Sentiment Analysis on Movie Reviews

University of Delaware

CISC 882 Natural Language Proccessing

Final Project

Ye Fan, Lu Zhan, Mengdie Tao, Chen Ling

Background

Sentiment Analysis is also known as Opinion Mining, which is a field within Natural Language Processing (NLP) that builds systems that try to identify and extract opinions within the text (Bo and Lillian, 2008). Currently, sentiment analysis has become a topic with great interests and developments since it has many practical applications, including social media monitoring, marketing analysis, customer service, and product analytics. With the advantage of growing data size in most areas, a large number of texts expressing opinions and attitudes are available in review sites, blogs, and social media.

Project Details

The objective of this project is to apply different machine learning and deep learning methods in the task of sentiment analysis of movie reviews. Specifically, there are multiple movie review websites such as Rotten Tomatoes, IMDB, and Flixster, where users can rate based on their own feelings. Our sentiment analysis project aims at using raw movie review text with associated labels from IMDB to classify phrases on a scale of five classes: negative, somewhat negative, somewhat positive, positive (We use numerical value 0, 1, 2, 3 to represent them respectively). The challenging part of the task is dealing with obstacles like sentence negation, abbreviation, language ambiguity, and metaphors.

The full details about this project is available on our Github Repository.

Data Preprocessing

The first part of the project is data-preprocessing. Since the original dataset contains only raw text with its label, we need to transform the shape of natural language and remove noises in order to better fit our model. Therefore, the data preprocessing part contains the following steps:

> Tokenization & Segmentation
>
> Noise Removal (Remove stop words)
>
> Lemmatization & Normalization

Besides, for the purpose of contrasting with traditional NLP classification methods, it is also essential to vectorize phrases (TF-IDF, Word Embedding) in order to learn the correlation between texts rather than treating words as discrete symbols like the *bag of words* model.

Tokenization & Segmentation

With the advance of the famous Python NLP library - NLTK, we are able to tokenize and segment our movie reviews.
Specifically, Tokenizers divide strings into lists of substrings. For example, tokenizers can be used to find the words and punctuation in a string (from our dataset):

> 'One would expect cast although thornton really tries can't really blame writers shame!'
>
> ['One', 'would', 'expect', 'cast', 'although', 'thornton', 'really', 'tries', 'can', "'", 't', 'really', ...]

Noise Removal

Once we have parsed the text from our movie review dataset, the challenge is to make sense of this raw data. Text cleansing is loosely used for most of the cleaning to be done on text, depending on the data source, parsing performance, external noise and so on.

Since there is no existing package to effectively remove stop words from a sentence, we have to design specific patterns to clean our data for our task. For example, a single word 'a' is normally treated as a stop word in NLTK's stop word's list, but in our movie reviews, people usually give comments like this:

> 'I would give this movie a big A'

The lower case 'a' can be removed because it's trivial in sentiment analysis, but the upper case 'A' is really important in our sentiment analysis task. Besides, the encoding format of characters in movie reviews is inconsistent. It's often to see ‘ instead of ' in the data sample.

Therefore, we designed multiple regular expressions to extract useful information and effectively remove irrelevant stop words.

Stemming & Normalization

In English text processing, the goal of stemming is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. In our data sample, there're different variations of words and expressions, like 'movies', 'movie' and 'film' have the same meaning - 'movie', but machine may treat them separately. Thus, it is significant to effectively map related words to the same stem, even if this stem is not in itself a valid root.

In this task, we used The Porter Stemming Algorithm as the basic way of stemming. We take a sentence from our movie reviews as an example:

> Original Text: 'This guy was treated improperly in the movie '
>
> Analysis Result: 'Thi guy wa treat improperli in the movi '

Feature Selection

Feature selection plays a really important role in machine learning and natural language processing, which serves two main purposes. First, it makes training and applying a classifier more efficient by decreasing the size of the effective vocabulary. Secondly, feature selection often increases classification accuracy by eliminating noise features. In this task, we have tried three feature extraction methods: *bag of words*, *TF-IDF* and *Word Embedding*. We will introduce these three methods here briefly.

