CA6000 Advanced Data Analytics Assignment

Priority Classification of Customer Support Tickets Using Neural Networks

A Comparative Analysis using Logistic Regression, TextCNN, and BERT

Prepared by:

Su Zhiyuan Tuo Xueting

1. Introduction

1.1 Background & Motivation

In today's companies, the operations department receives hundreds or even thousands of support tickets every day. However, these support tickets have different priorities, and employees need to classify and sort them before sending them to the appropriate departments for processing, which is extremely inefficient. A faster processing method could significantly reduce the time spent in this process.

Whether training a natural language processing model can efficiently handle this problem is the key to our exploration. Priority discrimination is often unclear; even for humans, judging medium and low priority is highly subjective.

1.2 What We Set Out to Do

Our goal is to compare and select the best among different neural network models to solve this problem：

- For the first model, we choose a baseline model to use as a reference. (TF-IDF + logistic regression).
- Attempt to use the neural network approach of TextCNN.
- Use a pre-trained Transformer model (DistilBERT) to test the effectiveness of transfer learning.

We wanted to understand the trade-offs between these approaches - not just accuracy, but also training time and practicality.

---

2. Problem Definition

2.1 Task Description

For this project, which is a typical supervised learning task, the model we selected was trained to predict priorities by analyzing the work order text: classifying them into three priorities: low, medium, and high.

For the evaluation method, we chose to use both accuracy and macro F1 score. In order to deal with class imbalance, using only accuracy may be misleading; in this case, macro F1 score can better reflect the model's ability to handle each class.

2.2 Constraints and Assumptions

Several points regarding the project setup are worth mentioning:

- The authors have indicated that the dataset is synthetic and may not contain real work order information;
- This project only considers English work orders;
- Considering the limited GPU memory constraints, DistilBERT is used instead of the full BERT;
- The dataset has an uneven distribution of categories (we'll discuss this later), so we need to discuss the metrics for each category.

---

3. Dataset Description

3.1 Where the Data Comes From

The dataset we used, called "customer-support-tickets," is taken from Hugging Face (by Tobi-Bueck). It contains customer support tickets from various industries, including software, healthcare, financial services, and infrastructure. Each ticket has a subject, body, and priority tags.

We filtered the dataset to include only tickets in English and combined the subject and body columns into a single text field.

3.2 Dataset Size and Split

After data cleaning, there are 28,261 samples in total. And we split them into 70/15/15:

Split	Samples
Training	19,782
Validation	4,239
Test	4,240

We used stratified splitting to maintain similar class distributions across all splits.

3.3 Data Format

The processed data is stored as CSV files with these columns: firstly, `text` is the combination of subject and boy; then `priority` is the label for the text; and also `cleaned_text` is the text we preprocessed for modeling

3.4 How We Cleaned the Data

We created a simple cleaning function that cleaned the dataset as follow:

1. Converts all the text content to lowercase.
2. Removes URLs and email addresses which is not useful for classification.
3. Normalizes whitespace.

```
def basic_clean(text):
    text = text.lower()
    text = re.sub(r'http\S+|www\S+', '', text)
    text = re.sub(r'\S+@\S+', '', text)
    text = ' '.join(text.split())
    return text.strip()
```

Nothing fancy, but it works. We applied the same cleaning to all data used across all models to keep things fair.

Each model needs a different input format though:

Model	How text is encoded

Model	How text is encoded
Logistic Regression	TF-IDF vectors (max 10k features, unigrams + bigrams)
TextCNN	Integer sequences using a vocabulary we built from training data
DistilBERT	WordPiece tokens using HuggingFace's tokenizer

3.5 Looking at the Data

Here's the class distribution in the training set:

Priority	Count	Percentage
High	7,698	38.9%
Medium	8,041	40.6%
Low	4,043	20.4%

We can see that there's some imbalance: "low" priority has about almost half the samples of the other classes. This does also make sense for support tickets since people tend to think their issues are more urgent than others.

Text length stats:

- Average: ~411 characters / ~60 words
- Range: 16 to 1,715 characters
- Most tickets are between 200-600 characters

---

4. Why We Chose These Models

We adopted a "start with the simple and then gradually increase the complexity" approach.

4.1 Baseline: TF-IDF + Logistic Regression

We chose this method because it is fast and provides a reasonable benchmark. TF-IDF converts text into numerical features based on word frequency, and then logistic regression finds a linear decision boundary.

Its main limitation is that it treats text as a "bag of words"—word order is irrelevant. "Bad" and "good or bad" are treated the same in this model.

