## Student Information

Name: 陳妘儒

Student ID: 113065541

GitHub ID: rheayunru

Kaggle name: googoose

Kaggle private scoreboard snapshot:


---

# Version Progression

## v1: Baseline Naive Bayes

**Reason for Starting Point:** Naive Bayes is a classic baseline model for text classification. Combined with TF-IDF, it enables the rapid establishment of a measurable starting point to evaluate the foundational performance of the model. This version was chosen to simplify the model construction process and expedite experimentation.

**Methodology:**

- Utilized TF-IDF for text vectorization, limiting features to 5,000.
- Employed Multinomial Naive Bayes, a suitable choice for text classification.

**Preprocessing:**

- Combined tweet data, sentiment labels, and data identifiers to construct training and testing datasets.

**Results and Evaluation:**

- **Public Score:** 0.37032
- **Private Score:** 0.35144

**Analysis:** The model exhibits clear deficiencies in capturing text semantics as it infers solely based on conditional probabilities of words, without considering contextual information. However, it is highly efficient and resource-light, making it suitable as a first attempt.

---

## v2: Random Forest

**Reasons for Selection:**

1. To experiment with a tree-based nonlinear classification approach and evaluate its potential to enhance performance.
2. To balance class imbalance, making the model more suitable for real-world data distributions.

**Methodology:**

- Applied TF-IDF feature engineering with a Random Forest Classifier.
- Used compute_class_weight to balance class imbalance.

**Preprocessing:**

- Cleaned tweet text by standardizing to lowercase and removing special characters.
- Split data into training and validation sets using stratified sampling.

**Results and Evaluation:**

- **Public Score:** 0.24155
- **Private Score:** 0.22482

**Analysis:** The performance of Random Forest was lower than that of Naive Bayes, indicating that the model still lacks semantic understanding for text data. Additionally, the TF-IDF features might be too sparse, further reducing classification effectiveness.

---

## v8: Twitter RoBERTa Base

**Motivation to Shift to Pretrained Models:** Transformer-based models (e.g., BERT and its variants) have outperformed traditional methods in natural language processing. Twitter RoBERTa is a pretrained model specifically fine-tuned for Twitter text, making it well-suited for social media data.

**Reasons for Selection:**

1. Pretrained models possess strong semantic understanding capabilities, compensating for the shortcomings of traditional methods.
2. Twitter RoBERTa's pretraining corpus aligns closely with the application scenario, maximizing performance potential.

**Methodology:**

- Leveraged Hugging Face’s Twitter RoBERTa Base model.
- Preprocessing included removing social media-specific noise (e.g., @user mentions, URLs).

**Results and Evaluation:**

- **Public Score:** 0.55604
- **Private Score:** 0.54167

**Analysis:** The model significantly improved classification performance, demonstrating its superior text understanding capabilities. However, its design is mainly tailored for binary classification or simpler sentiment analysis, with limited support for multi-class classification.

---

## v9: BERTweet

**Considerations for Further Optimization:** The results of Twitter RoBERTa highlighted the effectiveness of specialized models for social media text. However, its limitations in supporting multi-class classification led to the selection of BERTweet, a more Twitter-focused model, with additional fine-tuning.

**Reasons for Selection:**

1. BERTweet, optimized for Twitter text, might perform better in multi-class classification.
2. Custom loss functions and evaluation metrics (e.g., weighted F1 score) can more precisely reflect multi-class classification performance.

**Methodology:**

- Used Hugging Face's BERTweet Base model.
- Applied custom evaluation metrics (weighted accuracy and weighted F1 score).
- Fine-tuned hyperparameters, including a learning rate of 2e-5 and batch size of 16.

**Results and Evaluation:**

- **Public Score:** 0.58559
- **Private Score:** 0.57223

**Analysis:** The model performance improved again, indicating that targeted fine-tuning strategies for multi-class problems are effective. However, the higher computational resource and training time costs must be considered.

---

## v11: Twitter RoBERTa Large

**Reasons for Selection:**

1. Larger models offer higher performance ceilings, especially for applications requiring precise sentiment nuances.
2. Further adjusted the output layer to support multi-class classification.

**Methodology:**

- Adopted Hugging Face's Twitter RoBERTa Large model.
- Adjusted the output layer for multi-class classification.

**Results and Evaluation:**

- **Public Score:** 0.59478
- **Private Score:** 0.58245

**Analysis:** This model achieved the best performance of the entire experiment but required significantly more resources. Its extended training and inference times make it unsuitable for resource-constrained scenarios.

---

## v11.2: Optimized Twitter RoBERTa Large

**Optimization Goals:** To balance performance and resource consumption. To address the high resource demands of v11, efficiency was improved through mixed-precision training and other techniques.

**Reasons for Selection:**

1. Increasing batch size and using mixed precision effectively reduces training time.
2. Adding an early stopping callback improves training stability and prevents overfitting.

**Methodology:**

- Conducted secondary fine-tuning on the v11 model:
    1. Increased batch size to 32 for improved computational efficiency.
    2. Used mixed-precision training (fp16) to reduce resource usage.
    3. Added early stopping callbacks to avoid overfitting.
    4. Reserved 20% of training data as a validation set.

**Results and Evaluation:**

- **Public Score:** 0.58886
- **Private Score:** 0.57584

**Analysis:** The optimization strategies significantly enhanced computational efficiency and stability, though with a slight performance sacrifice (likely due to additional hyperparameter adjustments). Overall, this is a resource-friendly improvement.

