Machine Learning Engineer Nanodegree

Capstone Project

Sentiment Analysis on Amazon Product Reviews of Uplocked Mobile Phones

Suiki Lau
September 12, 2017

I. Definition

Project Overview

Sentiment analysis is often used to derive the emotion / opinion expressed in a text. It has wide applications, including analysis of product reviews, discovering a brand's presence online and people's opinion on a subject online. However, it is hard to implement sentiment analysis by machine learning because of the nature of human language, such as negation, metaphors, multiple sentiments in languages.

The goal of this project is to conduct sentiment analysis on Amazon product reviews using Natural Language Processing (NLP) techniques, Bag of Words (BoW) model, Word2Vec model and Long Short Term Memory (LSTM) neural network. The dataset consists of 400 thousand reviews of unlocked mobile phones sold on Amazon.com which is publicly available on Kaggle. Solution to the problem would be useful for a brand to gain a broad sense of user's' sentiment towards a product through online reviews.

Problem Statement

The problem is to conduct a sentiment analysis (positive and negative sentiment) on online product reviews of unlocked mobile phones sold on Amazon.com. The trained model can be used to predict users' sentiment based on their online reviews.

I have attempted to solve the problem in the following steps (see “Algorithms and Techniques” for detailed implementation).

- Step 1 : Preprocess raw reviews to cleaned reviews
- Step 2 : Use word embedding (count vectorizer, tf-idf transformation, Word2Vec) to transform reviews into numerical representations
- Step 3 : Fit numerical representations of reviews to machine learning algorithms or deep neural networks (LSTM)

Metrics

I have used accuracy score to evaluate a model with the benchmark model (see “Benchmark” section). Accuracy score computes the fraction of correct predictions. As it is a binary classification problem, accuracy score is easy to interpret and gives a single measurement for model comparison.

In addition to the accuracy score, I have also used Area Under Curve (AUC) score, precision and recall, F1 score and confusion matrix to conduct detailed evaluation on model performance. AUC score computes the area under the receiver operating characteristics (ROC) curve. ROC curve is the graphical plot of true positive rate against false position rate. By computing AUC under ROC, it gives a single number between 0 to 1, measuring the performance of a model in terms of sensitivity and false alarm. AUC score of 1 indicates a perfect model while AUC score of 0.5 indicates a model with performance similar to random guessing.

Confusion matrix is a table summarizing true positives, true negatives, false positives, and false negatives. Precision is the number of true positives divided by the total number of elements labeled as positive class. Recall is the number of true positives divided by the total number of elements that actually belong to the positive class. F1 score is the harmonic mean of precision and recall, which provides a single measurement of the model. By further investigating these metrics, we can better understand how the models perform on skewed data.

II. Analysis

Data Exploration

The dataset consists of 400 thousand reviews of unlocked mobile phones sold on Amazon.com. The data was acquired in December, 2016 by the crawlers build by PromptCloud and is available to public on Kaggle. The data set has the following fields.

- Product Title
- Brand
- Price
- Rating (1 to 5)
- Review text
- Number of people who found the review helpful

Below are some summary statistics about the data.

- Total number of reviews: 413840
- Total number of brands: 385
- Total number of unique products: 4410
- Average price: $227
- Average rating: 3.8
- Average review votes: 1.5

There are 69% reviews with positive sentiment (rating > 3), 23% reviews with negative sentiment (rating < 3), and 8% reviews with neutral sentiment (rating = 3). Apparently, there are more positive sentiments than negative sentiments in this dataset.

Exploratory Visualization

In Part 1 of my jupyter notebook, I have made the following plots for exploratory visualization.

- Figure 1 : Distribution of rating
- Figure 2 : Number of reviews for top 20 brands
- Figure 3 : Number of reviews for top 50 products
- Figure 4 : Distribution of review length

Figure 1 shows that reviews with rating of 5 dominate the distribution. In fact about 70% of the reviews have positive sentiment. Figure 2 shows top 20 brands with the largest

numbers of reviews. The top 3 brands, Samsung, Blu and Apple dominate the total number of reviews. Figure 3 shows top 50 products with the largest number of reviews. Figure 4 shows the distribution of review length (number of characters). The number of reviews falls exponentially with the increase of review length. In some of the models constructed later, I have truncated long reviews if necessary.

Algorithms and Techniques

There are two main steps involved in text classification. First, we need to find a word embedding to convert text into numerical representations. Second, we fit the numerical representations of text to machine learning algorithms or deep learning architectures.

One common approach of word embedding is **frequency based embedding** such as BoW model. BoW model learns a vocabulary list from a given corpus and represents each document based on some counting methods of words. In Part 3 of my jupyter notebook, I have implemented **CountVectorizer** in sklearn to compute occurrence counting of words.