> **Bag of Words**
> A bag of words model is the most straightforward way to represent text in NLP. In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity.
>
> **TF-IDF**
> TF-IDF stands for term frequency-inverse document frequency, and the TF-IDF weight is a weight often used in information retrieval and text mining.
> TF-IDF is made up with two parts: *term frequency* and *inverse document frequency*. The number of times a term occurs in a document is called *term frequency* (TF), and the *inverse document frequency factor* (IDF) is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely.
>
> **Word Embedding**
> *Word Embedding* is currently one of the most popular representations of document vocabulary. A Word Embedding format generally tries to map a word using a dictionary to a vector. Word2vec is a great example of using Word Embedding, which is a two-layer neural net that processes text. Its input is a text corpus and its output is a set of vectors: feature vectors for words in that corpus. While Word2vec is not a deep neural network, it turns text into a numerical form that deep nets can understand.
> In this task, I mainly used pre-trained word vector (6B tokens, 400K vocab) from Stanford GloVe, which is mainly trained on English Gigaword and Wikipedia 2014.

Models

Baseline Model

For the baseline model, we chose SVM (with RBF kernel) for the sentiment classification task. Support Vector Machine tends to have better performances in relative small scale dataset comparing with other models. The performance comparison of models will be introduced in the **Statistics** section.

Machine Learning Models

Naive Bayes Classifier

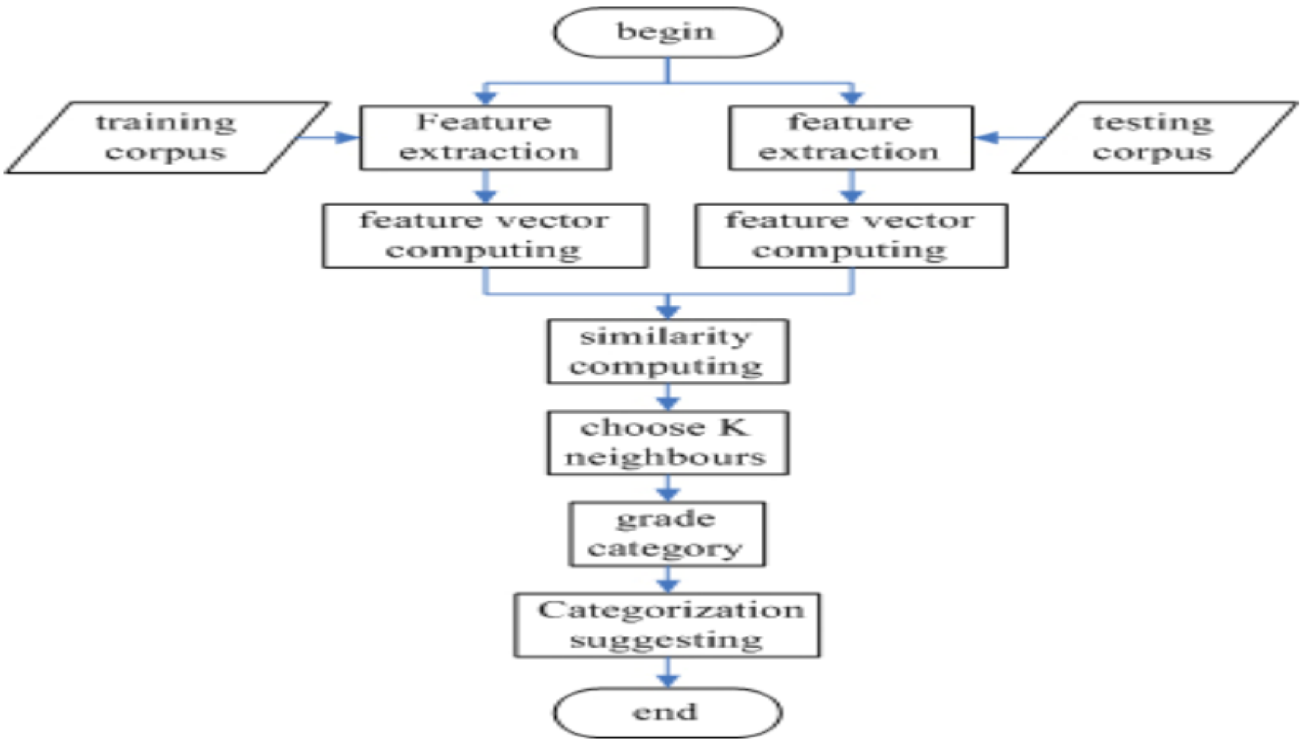
Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

k Nearest Neighbor

Traditional KNN algorithm is a simple and effective non-parametric and lazy algorithm. This algorithm is suitable for multi-modal problems. That is to say, objects have multiple labels. And KNN classifier is strongly affected by the distribution of training data in efficiency.

