41021112L 王玟婷

Lab 2 Report

This is my report of Lab2. I tried a lot of different method. However, I didn't get satisfied result in the end. So this report is more like a reflection to inspect where I can improve my models. In total I tried six methods, but I only submited five of them.

There are three datasets we use. They are data_identification, which contains all the tweet ids, emotion.csv, which are the gold labels of our training datas, and the last one – tweets_DM.json, which is the main content of the tweets in json format. In the data processing method, there are some findings. First, there are 1455563 ids in the training data, and 411972 ids in the test data. It is a quite large number. Then, I combined ids, emotions, and text together into a dataframe via using pandas. There is an additional things I did is that I extract the hashtags in the text because I thought it might be useful speaking of classifying. (If there is no hashtag, just lable "no hashtag".) Hence, basically, the dataframe would be like:

	tweet_id	text	\
0	0x376b20	People who post "add me on " must be dehydrate...	
1	0x2d5350	As we see, Trump is dangerous to around the w...	
2	0x1cd5b0	Now ISSA is stalking Tasha 😂😂😂	
3	0x1d755c	Thx for the BEST TIME tonight. What stories! H...	
4	0x2c91a8	Still waiting on those supplies Liscus.	
...	...	...	
1455558	0x321566	I'm SO HAPPY!!! the name of this show! ! 👏...	
1455559	0x38959e	In every circumtance I'd like to be thankful t...	
1455560	0x2cbca6	there's currently two girls walking around the...	
1455561	0x24faed	Ah, corporate life, where you can date using ...	
1455562	0x34be8c	Blessed to be living	

	emotion	hashtags
0	anticipation	[#Snapchat]
1	sadness	[#freepress, #TrumpLegacy, #CNN]
2	fear	[no hashtag]
3	joy	[#authentic, #LaughOutLoud]
4	anticipation	[no hashtag]
...	...	...
1455558	joy	[#NoWonder, #Happy]
1455559	joy	[no hashtag]
1455560	joy	[#blessyou]

In the following part, I would like to list out my methods one by one.

1. Decision Tree + BOW500

Feature Engineering:

1. **Text Tokenization and Preprocessing:**
   - The text data is tokenized into individual words and emojis.
   - Emojis are treated as separate features to capture emotional cues effectively.
2. **Bag of Words (BOW) Representation:**

- A vocabulary of **500 features** is constructed based on term frequency.
- Only the **most frequent 500 terms** in the dataset are retained to avoid sparsity and improve computational efficiency.
- Emojis are included in the vocabulary alongside words to account for non-verbal emotional signals.

3. **Max Features Parameter:**
    - The max_features=500 parameter ensures that the feature matrix remains of manageable size, improving model interpretability and reducing training time.

Model Training:

- **Algorithm:** A **Decision Tree Classifier** is chosen for its simplicity and interpretability.
- **Training Process:**
    - The BOW feature matrix is used as input to the Decision Tree algorithm.
    - The model learns hierarchical decision rules to classify text samples into different emotion categories.
    - The model took 430**8 seconds** to complete training, suggesting a moderate computational cost given the feature size.

Advantages:

- Simple and interpretable model suitable for identifying key decision splits.
- Incorporation of emojis enriches the emotional context captured by the BOW representation.

Limitations:

- Decision Trees are prone to overfitting, especially with high-dimensional data.
- The BOW approach may lose semantic relationships between words, limiting its ability to capture nuanced emotions

Therefore, I tried another feature engineering way: TF-IDF in order to reduce the importance of frequent words.

Method 2: TF-IDF + Decision Tree

Feature Engineering:

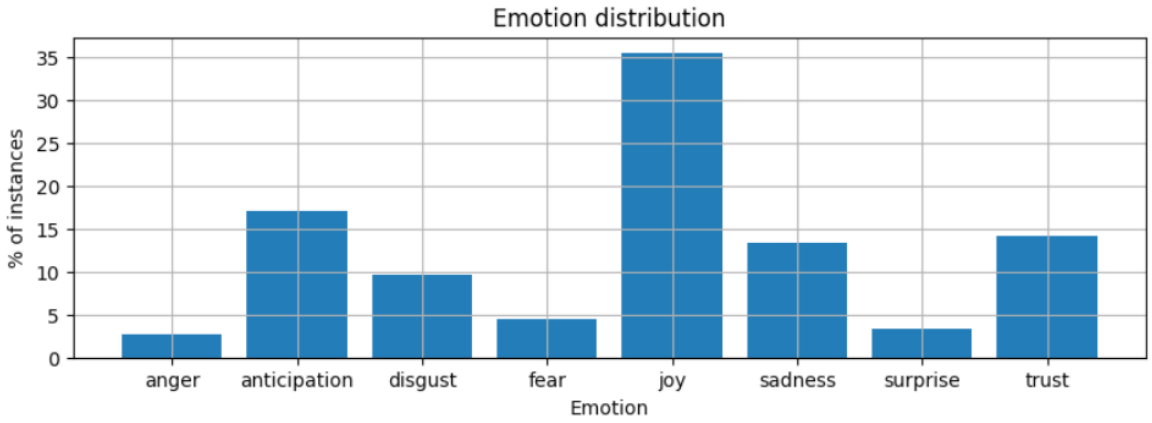
1. **TF-IDF Representation:**
    - **TF-IDF Transformation:**
        - Converts raw text into a matrix of numerical values representing the importance of terms relative to their frequency in the document and across the corpus.
        - Words and emojis are treated as tokens to ensure emotional nuances are captured.
    - **Feature Size:**