---

# Version Comparison Table

| Version | Method/Model | Public Score | Private Score | Advantages | Disadvantages |
|---|---|---|---|---|---|
| **v1** | Naive Bayes + TF-IDF | 0.37032 | 0.35144 | Simple and fast; low resource demands | Lacks contextual semantic understanding |
| **v2** | Random Forest + TF-IDF | 0.24155 | 0.22482 | Balances class imbalance; interpretable model | Underperforms; limited semantic understanding |
| **v8** | Twitter RoBERTa Base | 0.55604 | 0.54167 | Adapts well to social media text; large improvement | Limited support for multi-class classification |
| **v9** | BERTweet | 0.58559 | 0.57223 | Fine-tuned for multi-class classification | Increased resource requirements |
| **v11** | Twitter RoBERTa Large | 0.59478 | 0.58245 | Best performance; captures subtle sentiment differences | High training time and resource demands |
| **v11.2** | Optimized Twitter RoBERTa Large | 0.58886 | 0.57584 | Enhanced stability; improved computational efficiency | Slightly reduced performance |

## Summary and Insights

The progression from Naive Bayes to Twitter RoBERTa Large highlights the ongoing effort to balance model performance and resource efficiency. Each step involved evaluating the current approach's strengths and weaknesses while aligning with application needs. Key takeaways include:

- **Baseline Models (v1, v2):** Simple and quick to deploy, serving as foundational benchmarks, but limited in semantic understanding.
- **Pretrained Models (v8, v9):** Utilize Transformers to significantly enhance classification performance, particularly with task-specific fine-tuning.
- **Optimized Models (v11, v11.2):** Deliver peak performance while requiring strategic resource management for practical deployment.

Each iteration addressed specific requirements and application contexts, ensuring choices were rational and meaningful in practice.

---

## Directions for Improvement

- Combine the best performance of v11 with the efficient training strategies of v11.2.
- Explore model ensembles that integrate the rapid inference of lightweight models with the accuracy of larger models.
- Optimize the inference pipeline to reduce inference time, aligning better with practical application requirements.

```python
# v11
import pandas as pd
import torch
import json
from transformers import AutoTokenizer, RobertaForSequenceClassification, Trainer, TrainingArguments
from datasets import Dataset


# 讀取 JSON 格式的推文數據，轉換為 DataFrame 格式
with open('data/tweets_DM.json', "r") as file:
    tweets_data = [json.loads(line) for line in file]
tweets_df = pd.DataFrame([tweet["_source"]["tweet"] for tweet in tweets_data])

# 讀取情感標籤和數據集劃分文件
emotion_df = pd.read_csv("data/emotion.csv")
data_identification_df = pd.read_csv("data/data_identification.csv")

# 分離訓練集和測試集
train_ids = data_identification_df[data_identification_df["identification"] == "train"]
test_ids = data_identification_df[data_identification_df["identification"] == "test"]

# 合併數據集，過濾空值
train_df = train_ids.merge(tweets_df, on="tweet_id").merge(emotion_df, on="tweet_id", how="left")
test_df = test_ids.merge(tweets_df, on="tweet_id", how="left")
train_df = train_df.dropna(subset=["emotion"])

# 定義情感標籤並將其轉為數字
emotion_labels = ["anger", "anticipation", "disgust", "fear", "sadness", "surprise", "trust", "joy"]
train_df["label"] = train_df["emotion"].apply(lambda x: emotion_labels.index(x))

# 創建 Hugging Face Dataset，僅保留文本和標籤
train_dataset = Dataset.from_pandas(train_df[["text", "label"]])
test_dataset = Dataset.from_pandas(test_df[["text"]])
```

```python
# 加載預訓練模型和分詞器
MODEL_NAME = "cardiffnlp/twitter-roberta-large-emotion-latest"
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
model = RobertaForSequenceClassification.from_pretrained(
    MODEL_NAME,
    num_labels=len(emotion_labels),
    ignore_mismatched_sizes=True  # 忽略維度不匹配的情況
)

# 顯式設置問題類型為單標籤分類
model.config.problem_type = "single_label_classification"
print(f"Number of labels: {model.config.num_labels}")


# 定義分詞函數，確保長度和填充
def preprocess_function(examples):
    return tokenizer(examples["text"], truncation=True, padding="max_length", max_length=128)

# 應用分詞至數據集
train_dataset = train_dataset.map(preprocess_function, batched=True)
test_dataset = test_dataset.map(preprocess_function, batched=True)

# 確保標籤為整數類型
train_dataset = train_dataset.map(lambda x: {"labels": int(x["label"])})

# 設置數據格式供模型使用
train_dataset = train_dataset.remove_columns(["text"])
train_dataset.set_format(type="torch", columns=["input_ids", "attention_mask", "labels"])

test_dataset = test_dataset.remove_columns(["text"])  # 僅保留必要的欄位
print(test_dataset.column_names)
test_dataset.set_format("torch")
```

```python
# 訓練參數設置
training_args = TrainingArguments(
    output_dir="./results",   # 輸出目錄
    eval_strategy="no",  # 不進行評估
    learning_rate=2e-5,  # 設置學習率
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,  # 訓練週期數
    weight_decay=0.01,  # 權重衰減
    save_total_limit=1,  # 最多保存1個模型
    logging_dir="./logs",
    metric_for_best_model="accuracy",  # 評估的最佳指標
)

# 創建訓練器
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    tokenizer=tokenizer,
)

# 訓練模型
trainer.train()


# 預測測試集
predictions = trainer.predict(test_dataset)
predicted_labels = predictions.predictions.argmax(axis=1)

# 保存測試結果至 CSV 文件
test_df["predicted_emotion"] = [emotion_labels[label] for label in predicted_labels]
test_df[["tweet_id", "predicted_emotion"]].to_csv("predicted_emotions_v11.csv", index=False)
print("v11儲存✧*。٩(ˊᗜˋ*)و✧*。")
```