SSN MLRG1 at SemEval-2018 Task 3: Irony Detection in English Tweets Using MultiLayer Perceptron

Rajalakshmi S, Angel Deborah S, S Milton Rajendram, Mirnalinee T T
SSN College of Engineering
Chennai 603 110, Tamil Nadu, India
rajalakshmis@ssn.edu.in, angeldeborahs@ssn.edu.in
miltonrs@ssn.edu.in, mirnalineett@ssn.edu.in

Abstract

Sentiment analysis plays an important role in E-commerce. Identifying ironic and sarcastic content in text plays a vital role in inferring the actual intention of the user, and is necessary to increase the accuracy of sentiment analysis. This paper describes the work on identifying the irony level in twitter texts. The system developed by the SSN MLRG1 team in SemEval-2018 for task 3 (irony detection) uses rule based approach for feature selection and MultiLayer Perceptron (MLP) technique to build the model for multiclass irony classification subtask, which classifies the given text into one of the four class labels.

1 Introduction

Humans have the natural ability to identify the sentiment or the irony intended in a review or comment. However, identifying the intention of the user is a difficult task for the machine. Detecting irony present in a text is critical to sentiment analysis since it will inverse the polarity of the sentiment inferred (Hernandez-Farias et al., 2015).

Choice of shops, books, movies, hotels and various other services and products is influenced by comments and reviews in social media to a large extent. Huge amount of data is available in the Internet about the choices people make and their reviews about it.

Irony in texts affects the polarity of the sentiment inferred from them. Since it gives the text a meaning that is just the opposite to what is actually said, it is called as a *polarity reverser* (Farias et al., 2016). Irony is studied in various disciplines such as linguistics, philosophy and psychology. Due to the frequent use of irony in social media, its detection has gained importance in natural language processing, which faces difficulty in achieving a high performance (Liu, 2012; Wallace, 2015). The potential applications of irony detection include text mining, author profiling, detecting online harassment and sentiment analysis (Van Hee et al., June 2018). SSN MLRG1 team has already worked in sentiment analysis tasks conducted in SemEval 2017 (Angel Deborah et al., 2017a,b).

We can identify three types irony namely *verbal* irony, *situational* irony and *dramatic* irony. Subtask B in task 3 is a multiclass classification task for classifying a given tweet to one of these four classes:

1. verbal irony realized through a polarity contrast,
2. verbal irony without such a polarity contrast,
3. situational irony, and
4. non-irony.

2 Related Work

Unlike factual information, sentiment analysis and opinion mining have to deal with subjective information. Consequently, for any problem, it is important to analyze opinions collected from many people and summarize them. Social and political discussions are much harder due to complex topic and sentiment expressions, instances of sarcasm and irony (Liu, 2012). Maynard and Greenwood (2014) discusses the need for analyzing sarcasm in social media. They have developed a hashtag tokenizer for GATE (General Architecture for Text Engineering) tool and detected the sentiments and sarcasm in hashtags. Ghosh and Veale (2016) have found deep neural networks to perform better compared to Support Vector Machines (SVM) for sarcasm detection. Hernandez-Farias et al. (2015) have used MLP for automatic irony detection using the basic features from sentiment analysis and observed that MLP yields better results, compared to Naive Bayes, decision trees, maximum entropy and SVM. Barbieri and Saggion (2014) have used

Proceedings of the 12th International Workshop on Semantic Evaluation (SemEval-2018), pages 633–637
New Orleans, Louisiana, June 5–6, 2018.

random forest and decision tree for analyzing the irony and humour content in twitter dataset using Weka tool. They have used seven features for detecting imbalance, unexpectedness and common patterns.

3 System Overview

The system consists of the following modules: data extraction, preprocessing, rule based feature selection, feature vector generation and multilayer perceptron for classification.

3.1 Feature Engineering and Implementation

The dataset is cleaned and processed using functions from NLTK toolkit. We identified the keywords for irony detection using rule based feature selection. The selected features are formed into a Bag of Words (BoW) dictionary. For each sentence, feature vectors are generated by one-hot encoding method, using the sentence keywords and BoW dictionary. The feature vectors are given to the MLP and output class label is predicted. Error is calculated and backpropagated to update the weight vectors. Nadam (Nesterov-accelerated Adaptive Moment Estimation) algorithm is used for optimization.

The procedure for data preprocessing is outlined in Algorithm 2:

Algorithm 2: Data preprocessing.
Input: Input dataset.
Output: Tokenized words and their parts of speech.
begin

1. Separate labels and sentences.
2. Perform tokenization using `word_tokenize` function of the NLTK toolkit.
3. Perform Parts of Speech tagging using `pos_tag` function from the NLTK toolkit.
4. Return the tokenized words and their parts of speech which will be given as inputs to rule based feature selection.

end

The procedure for rule based feature selection and feature vector generation is outlined in Algorithm 3:

Algorithm 3: Rule based feature selection and feature vector generation
Input: Tokenized words and their parts of speech.
Output: BoW feature representation with labels.
begin

For each of the tokenized words, falling under one of the categories listed in Table 1, do the following steps.

