Sentiment Analysis

Arda Derbent, Anton Yahorau
Group 21

June 1, 2023

1 Preprocessing

1.1 Introduction

Text Preprocessing is an essential step in any Natural Language Processing (NLP) task. It involves preparing the text data for further analysis.

1.2 Steps in the Preprocessing Function

The function under consideration takes a dataset and applies a series of transformations to clean the tweet text data. The following steps are taken to preprocess the tweet data:

1. **URL Removal:** URLs in the tweet are removed. We assumed URLs do not carry meaningful information for text analysis in our project and removed them.
2. **Lowercasing:** The text is converted to lowercase. This standardizes the text and ensures that the same words with different cases are not treated as distinct entities.
3. **Expanding Contractions:** Contractions are expanded to their full form (for example, "don't" is expanded to "do not"). This is done to make sure we will get rid of all the stopwords. Might not be fully necessary.
4. **Removing Non-Alphabetic Characters:** Non-alphabetic characters are removed from the text. This includes punctuation, numbers, and special characters. This leaves only the words for further processing.
5. **Removing Duplicate Letters:** Consecutive duplicate letters in words are reduced to a single instance of the letter.
6. **Tokenization:** The tweet is split into individual words or "tokens". This is necessary for further steps such as stopword removal and stemming.

7. **Stopword Removal:** Commonly used words (such as "the", "a", "an") that do not carry much meaningful information for the analysis are removed.

8. **Stemming:** Words are reduced to their root or base form ("stems"). For example, "running" is stemmed to "run". This is done to reduce the dimensionality of the data and to treat different forms of the same word as a single entity.

Once all the above steps are performed, the cleaned tweet is added to the original dataset as a new column. Finally, the function removes any duplicate entries based on this cleaned text, to ensure each entry is unique.

1.3 Steps in the Data Balancer Function

This function separates the data based on sentiment classes, then it balances the classes by undersampling the majority classes, or removes the neutral sentiment class if desired. The following steps are followed:

1. **Separate Data:** The data is separated based on sentiment classes into negative, positive, and neutral.

2. **Determine Desired Instances:** The desired number of instances per class is determined. In this case, it is equal to the number of instances in the positive class.

3. **Balance Classes:** If the function is instructed to balance the classes:

   (a) The negative class is undersampled to match the desired number of instances.

   (b) The neutral class is undersampled to match the desired number of instances.

   (c) The undersampled negative, positive, and neutral classes are concatenated to create the processed data.

4. **No Balancing:** If the function is not instructed to balance the classes, the negative, positive, and neutral classes are concatenated without undersampling.

5. **Remove Neutral Sentiment:** If instructed, the neutral sentiment is removed from the processed data.

2 SVM Classification

2.1 Introduction

Support Vector Machines (SVM) is a powerful algorithm used for classification tasks. The following parts describe the implementation of an SVM classifier and an objective function used for hyperparameter optimization.

2.2 SVM Classification Function

The SVM classifier is trained on a vector representation of the training data and corresponding labels. The following steps are followed in this function:

1. **Train SVM Classifier:** The SVM classifier is trained using a linear kernel with predefined parameters.
2. **Predict Labels:** The trained classifier is used to predict the sentiment labels of the test data.
3. **Define Scoring Metric:** The F1 score is used as the scoring metric. This is a weighted average of the precision and recall.
4. **Cross-Validation:** The performance of the classifier is validated using a 5-fold cross-validation method.
5. **Evaluation:** The classifier's performance is evaluated using a confusion matrix and classification report which includes accuracy, precision, recall and F1-score.

3 Objective Function for Hyperparameter Optimization

This function is designed to optimize the hyperparameters of the SVM classifier. The following steps are taken:

1. **Define Search Space:** The function defines a search space for the hyperparameters 'C', 'kernel', and 'gamma'.
2. **Train SVM Classifier:** The SVM classifier is trained with the hyperparameters sampled from the search space.
3. **Predict Labels:** The trained classifier is used to predict the sentiment labels of the test data.
4. **Calculate F1 Score:** The F1 score is calculated for the predictions.
5. **Return F1 Score:** The function returns the F1 score. The goal of hyperparameter optimization is to find the parameters that maximize this score.