Some words might appear frequently but have little meaningful information about the sentiment of a particular review. Instead of using occurrence counting, we can use tf-idf transform to scale down the impact of frequently appeared words in a given corpus. Thus, I have also implemented **TfidfVectorizer** in sklearn to compute tf-idf weighted counting.

Another approach of word embedding is **prediction based embedding** such as **Word2Vec embedding**. In gist, Word2Vec is a combination of two techniques: Continuous Bag of Words (CBoW) and skip-gram model. Both are shallow neural networks which learn weights for word vector representations. In Part 4, I have implemented Word2Vec model to create our own word vector representations using gensim library.

Once we have numerical representations of the text data, we are ready to fit the feature vectors to supervised learning algorithms, such as Multinomial Naive Bayes, Logistic Regression and Random Forest. We can further streamline the workflow by using pipeline and tune hyper-parameter using grid search in sklearn.

LSTM networks are a special kind of Recurrent Neural Networks (RNN), capable of learning long-term dependencies. LSTM can be very useful in text mining problems since it involves dependencies in the sentences which can be captured in the "memory"

of the LSTM. In Part5, I have trained a simple LSTM using Keras library. A simple LSTM consists of an embedding layer as input layer, a LSTM layer as hidden layer and a dense layer as output layer. Finally, I have constructed a LSTM with Word2Vec embedding which further takes into account the semantic similarity of words.

At the end of the project, I have constructed the following models for evaluation.

- CountVectorizer + Multinomial Naive Bayes
- TfidfVectorizer + Logistic Regression
- Word2Vec Embedding + Random Forest Classifier
- Simple LSTM
- Word2Vec Embedding + LSTM

Benchmark

The benchmark model is to use CountVectorizer and Multinomial Naive Bayes model. The reviews are first converted into feature vectors of word occurrence count by CountVectorizer. The feature vectors and the sentiment labels of the training data are then fit to Multinomial Naive Bayes model. A simple run in sklearn using the default setting of CountVectorizer and Multinomial Naive Bayes on 10% of the training data gives 91.84% accuracy score.

III. Methodology

Data Preprocessing

For illustrative purpose, I have only considered data with positive sentiment (rating = 4, 5) and negative sentiment (rating = 1, 2) and worked on 10% of the training data due to limitation of computational resources. To prepare the data, reviews with ratings greater than 3 will be labeled and encoded as “1” (positive sentiment) and reviews with ratings less than 3 will be labeled and encoded as “0” (negative sentiment). Neutral reviews (rating=3) will not be used. Since it is a huge data set, the labeled data have been split into training set and validation set in 90/10 so that the model could be trained on more data to improve accuracy.

Text preprocessing is needed to convert raw reviews into cleaned review. Necessary steps include conversion to lowercase, removal of non-characters, removal of stopwords, removal of html tags. Depending on machine learning algorithms or deep

architectures used, cleaned text data are further transformed to suitable numerical representations, such as 1D feature vector for supervised learning algorithms or 2D tensor for LSTM.

Implementation

In Part 3 “Bag of Words” of my jupyter notebook, the following steps have been implemented.

- Step 1 : Preprocess raw reviews to cleaned reviews
- Step 2 : Create BoW using CountVectorizer / Tfidfvectorizer in sklearn
- Step 3 : Transform review text to numerical representations (feature vectors)
- Step 4 : Fit feature vectors to supervised learning algorithm using MultinomialNB and LogisticRegression in sklearn
- Step 5 : Build a pipeline of vectorizer and classifier and improve the model performance by GridSearch in sklearn

In Part 4 “Word2Vec”, the following steps have been implemented.

- Step 1 : Parse review text to sentences (Word2Vec model takes a list of sentences as inputs)
- Step 2 : Create vocabulary list using Word2Vec model in gensim
- Step 3 : Transform each review into numerical representation by computing average feature vectors of words therein
- Step 4 : Fit the average feature vectors to RandomForestClassifier in sklean

In Part 5 “LSTM”, the following steps have been implemented.

For simple LSTM,

- Step 1 : Prepare text data to 2D tensor in Keras
- Step 2 : Train a simple LSTM network consisting of an embedding layer, a LSTM layer and a dense layer
- Step 3 : Compile and fit the model using log loss function and ADAM optimizer

For LSTM with Word2Vec Embedding,

- Step 1 : Load pretrained Word2Vec model (trained in Part 4)
- Step 2 : Construct embedding layer using embedding matrix as weights
- Step 3 : Train a LSTM with Word2Vec embedding
- Step 4 : Compile and fit the model using log loss function and ADAM optimizer

All models are evaluated using accuracy score on validation set, and supplemented by AUC score, classification report and confusion matrix for more detailed evaluation.