According to the traditional vector space model, text features are formalized as the weighted feature vector. For a given text to be classified, calculate similarity (distance) for each text in the training set. Then select the K texts with the nearest distance between the training set of documents and text sets to be classified. Determine which categories of the new text according to the above K texts category.

In this problem, we choose parameter k equal to 7, and we may have a higher score if we use higher value of k. The following is the pipeline model.



Logistic Regression

Logistic Regression is a special type of regression where binary response variable is related to a set of explanatory variables. The predictor variables can be binomial, categorical, or numerical. The goal of logistic regression is to find the best fitting model to describe the relationship between a set of variables. Logistic regression generates the coefficients of a formula to predict a logit transformation of the probability of presence of the characteristic of interest. Rather than choosing parameters that minimize the sum of squared errors (like in ordinary regression), estimation in logistic regression chooses parameters that maximize the likelihood of observing the sample values.

Whereas logistic regression for binary classification the classification task is to predict the target class which is of binary type. Like Yes/NO, 0/1,Male/Female. When it comes to multinomial logistic regression. The idea is to use the logistic regression techniques to predict the target class. We use one vs All algorithm.

About choosing solver when using Scikit-Learn's LogisticRegression model, we choose LBFGS since it use linear memory and the performance is better.

Random Forest Resemble Model

Random forest is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual tree.

> **Ensemble learning**

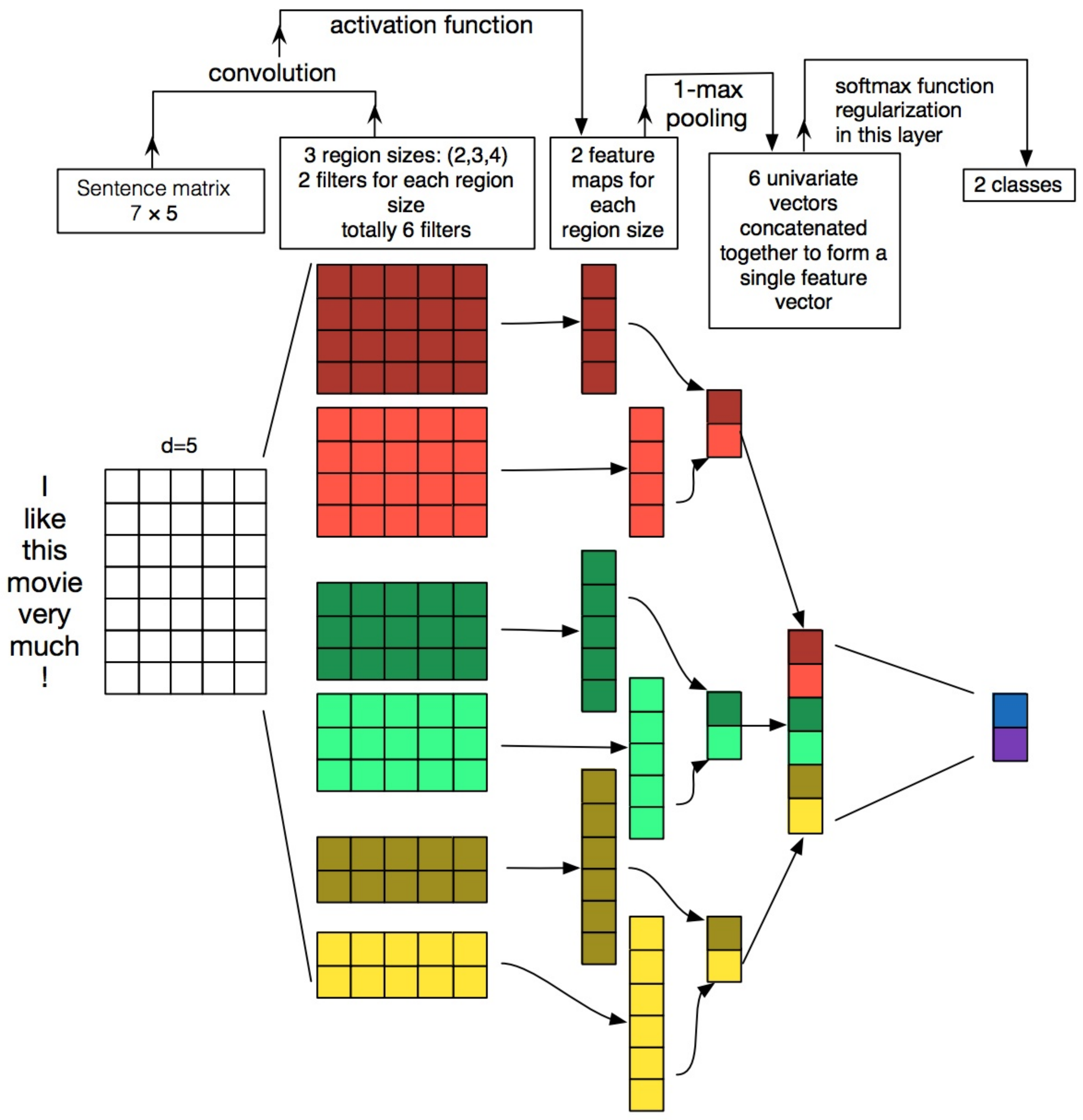
> An ensemble method is a technique that combines the predictions from multiple machine learning algorithms together to make more accurate predictions than any individual model. Bootstrap Aggregation (or Bagging for short), is a simple and very powerful ensemble method.