4 Confusion Matrix

The confusion matrix gives us a tabular representation of the performance of our classifier:

```
[[417  47]
 [ 53 359]]
```

Each row in the confusion matrix represents the instances in an actual class, while each column represents the instances in a predicted class. Here:

- The classifier correctly identified 417 negative instances (true negatives).
- It incorrectly classified 47 positive instances as negative (false negatives).
- It incorrectly classified 53 negative instances as positive (false positives).
- It correctly identified 359 positive instances (true positives).

5 Cross-Validation Score

The cross-validation score, in this case, is 0.86. This score is calculated by dividing the data into five subsets and training and testing the model five times, each time using a different subset as the testing set. This score gives a more robust measure of the model's performance than just a single train-test split.

6 Results

The classification report provides a breakdown of the main classification metrics, including precision, recall, and the F1-score for each class, as well as the overall accuracy of the model:

	precision	recall	f1-score	support
negative	0.89	0.90	0.89	464
positive	0.88	0.87	0.88	412
accuracy			0.89	876
macro avg	0.89	0.89	0.89	876
weighted avg	0.89	0.89	0.89	876

with hyperparameters:

```
kernel='sigmoid', C=61.076537443247815,
gamma=0.5021705322081941, random_state = 10
```

Here:

- **Precision** is the ratio of correctly predicted positive observations to the total predicted positives. High precision indicates a low false-positive rate.
- **Recall (Sensitivity)** is the ratio of correctly predicted positive observations to the total actual positives.
- **F1 Score** is the weighted average of Precision and Recall. It tries to find the balance between precision and recall.

- **Support** is the number of actual occurrences of the class in the specified dataset.

Before we improved our preprocessing our results:

```
[[405  72]
 [ 52 417]]
cross-val(f1) = 0.8696092804730305
              precision    recall  f1-score   support

    negative       0.89      0.85      0.87       477
    positive       0.85      0.89      0.87       469

    accuracy                           0.87       946
   macro avg       0.87      0.87      0.87       946
weighted avg       0.87      0.87      0.87       946

with the hyperparameters:

'kernel': 'linear', 'C': 1.8215151565195467,
'gamma': 0.015071918797023796, random_state = 10
```

7 Conclusion

The classifier shows good performance with an overall accuracy of 0.89 and balanced precision, recall, and F1-scores across classes. The cross-validation score indicates that the classifier's performance is consistent across different subsets of the data.

8 Challenges and Future

8.1 Presentation and discussion on the challenges, results, and findings during the project so far, and potential corrections to the previous assumptions. This should also be included in the report. (2 points)

Correcting previous assumptions:

- Our SVM was corrected to a binary classification problem. Otherwise, the evaluations were not good.
- We made the preprocessing much more robust, which increased our evaluation scores.

Challenges:

- Figuring out what type of preprocessing features we wanted to include was a challenge. We ended up not doing spell checking, because it caused more issues than it did good.
- Due to preprocessing taking too much of our time, the current SVM classification is a bit basic; however, its hyperparameters were optimized.
- For some reason, the positive and negative sentiment results are not equal, these will be later debugged and corrected.

Results:

- Our initial comparisons for the results cannot be made now, mainly because the comparison we wanted to make was a binary unbalanced implementation of SVM, which would require more specialized algorithms compared to our basic SVM implementation.
- We managed to raise our evaluation scores higher when we improved our preprocessing and expanded the contractions in the text.

8.2 Discussion on the plans for finishing the project, e.g., what experimental results do you want to show in the final report. (1 point)

We will be doing a neural network sentiment analysis algorithm, which we will compare with the results we have obtained now. We might also further improve our classification algorithm, make the vectorization and feature extraction more robust.