    - The matrix is constrained to the 1000 **most relevant features**
- **Normalization:**
    - The resulting TF-IDF values are normalized, ensuring all features have comparable scales.

Model Training: A **Decision Tree Classifier** is chosen for its simplicity and interpretability.

- The model took **2698 seconds** to complete training, suggesting a moderate computational cost given the feature size.

However, there is something I missed: Random Forest



As we can see, the distribution of each emotion varies. When it comes to the unbalanced data, using random forest might be better. I should pay more observation on data distribution.

Method 3: Word2Vec + DNN
Feature Engineering:

1. **Word Embeddings:**

    - **Pre-trained Sentence Embeddings:**
        - The SentenceTransformer library is used to obtain dense, high-dimensional sentence embeddings.
        - Sentence embeddings are computed to capture the semantic meaning of each text sample.
        - These embeddings incorporate contextual nuances, making them more powerful than bag-of-words or TF-IDF.

    - **Embedding Representation:**
        - Each sentence is converted into a 384 vector representation

2. **Label Encoding:**

    - **Label Encoding for Classification:**
        - Labels (emotions) are encoded into numerical values using a Labelencoder.

- This transforms the categorical labels into a format compatible with the DNN.

Model Architecture:

1. **Deep Neural Network (DNN):**
    - **Input Layer:** Accepts the pre-computed sentence embeddings as input.
    - **Hidden Layers:**
        - Two fully connected (dense) hidden layers are added to the network.
        - Each hidden layer uses **ReLU activation**, ensuring non-linear transformations and robust feature learning.
    - **Output Layer:**
        - A dense layer with the number of neurons equal to the emotion classes.
        - **Softmax Activation** is applied to output probabilities for each class.

2. **Compilation:**
    - **Loss Function:** Categorical Crossentropy (for multi-class classification).
    - **Optimizer:** Adam optimizer for efficient gradient descent.
    - **Evaluation Metric:** Accuracy, loss

Training and Results:

1. **Training Process:**
    - The model is trained on the sentence embeddings and encoded labels for 25 epochs.

2. **Running Time:**
    - The model took **2499 seconds** to complete training

Performance Metrics:

1. **Accuracy:**
    - Accuracy: 0.5124, loss= 1.3285

Advantages:

1. Word2Vec embeddings capture semantic relationships between words, enriching the contextual understanding of text.
2. DNN models are highly flexible and capable of learning complex patterns in the data.

Comparison with Other Methods:

- Word2Vec embeddings outperform BOW and TF-IDF in capturing deeper semantic and contextual relationships.
- DNNs are more capable of leveraging dense embeddings, whereas Decision Trees might struggle with such representations.

Besides, I also use the BOW and TF-IDF feature engineering with DNN.

4. Wordvec + KNN

Feature Engineering:

1. **Word Embeddings:**
    - **Sentence Embedding Computation:**
        - Using the same pre-trained model from the SentenceTransformer library as in the previous method.
    - **Consistency Across Methods:**
        - Ensures the same embedding technique is used to allow fair comparisons between models.
2. **Label Encoding:**
    - The emotion labels are encoded into numerical values using a LabelEncoder.

Model Training:

1. **K-Nearest Neighbors (KNN):**
    - **Classifier Setup:**
        - A KNeighborsClassifier is used with n_neighbors=10.
        - The n_neighbors parameter indicates that the label of a sample is determined by the majority vote among its 10 nearest neighbors in the embedding space.
    - **Distance Metric:**
        - By default, the KNN algorithm uses Euclidean distance to compute proximity between data points.
    - **Training Process:**
        - KNN is a lazy learner, so no explicit training is performed. Instead, it stores the embedding vectors and their corresponding labels for classification during inference.
        - The model took **5888 seconds** to complete training, which is quite long.

Advantages:

1. Simplicity and Interpretability:
    - KNN is straightforward to implement and understand.
2. Non-parametric Approach:
    - The algorithm does not assume any prior distribution, making it flexible for various data types.

Limitations:

1. Computational Cost:

- KNN is computationally expensive during inference, especially for large datasets and high-dimensional embeddings like Word2Vec.

2. Sensitivity to Distance Metric:
    - Performance heavily depends on the choice of the distance metric, which may require tuning.
3. Imbalanced Classes:
    - KNN is prone to biases if the dataset contains imbalanced classes.

