Sentiment Analysis with IMDb Dataset

Le Duc Anh Tuan
Hanoi University of Science and Technology
tuan.lda204929@sis.hust.edu.vn

Nguyen Van Thanh Tung
Hanoi University of Science and Technology
tung.nvt190090@sis.hust.edu.vn

Hoang Gia Nguyen
Hanoi University of Science and Technology
nguyen.hg204889@sis.hust.edu.vn

Nguyen Huu Tuan Duy
Hanoi University of Science and Technology
duy.nht204907@sis.hust.edu.vn

Hoang Long Vu
Hanoi University of Science and Technology
vu.hl204897@sis.hust.edu.vn

Abstract

Sentiment analysis is a powerful method to obtain helpful information about a review and classify it as positive or negative sentiment. In this Sentiment Analysis with IMDb Movie Reviews Project, we wish to apply Machine Learning and Deep Learning approaches to measure the accuracy of the model and identify the best algorithm for sentiment analysis.

1 Introduction

IMDb Reviews is a large dataset for binary sentiment classification, consisting of 50,000 highly polar reviews (in English) with an even number of examples for training and testing purposes. The dataset contains additional unlabelled data. A negative review has a score ≤ 4 out of 10, and a positive review has a score ≥ 7 out of 10. No more than 30 reviews are included per movie.

The dataset is imported directly from the `tensorflow_datasets`, with the default split of 50/50 for training and test dataset, along with $50,000$ additional unlabelled examples for Unsupervised learning.

After text processing phase with Word Vectorisation, we will implement multiple algorithms to evaluate the accuracy. ML algorithms are traditional algorithms including Supervised Learning (KNN, Naive Bayes, Decision Tree, Random Forest, SVM, Logistic Regression) and Unsupervised Learning (K-Means Clustering), whereas DL algorithms work with multilayers artificial neural network and are expected to give better output. The output of the model is to classify a movie review as Positive (1) or Negative (0).

2 Data Preprocessing

2.1 Clean

We do the basic learning techniques to clean the data:

- remove "(<.*?>)" markup
- remove punctuation marks
- remove white space

- remove all strings that contain a non-letter
- convert to lower
- tokenization: the process of replacing sensitive data with unique identification symbols that retain all the essential information about the data without compromising its security.
- remove stop words

2.2 Machine Learning approach: Tf-idf

TF-IDF (term frequency-inverse document frequency) is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents.

$idf(t, D) = \log \frac{|D|}{|\{d \in D; t \in d\}|+1} = \log \frac{|D|}{df(d,t)+1}$
$tfidf(t, d, D) = tf(t, d) \times idf(t, D)$

Where:

- $|D|$ is the number of documents in the collection.
- $df(d, t) = |\{d \in D; t \in d\}|$ is the frequency of the document $d \in D$ that the word *t* appears.
- $tf(t, d)$ is the frequency of the word *t* in the document *d*.

2.3 Deep Learning approach: Word Embedding

Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located close to one another in the space.

3 Machine Learning

3.1 SVM

SVM is a linear classification method, whose main purpose is to define a separator in the search space that can separate different classes. This separator is commonly referred to as a hyperplane. One of the advantages of this SVM method is that it is quite good at classifying high-dimensional data because the method tries to determine the optimal direction of discrimination in the feature space by examining the right feature combination. Since our problem is linear (just positive and negative) here, we will go for "linear SVM" (1). The `LinearSVC` model provided by `sklearn` supports the tuning of the hyper-paremeter *C*. This parameter tells the SVM optimisation how much we want to avoid misclassifying each training example. For large values of *C*, the optimisation will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of *C* will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. The default model uses $C = 1.0$ yields the accuracy on the training and test set as 99.996% and 88.98% respectively.

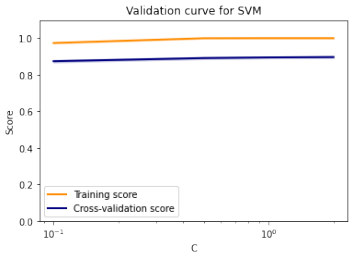


Figure 1: C from 0.4 to 0.8

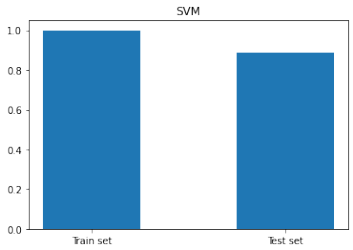


Figure 2: SVM model with $C = 0.5$

We used the Grid Search with 10-fold cross validation schema to find the best value of C. The model seems to start overfit since $C = 0.8$, and a value of 0.5 should suffice to balance the running time, and slightly reduce overfitting.