Refinement

I have started with the benchmark model of CountVectorizer with Multinomial Naive Bayes (accuracy score = 91.84%). The model performance is then improved by using Grid Search on TfidfVectorizer with Logistic Regression (accuracy score = 95.63%).

I have also explored the potential of model enhancement using Word2Vec Embedding with Random Forest Classifier (accuracy score = 92.26%), simple LSTM (accuracy score = 94.14%, Word2Vec Embedding with LSTM (accuracy score = 94.40%).

IV. Results

Model Evaluation and Validation

The table below summarizes the accuracy score on validation set of various models trained on 1% and 10% of the training data respectively. .

Model	**Trained on 1% of training data**	**Trained on 10% of training**
CountVectorizer + Multinomial Naive Bayes	88.35%	91.84%
TfidfVectorizer + Logistic Regression	89.32%	93.10%
TfidfVectorizer + Logistic Regression	91.59%	95.63%

(best parameter set by GridSearch)		
Word2Vec Embedding + Random Forest Classifier	84.79%	92.26%
Simple LSTM	90.61%	94.14%
Word2Vec Embedding + LSTM	88.35%	94.40%

Justification

LSTM is proven to be very useful in many text classification problems, whereas Word2Vec embedding provides additional information on the semantic similarity of words. Thus, the final model of LSTM with Word2Vec embedding is expected to outperform other models. The results show that the final model has better performance than the benchmark model, but it does not outperform every models (TfidfVectorizer with Logistic Regression has better performance).

One possible reason is that I have just used 10% of training data to train the model. It is expected that deep neural network like LSTM could deliver better performance by training on a larger data set. Second, I have not done any hyperparameter tuning for LSTM. To further enhance the performance, we could conduct random search to find the best parameter set.

V. Conclusion

Free-Form Visualization

In Part 6 of my jupyter notebook, I have created word clouds for positive sentiment reviews and negative sentiment reviews of a selected brand, to get an intuition of words that frequently appear in different sentiments. For example, reviews with positive sentiment on Apple's products frequently contain words such as "good", "great", "phone", whereas reviews with negative sentiment frequently contain words such as "battery", "unlocked", "n't".

Recall that in Part 2 of my jupyter notebook, I have implemented TfidfVectorizer with Logistic Regression and investigated the following results.

- Top 10 features with smallest coefficients : ['not' 'return' 'disappointed' 'waste' 'horrible' 'worst' 'poor' 'slow' 'stopped' 'doesn'].
- Top 10 features with largest coefficients : ['great' 'love' 'excellent' 'perfect' 'good' 'easy' 'best' 'far' 'amazing' 'awesome']

This visualization helps us to justify a bit about how logistic regression learn classifying positive and negative sentiment in this setting.

Reflection

The most difficult part in text classification is to understand the meaning of words and handle ambiguity in human languages. Word embedding plays a crucial rule in text classification, not only because it transforms text into numerical representations which allows us to fit into machine learning algorithm, word embedding like Word2Vec also learn semantic similarity between words (famous “King - Man + Woman = Queen”).

Training a deep neural network is computationally expensive, conducting random search / grid search for a deep neural network is even more expensive as there are numerous hyperparameter for deep neural networks (eg. number of epochs, number of neurons, number of hidden layers, etc.). With limited computational power, it is not easy to train on large data set as well. There is always a tradeoff between model accuracy and computational time.

Improvement

We can further improve the final model of LSTM with Word2Vec Embedding by using more training data (I have only use 10% of the data!). We can also use pretrained word embedding such as GloVe, fastText which are trained on much larger data set from external resources. It should give better performance and save time for training our own word embedding. Besides, we can try different LSTM architectures to enhance performance, for instance a deeper network, more neurons, etc., but we should also consider the tradeoff between computational time and model complexity.

References

- Data Source from Kaggle https://www.kaggle.com/PromptCloudHQ/amazon-reviews-unlocked-mobile-phones
- *"Working with text Data"* from sklearn http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
- *"Using pre-trained word embeddings in a Keras model"* from Keras Blog https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html
- *"Deep Learning with Word2Vec"* from Gensim https://radimrehurek.com/gensim/models/word2vec.html
- *"Deep Learning, NLP, and Representations"* http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/)
- "An Intuitive Understanding of Word Embeddings: From Count Vectors to Word2Vec" https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/
- *"Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras"* https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/
- *"Embedding and Tokenizer in Keras"* http://www.orbifold.net/default/2017/01/10/embedding-and-tokenizer-in-keras/