_metrics,
    )

    # train the model
    trainer.train()
```

```
Some weights of RobertaForSequenceClassification were not initialized from the model check
- classifier.out_proj.weight: found shape torch.Size([3, 768]) in the checkpoint and torch
- classifier.out_proj.bias: found shape torch.Size([3]) in the checkpoint and torch.Size([
You should probably TRAIN this model on a down-stream task to be able to use it for predic
Map: 100% 744/744 [00:00<00:00, 2982.20 examples/s]
[190/190 50:22, Epoch 5/5]
```

```
- classifier.out_proj.weight: found shape torch.Size([3, 768]) in the checkpoint and torch.Size([4, 768]) in the model instantiated
- classifier.out_proj.bias: found shape torch.Size([3]) in the checkpoint and torch.Size([4]) in the model instantiated
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Map: 100% 744/744 [00:00<00:00, 2982.20 examples/s]
[190/190 50:22, Epoch 5/5]
```

Epoch	Training Loss	Validation Loss	Accuracy	Positive Recall	Negative Recall	Neutral Recall	No Match Recall
1	0.673100	0.633913	0.751678	0.964286	0.980392	0.216216	0.000000
2	0.428400	0.505725	0.825503	0.946429	0.960784	0.567568	0.000000
3	0.381400	0.598435	0.785235	0.946429	1.000000	0.351351	0.000000
4	0.245400	0.562445	0.818792	0.946429	1.000000	0.486486	0.000000
5	0.212700	0.641171	0.812081	0.946429	1.000000	0.459459	0.000000

```
[190/190 52:00, Epoch 5/5]
```

3	0.381400	0.598435	0.785235	0.946429	1.000000	0.351351	0.000000
4	0.245400	0.562445	0.818792	0.946429	1.000000	0.486486	0.000000
5	0.212700	0.641171	0.812081	0.946429	1.000000	0.459459	0.000000

```
[190/190 52:00, Epoch 5/5]
```

Epoch	Training Loss	Validation Loss	Accuracy	Positive Recall	Negative Recall	Neutral Recall	No Match Recall
1	0.673100	0.633913	0.751678	0.964286	0.980392	0.216216	0.000000
2	0.428400	0.505725	0.825503	0.946429	0.960784	0.567568	0.000000
3	0.381400	0.598435	0.785235	0.946429	1.000000	0.351351	0.000000
4	0.245400	0.562445	0.818792	0.946429	1.000000	0.486486	0.000000
5	0.212700	0.641171	0.812081	0.946429	1.000000	0.459459	0.000000

```
TrainOutput(global_step=190, training_loss=0.4009590048539011, metrics={'train_runtime': 3141.6533, 'train_samples_per_second': 0.947,
'train_steps_per_second': 0.06, 'total_flos': 195692361446400.0, 'train_loss': 0.4009590048539011, 'epoch': 5.0})
```

```
fig.update_layout(
    title="Multi-Class Sentiment Detection",
    xaxis_title="Predicted Label",
    yaxis_title="Label",
)
fig.show()
```

```
-------------------------PER CLASS BREAKDOWN---------------------------

Positive  : 0.972
Negative  : 0.947
Neutral   : 0.707
No Match  : 0.333
Positive → Negative Misclassifications: 3
Negative → Positive Misclassifications: 1
```

```
-------------------------EVALUATION CRITERIA CHECK--------------------

Positive ≥ 0.96               Passed
Neutral ≥ 0.56                Passed
Negative ≥ 0.646              Passed
No Match = 1.0                Failed
Positive→Negative = 0         Failed
Negative→Positive ≤ 8         Passed