Decision trees are sensitive to the specific data on which they are trained. If the training data is changed (e.g. a tree is trained on a subset of the training data) the resulting decision tree can be quite different and in turn the predictions can be quite different.

The general idea of Random Forest is an ensemble of Decision Trees, most of the time trained with the "bagging" method.

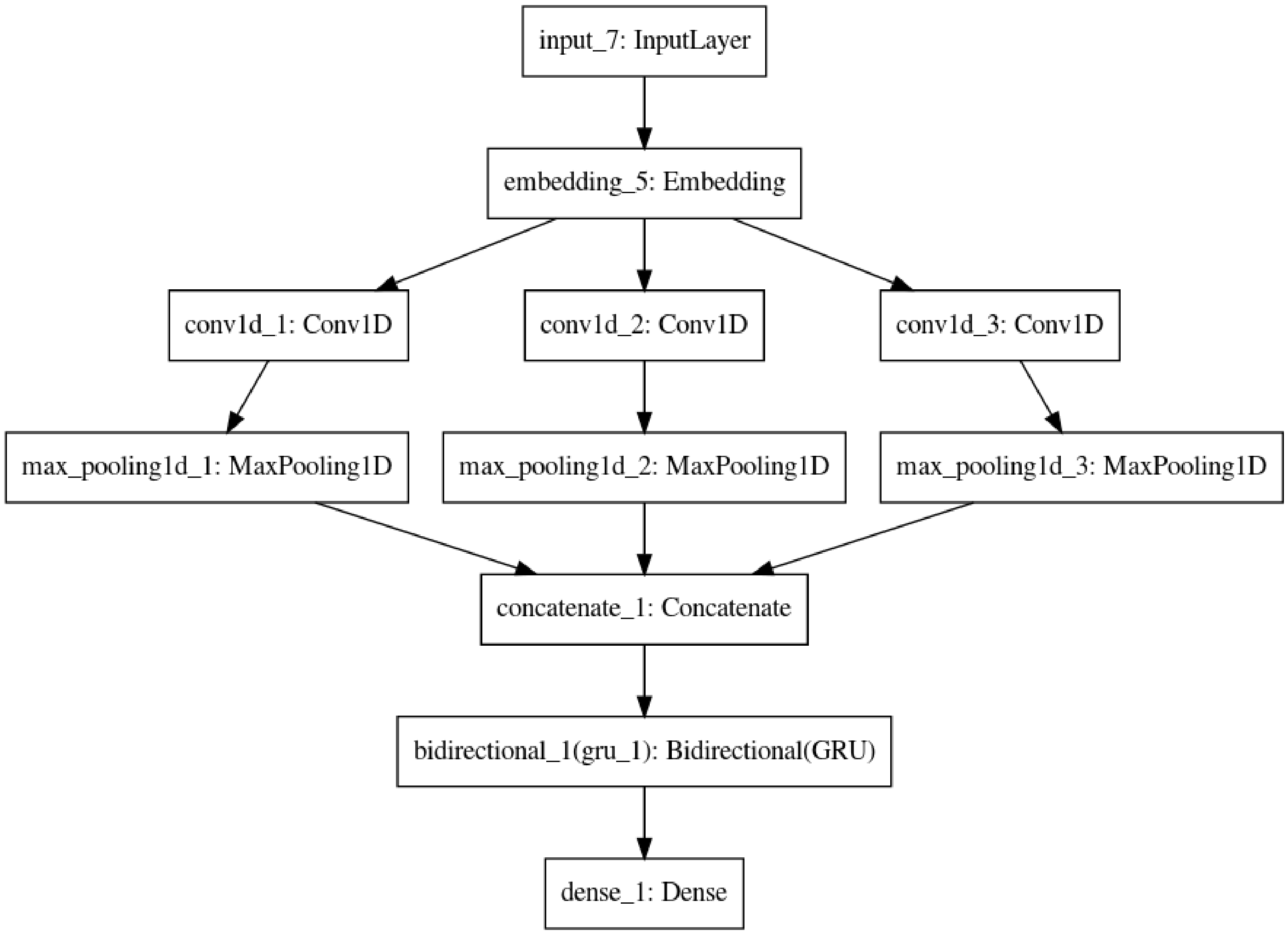
Deep Learning Model

For deep learning model, I used TextCNN as the primary model. In the area of text classification, RNN has its natural advantage because the human text has a sequential and consistent feature. However, Yoon Kim (2014) proposed that CNN can also be used in the text classification. The following graph demonstrates the structure of the CNN model:


activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I like this movie very much !

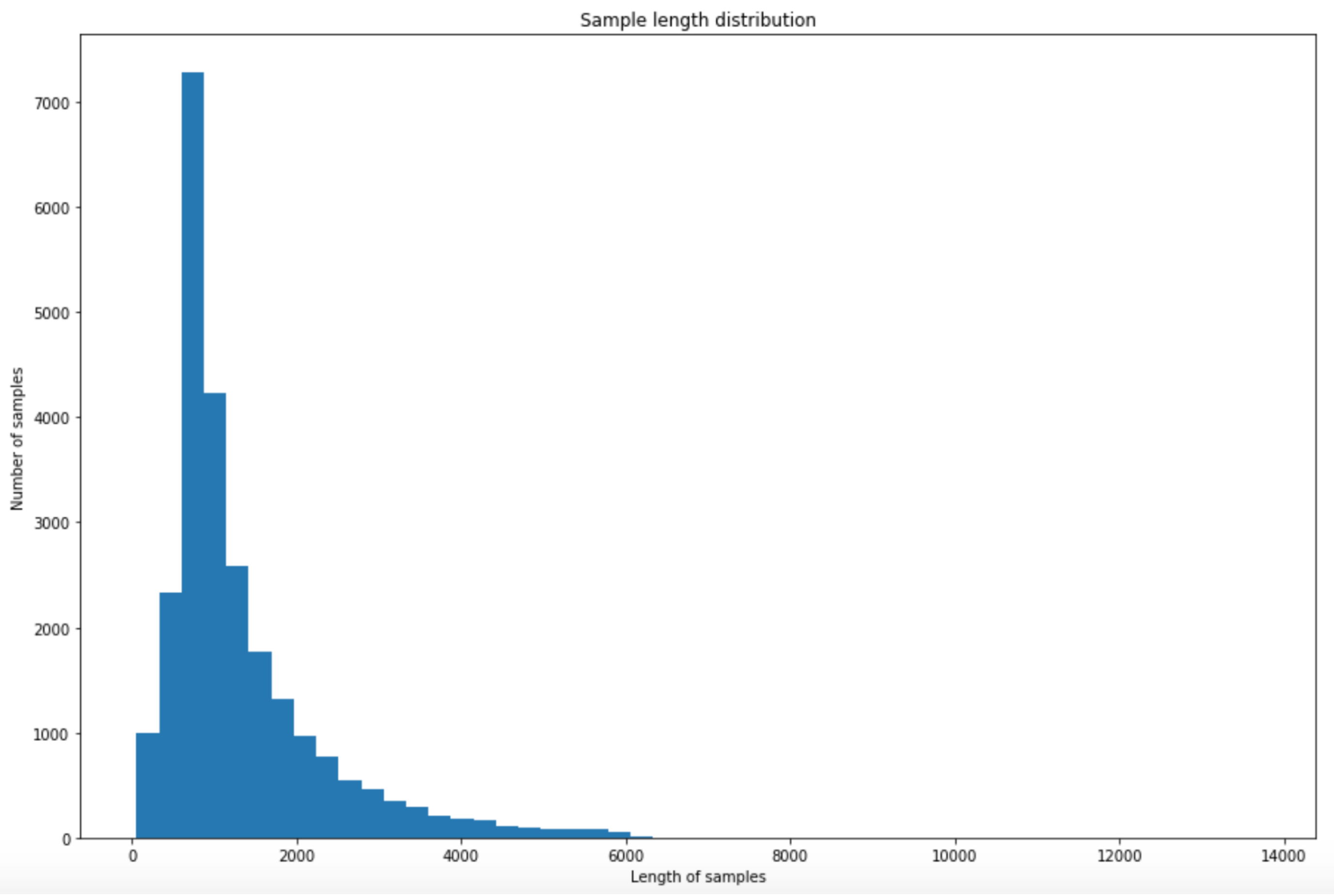
Suppose we have a sentence waiting to be classified, and the sentence is made up with multiple n-dimensional word vectors. Suppose the length of the sentence is m, which implies that the input is a m by n matrix. For this sentiment analysis task, we conduct a convolution on an input matrix. For text data, instead of horizontally sliding our filter, we only move our filter vertically, which is similar to N-Gram extracting the local correlations between word and word. I have used three strides (3, 4, 5), and we have two filters for each stride. Finally, we can get 6 vectors after the convolution process. We apply max-pooling for each vector and concatenate them into a final vector.