Our settings for TF-IDF are up to 10,000 features, including bigrams using logistic regression, class weighted

4.2 TextCNN

This is our first attempt to use a neural network. TextCNN uses convolutional filters to capture local patterns (such as n-grams) in text. The basic idea is that certain word combinations may represent priorities.

Architecture:

- Embedding layer (128 dimensions)
- Three parallel conv layers with filter sizes 3, 4, 5 (100 filters each)
- Max pooling, then concatenation
- Dropout (0.5) and final classification layer

We trained the program for 5 epochs using the Adam optimizer. We found that TextCNN benefits greatly from pre-trained word embeddings, which we didn't use here. This likely impacted its performance.

4.3 DistilBERT

Finally, we attempted to fine-tune DistilBERT. It's a streamlined version of BERT, pre-trained on massive amounts of text, and therefore already possesses some language understanding capabilities.

We simply added a classification head and fine-tuned the entire model for 3 epochs. The learning rate was much smaller (2e-5) because we didn't want to break the pre-trained knowledge.

On our GPU, training took approximately 30 minutes, while logistic regression took less than 1 minute.

4.4 Other Things We Could Have Tried

Looking back, there are also some alternatives which we didn't explore for this project:

1. Using SVM or Naive Bayes model as optional baselines
2. Using a pre-trained word embeddings model like Word2Vec for TextCNN
3. Larger models like BERT-base or RoBERTa
4. Handling class imbalance more explicitly (oversampling, focal loss)

All these can be futher explored for future work.

---

5. Methodology

5.1 Overall Pipeline

Our workflow is fairly standard:

1. Load raw data, filter the data for only English, and clean the text.
2. Split the dataset into training/validation/test sets (70/15/15, stratified sampling).
3. For each model: prepare input, train, and evaluate.
4. Compare results on the reserved test set.

All three models used the exact same data segmentation and cleaned text, so the comparison is fair.

5.2 Training Details

Logistic Regression:

- Simple sklearn fit/predict operations
- Handle imbalanced data using class_weight='balanced'
- Training takes less than a minute

TextCNN:

- Build a vocabulary from the training data (all words that appear at least once)
- Pad/truncate sequences to a length of 256
- Batch size: 64, Training epochs: 5
- Use weighted cross-entropy loss function

DistilBERT:

- Maximum sequence length 128 (work orders are usually short enough)
- Batch size 32, 3 epochs
- AdamW optimizer, learning rate 2e-5

5.3 What We Measured

For evaluation, we calculated the following metrics:

- Overall accuracy
- Precision, recall, and F1 score for each class
- Macro-average F1 score (treating all classes equally)
- Confusion matrix (used to analyze the location of errors)

We primarily focus on the macro-average F1 score because predicting only the majority of classes might artificially inflate accuracy.

---

6. Implementation Notes

6.1 Code Organization

We created separate notebooks for each model to build the project:

- `01_eda.ipynb` - Data Exploration Analysis
- `02_baseline_ml.ipynb` - Logistic Regression Model
- `03_cnn_model.ipynb` - TextCNN Model
- `04_bert_model.ipynb` - DistilBERT Model

Shared code (data loading, preprocessing, evaluation) is located in the `src/` folder.

6.2 Some Issues We Ran Into

CUDA memory errors with BERT: Firstly we used batch size of 64, but our GPU kept running out of memory. So we had to reduce the batch size to 32.

TextCNN underfitting: Our initial TextCNN was not learning a lot. It turned out we had the embedding dimension too small which was 50, and we changed it to 128 and needed more filters.

Class imbalance: The "low" class kept having worse predictions than others. So we added class weights to the loss function helped on some aspects.

6.3 Reproducibility

We set random seeds as far as possible and saved all trained models. Assuming the data is in the correct location, these notebooks should run successfully.

7. Results

7.1 Test Set Performance

The following table shows the prediction results of these three models on the test set:

Model	Accuracy	Macro F1	Training Time
Logistic Regression	64.4%	63.6%	<1 min
TextCNN	60.4%	59.3%	~10 min
DistilBERT	**73.2%**	**72.4%**	~30 min

Therefore, DistilBERT clearly wins. However, to our surprise, TextCNN's actual evaluation results on this dataset are worse than those of logistic regression—we will try to explore the reasons below.