Comparison with DNN (Method 3):

1. **Performance:**
    - KNN is likely faster to set up but slower at inference compared to the DNN model.
    - DNN can capture non-linear relationships, whereas KNN relies solely on proximity in the feature space.
2. **Suitability:**
    - KNN works well with small datasets but may struggle with scalability.

Method 5: Word2Vec + Naive Bayes

Feature Engineering:

1. **Word Embeddings:**
    - **Sentence Embedding Computation:**
        - Pre-trained embeddings from the SentenceTransformer library are used to represent each text sample as a dense, fixed-size vector.
    - **Embedding Consistency:**
        - The same embedding method is used across all models to ensure fair comparison.
2. **Label Encoding:**
    - Emotion labels are encoded into numerical values using a LabelEncoder.

Model Training:

1. **Naive Bayes Algorithm:**
    - **Classifier Type:**
        - Naive Bayes is typically used with text data transformed into word frequencies or probabilities.
        - However, dense sentence embeddings are numerical and continuous, which deviates from the usual input type. For this reason, **Gaussian Naive Bayes** (GNB) is used, as it can handle continuous input data.
    - **Feature-Label Independence Assumption:**
        - GNB assumes independence among features and that each feature follows a Gaussian distribution, which may not perfectly fit embeddings but provides a useful baseline for comparison.
2. **Training Process:**

- The sentence embeddings are fed as input to the Gaussian Naive Bayes classifier, and the corresponding encoded labels are used for supervision.
- Highlight the speed of training and inference, as Naive Bayes is generally computationally efficient. Taking only 1839s.

Advantages:

1. **Simplicity and Speed:**
   - Naive Bayes is fast to train and test, making it a practical choice for baseline models.

Limitations:

1. **Assumption of Feature Independence:**
   - The independence assumption is unlikely to hold for dense sentence embeddings, which capture highly interdependent semantic relationships.
2. **Fit to Word2Vec Embeddings:**
   - Gaussian Naive Bayes may not fully leverage the rich information in Word2Vec embeddings due to its reliance on feature-level independence and Gaussian distribution.
3. **The data is too large for it:**
   - Naive Bayes performs well with small datas.

Comparison with KNN and DNN:

- Unlike KNN and DNN, Naive Bayes does not leverage non-linear relationships in embeddings, which may limit its classification power for nuanced emotions.

Method 6: Text Content + Hashtags Using NN

Feature Engineering:

1. **Text Content (TF-IDF)**.
2. **Hashtags (Word2Vec):**
   - **Challenge with Hashtags:**
     - Few repeated hashtags can lead to sparsity if treated using traditional methods like one-hot encoding.
   - **Word Embedding Transformation:**
     - Use Word2Vec to convert each hashtag into a dense vector.
3. **Emotion Labels (One-Hot Encoding)**

41021112L 王玟婷

Model Architecture (Neural Network):

1. **Input Layers:**
    - Two separate inputs:
        - TF-IDF features from content.
        - Word2Vec embeddings from hashtags.
2. **Hidden Layers:**
    - **TF-IDF Branch:**
        - Dense layer with ReLU activation processes the sparse TF-IDF input.
        - Optional dropout can be applied to mitigate overfitting.
    - **Hashtag Branch:**
        - Dense layer with ReLU activation processes the Word2Vec embeddings.
        - Dropout may be added for regularization.
3. **Concatenation Layer:**
    - Outputs from the TF-IDF and hashtag branches are concatenated.
4. **Fully Connected Layers:**
    - Further dense layers with ReLU activation are added to learn joint representations from the combined features.
5. **Output Layer:**
    - A dense layer with a **softmax activation function** outputs probabilities for each emotion class.

Model Training:

1. **Loss Function:**
    - **Categorical Crossentropy**, suitable for multi-class classification tasks with one-hot encoded labels.
2. **Optimizer:**
    - **Adam Optimizer** for efficient gradient updates.
3. **Evaluation Metrics:**
    - loss: 280.88 (It is abnormally high.)

Overall Improvement:

1. Maybe don’t try too much method. Focus on data processing. Then, choose one to two suitable model to train. Evaluation is important. Using training data to draw a confused metrics and error analysis. After that, optimize the model, which I believe would be more accurate than trying different methods.

41021112L 王玟婷

2. Maybe Extracting hashtags is less important. The word in hashtags is less repeated and sometimes contains meaningless words. Also, there are “no hashtags” in some ids. They somehow influence the result.

3. Remember to save every version. I forgot to submit the Naive Bayes output and I found this issue after submission deadline…

4. Among all my submissions, the DNN + word2vec has the best performance. It might because:

(1) Word2vec can capture the order and a little bit semantic in text, which helps the prediction.

(2) Decision Tree is suitable for smaller database. DNN is better handling large database.

5. I have tried using LLM embedding, but I don’t have that much RAM. Maybe I should split it to different datasets to train.