-------------------------OVERALL MODEL ACCURACY: 89.38% ---------------
Predictions saved to: /content/Output_Dataset_3.csv
```

Challenges Faced By Me :

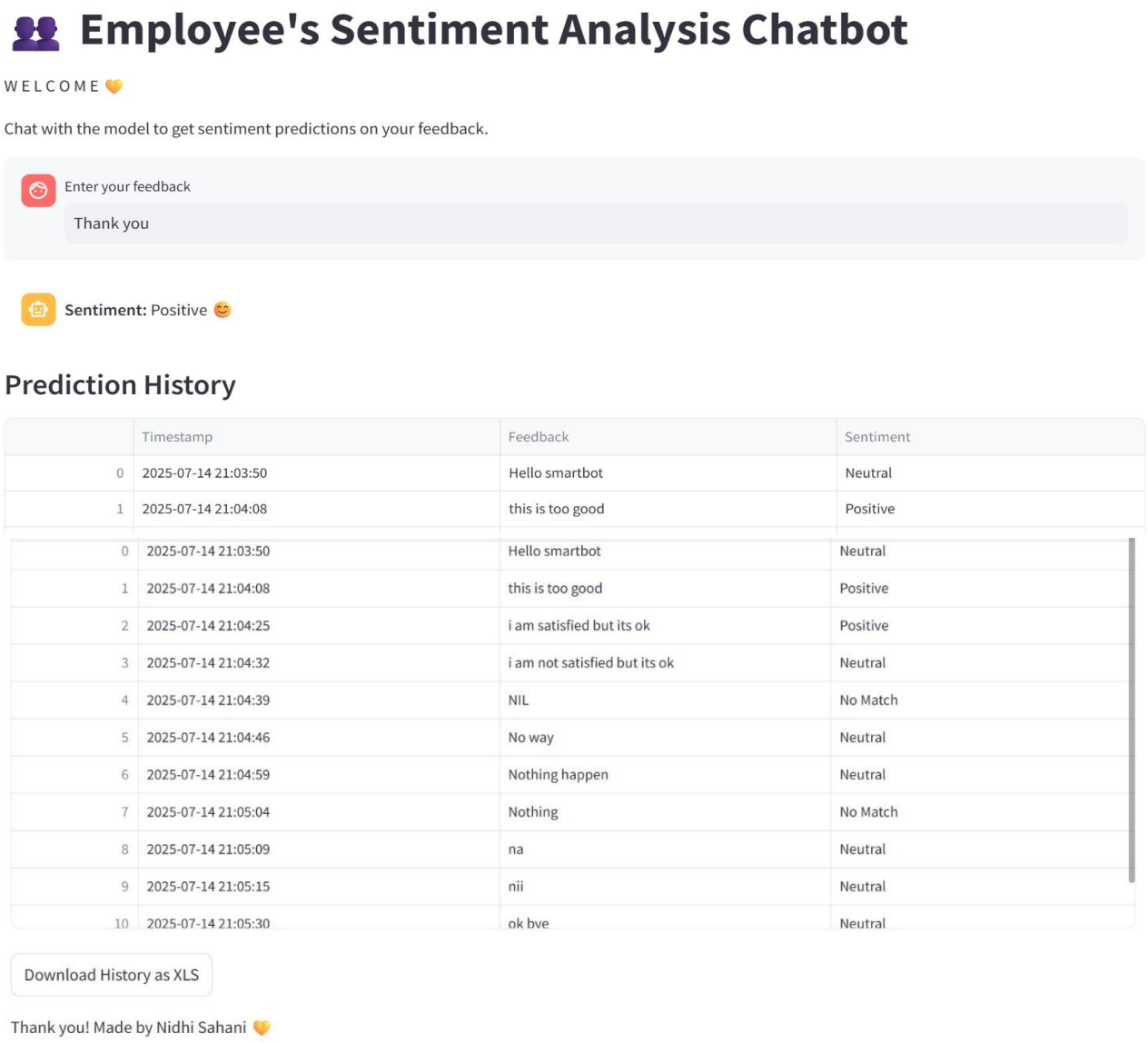
1. In my first attempt, the pre-trained model was only designed for 3 classes(positive, Neutral, Negative). It was not able to predict or recognize “No Match” category at all, and recall was 0%. The model treated “No Match” class as if it did not exist. But the overall accuracy was 97.98% that looked great, but it was misleading-because one class was never predicted.
2. To solve this, I need custom training on domain-specific dataset, I need to train the model by myself with all 4-class which means that handles all the imbalances in very high amount.
3. Even in my second attempt , model misclassifying between positive and negative , it made some mistakes, that may be risky in sentiment analysis and need to be reduce.
4. It give still low performance on “No Match” even after training, which means that “No Match” is very harder to detect and need more examples or logics.

Final Conclusion :

Metric	Attempt 1 (Pre-trained 3 classes)	Attempt 2 (Fine-tuned on 4 classes)
Overall Accuracy	**97.98%**	**89.38%**
Positive Recall	1.000 (100%)	0.972 (97.2%)
Negative Recall	1.000 (100%)	0.947 (94.7%)
Neutral Recall	1.000 (100%)	0.707 (70.7%)
No Match Recall	**0.000 (0%)**	**0.333 (33.3%)**
Pos → Neg Misclassifications	0	3
Neg → Pos Misclassifications	0	1
Evaluation Criteria Passed	5 / 6	4 / 6
Usability	Not usable for 4-class task	Usable with minor improvements

Streamlit Chatbot Application using Fine-Tuned RoBERTa Model :

I built an interactive chatbot web application using streamlit by my fine-tuned RoBERTa model trained to detect all 4-categories. This chatbot allow the users to input text feedback and immediately received an intelligent sentiment prediction. Also It has the functionality to track and export the prediction history in excel format , making it friendly. This model shows that how fine-tuned language language models can be deployed in real-time applications to extract insights from human feedback. This app is flexible, easy-to-use and a best solution for employee sentiment tracking or customer feedback system. It can overcomes the limitations of standard sentiment models and support for new category. This shows the full pipeline from model training to frontend application and deployment. And at last , It can be used by HR teams, analysts , customers and employees.



Deployment of Streamlit Chatbot Application

This application was deployed using streamlit in a local environment with the help of ngrok, which exposes local ports to the public internet. It allow anyone to interact with the chatbot in real-time without need to host one web server. It start the app locally and can be opened in any browser. It is the flexible deployment method with just few lines of code. It is perfect for testing and sharing projects and proves that advanced AI models can be integrated into real-time application without heavy infrastructure. Here are the steps to deploy app :

1. !pip install -r requirements.txt
2. !pip install pyngrok
3. !ngrok config add-authtoken 2**********
4. !streamlit run App.py