Intent Classification: Machine Learning Approach and Results Analysis
-Abhay Sastha S

Goal 1: Framing the Problem as a Machine Learning Task

The intent classification problem is framed as a supervised multi-class classification task, aiming to predict one of 21 discrete intent labels from user sentences. This aligns with machine learning's ability to identify patterns in text and map them to predefined categories. The dataset, originally 328 labeled entries (augmented to 1090), supports supervised learning, making classification the most suitable approach over regression (continuous outputs) or clustering (unlabeled data).

Task Definition: Take a user sentence, analyze its intent, and output exactly one label from a fixed set. This requires understanding semantic relationships (e.g., "discount" and "reduced price" mapping to "OFFERS").

Why Classification?: The categorical, discrete nature of intents and availability of labeled data make it a natural fit. Each entry has one label, with no missing values, reinforcing this choice.

Data and Features: The small dataset necessitated augmentation (paraphrasing, back-translation). Text vectorization progressed from Bag of Words (BoW) and TF-IDF (baselines) to Word2Vec (neural networks) for semantic depth, with preprocessing like lemmatization and stopword removal.

Evaluation: Accuracy was tracked, but weighted averages (precision, recall, F1)

Goal 2: Code Submission

- Approach to final model: https://www.kaggle.com/code/abhaysasthas/tifin
- Final model: tifin_model_ft.ipynb
- Datasets:
    - https://www.kaggle.com/datasets/abhaysasthas/sofmattress (original dataset)
    - https://www.kaggle.com/datasets/abhaysasthas/augmented (augmented)
    - https://www.kaggle.com/datasets/abhaysasthas/validation-gretel (synthetic)
    - Excel Sheet for data augmentatio Data Aug Tifin
- Instructions to run and reproduce code added in this document itself

Goal 3: Explaining Results and Suggesting Improvements

The final model, a fine-tuned Pythia-70m, achieved 92–93% accuracy on the augmented dataset (1090 entries). Executed on a GPU T4 runtime (see Kaggle notebook), it used a pre-trained tokenizer and was fine-tuned with a learning rate of 1.5e-5, batch sizes of 5 (training) and 3 (evaluation), and 5 epochs, yielding a sharp training loss drop and steady validation loss reduction.

Why It Worked:

- Pre-trained Embeddings: Pythia-70m's foundation mitigated the small dataset's limitations.
- Data Augmentation: Boosted diversity, improving generalization from 328 to 1090 entries.
- Fine-Tuning: Adapted the model to task-specific patterns, outperforming alternatives.

Performance Insights:

- Training Loss: Near-zero by Epoch 5, showing effective learning.
- Validation Loss: Generally decreased

- Comparisons: Outperformed Naive Bayes (50% ±8%), SVM (71% original, 92% augmented but keyword-dependent), simple neural nets (95% with overfitting), LSTM (12–13%), and BERT (failed due to data scarcity).

- Analysis: SVM struggled with contextual understanding (e.g., confusing "ORDER_STATUS" with "DELAY_IN_DELIVERY"), while Pythia-70m balanced accuracy and consistency.

Improvements:

- Class Imbalance: Apply weighted loss or oversampling for rare intents.
- Tuning: Use grid search for hyperparameters (e.g., learning rate, batch size) to gain 1–2% accuracy.
- Model Exploration: Test Pythia-160m or quantized versions (4/8-bit) for efficiency vs. accuracy trade-offs.
- Deployment: Enhance Streamlit with confidence scores and a misclassification flagging system for continuous learning.

This approach evolved from baselines (Naive Bayes, SVM) to neural networks, culminating in Pythia-70m, with clear paths for further robustness and usability.

This one page summary was helped by and created with ChatGPT based on the following pages.

Approach to reach final model:
https://www.kaggle.com/code/abhaysasthas/tifin
Kaggle notebook(rough comments)

Execution-

- Can be run directly on a runtime (GPU T4)
- Install commands have been put in places needed
- Make sure datasets have been connected
    - https://www.kaggle.com/datasets/abhaysasthas/sofmattress (original dataset)
    - https://www.kaggle.com/datasets/abhaysasthas/augmented (augmented)
    - https://www.kaggle.com/datasets/abhaysasthas/validation-gretel (synthetic)
- Might take a few seconds to download word2vec

Analysis-

- Understanding the dataset
    - 328 entries
    - 21 labels
    - Each entry had exactly one label
    - No none values
    - Potential of generalization of model - low accuracy models

- Framing the problem:
    - Task:
        - Find the intent of a user to respond/act accordingly
        - This requires us to take input from user
        - Understand the requirement
        - Find the most suitable act/label
        - Exactly one label
        - Give that label as output of our system
    - The requirement to select one output from a predefined set of labels based on the input analysis is a natural fit for classification over tasks like regression or clustering.
    - This requirement to analyze patters in sentences and labels is what makes it a task fit for machine leaning. Like finding relation ship between words like discount, reduced price might map to the label offers.
    - A dataset labels are discrete and categorical which further classifies this task as classification
    - Exploring alternative path:
        - <u>Clustering</u>: a technique that could have fairly mimicked the solution
            - Problem 1: 21 labels, 21 clusters