7.2 Per-Class Breakdown

Logistic Regression:

```
              precision    recall  f1-score   support
        high       0.69      0.67      0.68      1604
         low       0.54      0.65      0.59       876
      medium       0.67      0.62      0.64      1760
   macro avg       0.63      0.65      0.64      4240
```

TextCNN:

```
              precision    recall  f1-score   support
        high       0.61      0.74      0.67      1604
         low       0.50      0.62      0.55       876
      medium       0.70      0.47      0.56      1760
   macro avg       0.60      0.61      0.59      4240
```

DistilBERT:

```
              precision    recall  f1-score   support
        high       0.75      0.73      0.74      1604
         low       0.65      0.72      0.68       876
      medium       0.74      0.73      0.73      1760
   macro avg       0.71      0.73      0.72      4240
```

DistilBERT is clearly more balanced—it doesn't sacrifice other aspects to excel in one class.

7.3 Where Models Go Wrong

The confusion matrix reveals that the main problem with all models lies in the correctness of prioritizing "medium" or "low." This is quite reasonable, as even for humans, there isn't a clear boundary between the two.

"High" priority is easier to identify, likely because customers use more urgent language (e.g., "critical," "fault," "urgent," "as soon as possible") in most urgent work orders.

7.4 Why TextCNN Underperformed

We were surprised that the TextCNN model performed worse than the first baseline model. This may be due to the following reasons:

1. **No pre-trained embeddings:** We trained the word embeddings from scratch, but this method did not perform well with about 20,000 training samples.
2. **Dataset might be too clean:** Synthetic data often lacks the noise and diversity found in real text, which neural networks need to generalize.
3. **Hyperparameter tuning:** We haven't done extensive tuning of the model; there are likely better configuration options available.

If we had more time, we would try initializing it using GloVe word embeddings and performing a more comprehensive and diverse hyperparameter search.

---

8. Discussion

8.1 What We Learned

The most important conclusion is that pre-training remains crucial for natural language processing models. While DistilBERT is used almost directly, its performance evaluation significantly outperforms the other two methods. It has already been exposed to a large amount of text during pre-training, so only fine-tuning is needed on these 20,000 samples.

Meanwhile, the simple TF-IDF + logistic regression baseline model performed surprisingly well. It is also a good choice if a fast and easily interpretable model is needed for this project.

8.2 Practical Considerations

If we were deploying this for a real company:

- **For a quick prototype:** Logistic regression model. It is fast, highly interpretable, and its accuracy is not bad.
- **For best accuracy:** For DistilBERT， if you don't do repetitive training often, 30 minutes of training time is sufficient.
- **For edge deployment:** Further simplification of the BERT model or the use of a lighter model may be necessary in the future.

8.3 Limitations

The following factors may affect the generalizability of the results:

- The synthetic data is not entirely identical to real ticketing data.

- We only tried one random seed for data partitioning.
- The hyperparameter tuning was relatively low.
- No ensemble methods were attempted.

8.4 What We'd Do Differently

If we have more time, we can:

- Try training TextCNN using pre-trained text embeddings
- Use k-fold cross-validation instead of single-segmentation
- Try data augmentation (paraphrasing, back-translation)
- Perhaps try an ensemble model of logistic regression + BERT

---

9. Conclusion

We built a ticketing classification system and compared three methods. The main findings are as follows:

Performance ranking:

1. DistilBERT: 73.2% accuracy, 72.4% macro-F1
2. Logistic Regression: 64.4% accuracy, 63.6% macro-F1
3. TextCNN: 60.4% accuracy, 59.3% macro-F1

Key insights:

- Pre-trained models (BERT) have a big advantage for NLP tasks, even with moderate training data
- Simple baselines can be surprisingly competitive and shouldn't be skipped
- The medium/low priority boundary is inherently difficult - might need better labeling guidelines or additional features

For this specific task, if accuracy is important, we recommend using DistilBERT; if speed and interpretability are required, we recommend using logistic regression.

---

10. References

Papers:

Devlin et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." NAACL 2019.

Kim, Y. (2014). "Convolutional Neural Networks for Sentence Classification." EMNLP 2014.

Sanh et al. (2019). "DistilBERT, a distilled version of BERT." arXiv:1910.01108.

Libraries:

- PyTorch: https://pytorch.org/
- HuggingFace Transformers: https://huggingface.co/transformers/
- scikit-learn: https://scikit-learn.org/

Dataset:

Tobi-Bueck/customer-support-tickets on Hugging Face: https://huggingface.co/datasets/Tobi-Bueck/customer-support-tickets

Disclosure of AI-assisted work: We used Claude as a programming assistant during the project. Specifically, it was used to help with code generation for model classes and training loops, debugging runtime errors, and formatting this report. The dataset selection, preprocessing pipeline design, model training, and evaluation were performed by the authors, and all reported results were generated from our own executed code.