After getting the feature vector from the Text-CNN model, we add a RNN model in order to better capture the sequential features of the human text. We have tried different RNN models, including traditional RNN, Recurrent Neural Network(GRU) and Long Short-term Memory(LSTM), and we finally chose GRU because it has less parameters and training cost. The following is the final CRNN model.


input_7: InputLayer
embedding_5: Embedding
conv1d_1: Conv1D
conv1d_2: Conv1D
conv1d_3: Conv1D
max_pooling1d_1: MaxPooling1D
max_pooling1d_2: MaxPooling1D
max_pooling1d_3: MaxPooling1D
concatenate_1: Concatenate
bidirectional_1(gru_1): Bidirectional(GRU)
dense_1: Dense

Statistics

Since we have over 100k moview reviews, the distribution of the sequence length (in character level) is as following:



For our deep learning model, we used fixed sequence length as a hyper-parameter, if the sequence is less than the MAX_SEQ_LEN, we padded the sequence to the fixed length, and if the sequence length is larger than the MAX_SEQ_LEN, we truncate the sequence. After experiment, we found out that if we increase the MAX_SEQ_LEN, our neural network can extract more useful information from it. However, it would cost more training time.

The following table is our performance evaluation. From the table, our deep learning model - CRNN has the best score on four metrics.

	mean accuracy	**mean precision**	**mean recall**	**mean f_1**
SVM	0.56	0.59	0.56	0.58
Naive Bayes	0.53	0.53	0.51	0.51
kNN	0.44	0.47	0.44	0.41
Logstic Regression	0.54	0.54	0.54	0.53
Random Forest	0.48	0.48	0.48	0.44
CRNN	**0.5896**	**0.6458**	**0.7037**	**0.6735**

Future Improvments

- For most of multiclass movie review sentiment analysis task on Kaggle, the top performance is around 76% accuracy.

- Future improvements are on parameters tuning and more advanced feature engineering techniques (training batch, max sequence length, etc.), since our algorithm couldn't effectively filter unuseful human names and
- Our classifier is better at classifying polarized movie reviews, since reviewers tend to leave words like "terrible" and "worst" in for low rate movies, and "wonderful", "fantastic" for high rate movies. However, for those movie reviews' rate in the middle, our classifier's performance is not as good as classifying polarized reviews.

References

Kim, Yoon. "Convolutional neural networks for sentence classification." *arXiv preprint arXiv:1408.5882 (2014).*

Zhang, Ye, and Byron Wallace. "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification." *arXiv preprint arXiv:1510.03820 (2015).*

Ma, Xuezhe, and Eduard Hovy. "End-to-end sequence labeling via bi-directional lstm-cnns-crf." *arXiv preprint arXiv:1603.01354 (2016).*