1. Lemmatize the word using `WordNet Lemmatizer` from the NLTK toolkit.
2. Insert the lemmatized word into the dictionary.
3. Represent each sentence as a feature vector using one-hot encoding by looking up the dictionary.
4. Store the corresponding label in target vector using one-hot encoding.
5. Return the feature vector generated as the input to build the model.

end

Abbreviation	Parts of Speech
VB	verb, base form
VBZ	verb, 3rd person sing. present
VBP	verb, non 3rd sing. present
VBD	verb, past tense
VBG	verb, gerund/present participle
VBN	verb, past participle
JJ	adjective
JJR	adjective, comparative
JJS	adjective, superlative
RB	adverb
RBR	adverb, comparative
RBS	adverb, superlative
NN	noun, singular
NNP	proper noun, singular
NNS	noun plural
NNPS	proper noun, plural

Table 1: Parts of speech categories.

The procedure for building Multilayer Perceptron is outlined in Algorithm 4:

Algorithm 4: Build a Multilayer Perceptron model.
Input: BoW feature representation with labels.
Output: Learned model.
begin

1. Prepare the training dataset. `XTrain` contains the feature vectors and `YTrain` contains the target labels for irony class.
2. Build the classification model which comprises an input layer, two hidden layers and an output layer with `relu` activation function in the hidden layers and `softmax` activation function in the output layer.

3. Optimize the classification model using nadam optimizer of `keras` package for some n iterations.
4. Return the learned model.

end

For the test dataset, preprocessing is done and the feature vectors are generated from the training data BoW representation. The feature vectors are given as input to the learned model and the predicted output labels are stored.

3.2 MultiLayer Perceptron

MLP is a feedforward artificial neural network for supervised learning. MLP can be used for both classification and regression tasks. It consists of an input layer, one or more hidden layers and an output layer. Each neuron in one layer is fully connected to the neurons in next layer. Number of neurons in the output layer depends on the number of class labels in the given problem.

Each connection has a weight assigned to it. Output of each neuron is calculated by applying an activation function on the weighted sum of the inputs. Some of the common activation functions are linear, sigmoid, tanh, elu, softplus, softmax, relu, relu6, crelu, selu and relu_x.

Error value is calculated from the value predicted by the output layer and the actual class label. This error value is backpropagated and the weights and biases are updated. This procedure is repeated for the feature vectors of each input sentence. The whole procedure is repeated for some n iterations or until the error value converges to a value below a threshold.

Figure 1 depicts a simple MLP model, consisting of a single hidden layer. It takes four inputs and produces one output.

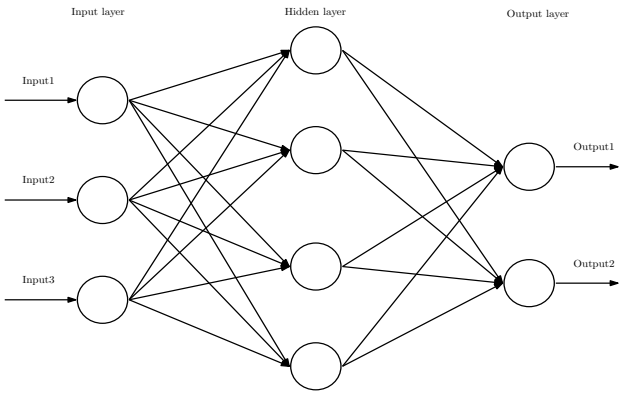

Figure 1: Multilayer Perceptron.

The working of Multilayer Perceptron model is outlined in Algorithm 1:

Algorithm 1: Multilayer Perceptron.
Input: Feature vectors and targets.
Output: Learned model.
begin

1. Initialize the weights with random values and choose a learning rate η.
2. Repeat steps 3 to 6 until the neural network is trained.
3. For each input example (feature_vector, target), do steps 4 to 6.
4. Forward Pass
   (a) For each neuron of a layer, find the weighted sum of the input vectors. Apply the activation function and pass the outputs as inputs to the next layer.
   (b) Predict the value in the output layer.
5. Backward Pass
   (a) Compute the error ∇ between the actual target and the predicted class.
   (b) Backpropagate the error and compute the error in all hidden layer neurons.
6. Update all the weights Δw_{ij} and biases b_{ij} by gradient descent technique.
7. Return the learned model.

end

4 Dataset

The dataset consists of 4792 English tweets that are collected between 01/12/2014 and 04/01/2015 from 2676 unique users. The entire corpus is split into training (80%) and test (20%) sets. The tweets are manually labeled using a fine grained annotation scheme for irony (Van Hee et al., 2016). The training dataset is further divided into training set and development test set for system building.