        - High complexity
        - Mapping and finding most relevant intent would be challenging
    - Problem 2: dataset
        - The limited dataset will restrict/mislable the mapping of unknown user inputs, such as the use of synonyms of exact sentences from dataset

Exploration-

- Text vectorization:

  To convert the text to vectors that could be understood by the models
    - One hot encoding
        - Wide spread unique words in dataset
        - Dimensionality would be too high
        - Very restrictive to new words from user
        - Option discarded
    - Bag of words (BoW)
    - TF-IDF
    - BoW was a simple, interpretable baseline, but ended up shifting to TF-IDF to be able to better accommodate and weight the term importance, potentially improving the intent differentiation
- Naive-Bayes:
    - Explored with both BOW and TF-IDF
    - With an accuracy range of 50% +-8
    - Ideal for setting up the baseline, simple to setup, and efficient
    - Highlights:
        - Zero division error: showed the evalution/test splits did not have adequate data
        - Referred to avg_weighted accuracy from here on to accommodate that
- SVM:
    - With tf-idf was able to achieve 71% acc on original given dataset
    - Sparse and high dimensional data results in svm to find meaning links

- Data Augmentation:
    - Done on excel, file attached
    - Conversion of the original dataset to a different language and converted back
    - Add it
    - Paraphrasing on new combined data
    - Add it

    - Duplicated fixing
    - Data formatting
    - New dataset with 1090 entries and labels ready

- SVM on new dataset
    - Showed a significant improvement on the dataset
    - 92% weighted avg accuracy
    - On further analysis we see that the mapping of the labels to classes in certain instances such what_size_to_order and size_customization or cancel_order, order_status tend to overlap and be confused as the sentences use similar words
    - Hence showing for a lack of contextual understanding
    - Very key word dependent
    - Beyond TF-idf term representation
        - Hence double checked using unseen data
        - And clearly the accuracy and recall for labels feel as this dataset used much more synonyms of the original dataset
        - Failing the keyword dependent svm

- Data preprocessing:
    - Punctuation removal
    - Stopwords removal
    - lemmatization
    - Lowercase
    - Tokenization - word2vec

- Simple Neural Net:
    - A feedforward neural net
    - Accuraciy of upto 95% with inconsistent and undesirable patterns in loss and validation accuracy
    - Showing overfitting signs
    - Parameter tuning usch as epochs, batch size had barely any effect
    - With L2 - regualarization:
        - Penalized weights
        - Accuracy at 92% but more inconsistencies in loss and accuracy per epoch
        - Models stats to over fit, early stopping did not help either
- LSTM:
    - 12-13% acc
    - Constrained by time, computational resources, insufficient dataset
- BERT:
    - Finetuning the model to add contextual understanding

    - failed due to the lack of adequate diversity and quantity of dataset

Final Model:

Pythia 70m fine tuned tifin_model_ft.ipynb

Execution-

- Untitled spreadsheet - sheet4.csv (if running on colab be placed in ./content)
- Run the notebook
    - While running notebook it will ask for wandb.ai api key https://wandb.ai/
    - (4748e2f18b6ce88446e0f24b0411d8bd29604f07)
    - temporary token for use
- After the main training script is done
    - There are two inference wrappers
        - Terminal
            - Run the snippet
            - Give user input
            - Predicts the best label
            - Type “exit” to end code
        - Streamlit
            - Run the scripts
            - It will give a IPv4 formatted pin (thats the password for next step)
            - A link will show up
            - Open up the link
            - Enter the pin as it is (34.127.89.21)
            - Give it a few secs, stream lit shall run
            - Start testing with user inputs

Analysis-

- Fine tuned on the augmented dataset, sheet 4.
- Used pre-trained tokenizer of pythia-70m (GPTneoX model)
- Training config was finalized on:
    - Learning rate: 1.5e-5
    - Batch size:
        - 5 for training,
        - 3 for evaluation
    - Epochs: 5
    - Weight decay: 0.02
    - Evaluation strategy: best model after each epoch, evaluated on lowest validation loss

- Could see a sharp drop in training loss at 5th epoch, along with steady drop in validation loss
    - Indicating the model trained on the data set effectively, but
        - multiple runs indicated certain cases of spike in validation loss at epoch 4
        - Indicating a likely near fitting of data
        - Potentially solvable through a bigger and diverse dataset / sampling techniques

- Hyperparameters such as the ones mentioned above were toyed around with leading a finer improvements in accuracy and more consistent and agreeable loss patterns
- Pythia-70m shows an accuracy range of 92-93% +-0.6
- Perks: https://huggingface.co/abhaysastha/intent-model-70m-ft/tree/main
    - Relatively small size
    - Ability to be used as call in adapter from HF
    - Low inference time

Scope-

- Comparison with performance of DistilBERT / Pythia 160m
    - Might lead to better accuracy and lower inference times
- Use a quantized version in 4/8bit to compare for accuracy loss wrt to model size
- Class imbalance can be addressed more carefully using a more curated and better sampled dataset
- Contrast learning could also be employed to better create separation between overlapping classes. Would improve recall value for such classes.
- Grid search based Hyperparameter optimization further optimize the model
- Add a misclassification button on streamlit to flag and assign appropriate flag, this can be paired with a batch retraining cycle for the model to evolve along with the deployment