5 Performance Evaluation

The performance of the system is measured using accuracy, precision, recall and F1-score, using formulas shown in Equations 1 to 4.

$$\text{Accuracy (A)} = \frac{\text{TP} + \text{TN}}{\text{N}} \quad (1)$$

$$\text{Precision (P)} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (2)$$

$$\text{Recall (R)} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (3)$$

$$F_1 \text{ score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4)$$

where TP denotes True Positive, TN denotes True Negative, FP denotes False Positive, FN denotes False Negative and N denotes total number of tweets.

The optimization of the model was performed using different gradient descent algorithms such as SGD, adam, adaGrad, RMSProp and nadam. Adam and nadam are the widely used optimizers. Adam (Adaptive Moment Estimation) computes the adaptive learning rates using momentum and RMSProp. Momentum points the model in the best direction, while RMSProp adapts how far the model goes in that direction on parameter basis. Nadam combines Nesterov momentum with Adam which is superior to momentum. (Dozat, 2016).

We split the training set into training set (80%) and development test set (20%). The different optimization algorithms were used with the model and nadam optimizer produced better results compared to other algorithms for the development test set.

There are 32 submissions for this particular task. The model has achieved the following values for the various measures as listed in Table 2.

Measure	Value	Ranking
Accuracy	0.5727	15
Precision	0.3484	21
Recall	0.3609	19
F1-Score	0.3337	20

Table 2: Performance.

From the result, it appears as if the basic text features selected by rule based approach is not enough to detect the irony level in the given text. Additional features like emoticons and hashtags can be added to the feature set to enhance the performance.

6 Conclusion

We built a basic MLP to detect the irony level in twitter text, which has an input layer, two hidden layers with 128 and 64 neurons, and an output layer with 4 neurons for the four class labels. Relu activation function was used in both hidden layers and softmax activation function in output layer. The various optimizers such as SGD, RMSprop, adam, adagrad, and nadam were tried. Nadam optimizer performed better than others.

The text features were taken into consideration for BoW representation. Since irony renders an opposite meaning to the text, it is difficult to detect the irony from the text features alone. The system performance can be enhanced with the emoticon and hashtag information. The performance can also be improved by doing tweet normalization before the feature selection. The accuracy of system can be increased by using deep neural networks such as Convolutional Neural Network (CNN) or Recurrent Neural Network (RNN). Feature selection techniques can be enhanced with semantics and lexicon information.

References

S Angel Deborah, S Milton Rajendram, and T T Mirnalinee. 2017a. Ssn mlrg1 at semeval-2017 task 4: Sentiment analysis in twitter using multi-kernel gaussian process classifier. In *Proceedings the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 709–712. ACL,Vancouver, Canada.

S Angel Deborah, S Milton Rajendram, and T T Mirnalinee. 2017b. Ssn mlrg1 at semeval-2017 task 5: Fine-grained sentiment analysis using multiple kernel gaussian process regression model. In *Proceedings the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 823–826. ACL,Vancouver, Canada.

Francesco Barbieri and Horacio Saggion. 2014. Automatic detection of irony and humour in twitter. In *ICCC, Fifth International Conference on Computational Creativity, Ljubljana, Slovenia, 9th 13th June 2014*, pages 155–162.

Timothy Dozat. 2016. Incorporating nesterov momentum into adam. In *Workshop track - ICLR 2016*.

Delia Irazu Hernandez Farias, Viviana Patti, and Paolo Rosso. 2016. Irony detection in twitter: The role of affective content. *ACM Transactions on Internet Technology (TOIT)*, 16(3):19.

Aniruddha Ghosh and Tony Veale. 2016. Fracking sarcasm using neural network. In *Proceedings of the 7th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 161–169.

Irazu Hernandez-Farias, Jose-Miguel Benedi, and Paolo Rosso. 2015. Applying basic features from sentiment analysis for automatic irony detection. In *Iberian Conference on Pattern Recognition and Image Analysis*, pages 337–344. Springer.

Bing Liu. 2012. Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, 5(1):1–167.

Diana Maynard and Mark A Greenwood. 2014. Who cares about sarcastic tweets? investigating the impact of sarcasm on sentiment analysis. In *Lrec*, pages 4238–4243.

Cynthia Van Hee, Els Lefever, and Veronique Hoste. 2016. Guidelines for annotating irony in social media text. Technical report, version 2.0. Technical Report 16-01, LT3, Language and Translation Technology Team–Ghent University.

Cynthia Van Hee, Els Lefever, and Veronique Hoste. June 2018. Semeval-2018 task 3: Irony detection in english tweets. In *In Proceedings of the 12th International Workshop on Semantic Evaluation (SemEval-2018), New Orleans, LA, USA*.

Byron C Wallace. 2015. Computational irony: A survey and new perspectives. *Artificial Intelligence Review*, 43(4):467–483.