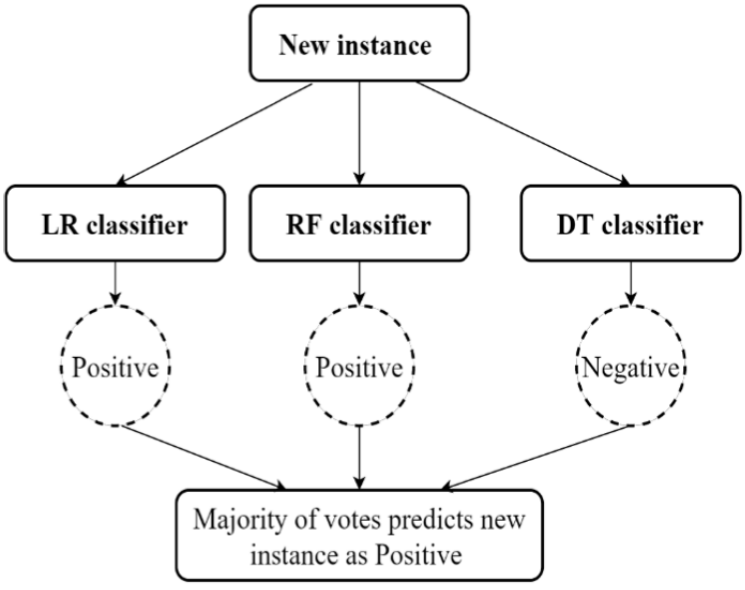
The linear SVM model with $C = 0.5$ achieves 99.64% accuracy on the train set, and 88.78% on the test set.

3.2 Voting

The intuition behind Voting Classifier is based on the Law of Large number. Suppose we toss a fair coin 4 times, then it is impossible to be sure that we will receive exactly 2 heads and 2 tails, but maybe 1 heads and 3 tails i.e. 25% of chance. By the Law of Large number, as we keep tossing the coin, the ratio of heads gets closer and closer to the probability of heads (which is 50%). Voting Classifier follows the same fashion, where we can achieve a higher accuracy than the best classifier in the ensemble provided there are a sufficient number of weak learners and they are sufficiently diverse. One way to improve the diversity is by combining multiple base classifiers in the Voting Classifier.

In this project, we propose three base classifiers: Logistic Regression (LR), Decision Trees (DT) and Random Forest (RF). (2).

Hard voting obtains an overall class prediction based on majority of predictions of ML algorithms in the ensemble.



LR uses a Logistic function in estimation of positive or negative class labels denoted by y for features of data w: $p(y = \pm 1|x, w) = \frac{1}{1+e^{-w^T h(x)}}$. After tuning using GridSearchCV, we yield that C makes

the most significant impact on the accuracy, with the optimal value of 2. The accuracy of the LR model (with $C = 2$) evaluated on the train and test set are 95.74% and 88.32% respectively.

RF grows multiple classifier trees with bootstrap sampling with replacement to train each tree with a different part of the dataset. We tried tuning `n_estimators`, `max_depth` to increase the accuracy of the model. However, the tuning does not seem to help much, thus we kept default values for these. The `criterion` hyper-parameter is tuned to yield *entropy* as the best. The accuracy on the train and test set of RF is 100% and 85.12% respectively.

DT is a widely used classifier algorithm and it is based on entropy theorem to select the attribute (feature) for node split. Previous works have demonstrated that decision trees (DT) are very effective for ensemble modeling (Bauer & Kohavi, 1999; Breiman, 1996; Drucker & Cortes, 1995; Freund & Schapire, 1996; Kim, 2018; Kim & Upneja, 2014; Margineantu & Dietterich, 1997; Quinlan, 1996; Tamon & Xiang, 2000; Zhang, Burer, & Street, 2006) (3). The Decision Tree is also tuned to yield *entropy* for the parameter `criterion` for the best accuracy, which is 100% and 85.2% for the train and test set.

The accuracy for the Voting Classifier with three base classifiers: LR, DT and RF on the test set is 86.932%.

Algorithms	Test set accuracy(%)
Logistic Regression	88.32
Random Forest	85.12
Decision Tree	85.20
Hard Voting Classifier	86.93

Table 1: Algorithms in Hard Voting Ensemble Summary

3.3 XGboost

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. XGBoost stands for eXtreme Gradient Boosting. Gradient Boosting works by sequentially adding predictors to an ensemble, each one correcting its predecessor. However, instead of tweaking the instance weights at every iteration like AdaBoost does, this method tries to fit the new predictor to the residual errors made by the previous predictor. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. XGBoost stands for eXtreme Gradient Boosting.

Gradient Boosting works by sequentially adding predictors to an ensemble, each one correcting its predecessor. However, instead of tweaking the instance weights at every iteration like AdaBoost does, this method tries to fit the new predictor to the residual errors made by the previous predictor.

The initial accuracy on test set of XGBoost is 80.676%. After trying tuning `learning_rate`, `max_depth`, `n_estimator`, `gamma`, and `subsample`. We found that `learning_rate` and `max_depth` have the most significant impacts on the accuracy of the XGBoost classifier. After narrowing down the scope and range of parameters, we use GridSearch with 10-fold CV to find the best combination.

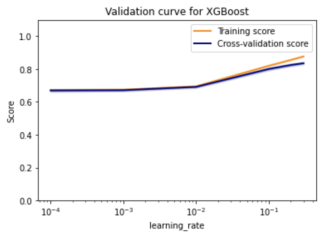


Figure 3: `learning_rate` from 0 to 0.1

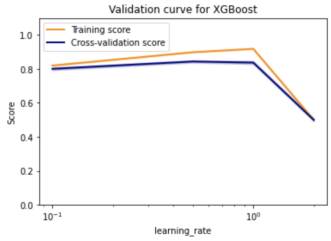


Figure 4: `learning_rate from 0 to 2`

The best combination (`learning_rate` $= 0.5$ and `max_depth` $= 7$) yields the accuracy of 97.91% for the train set and 84.628% for the test set.

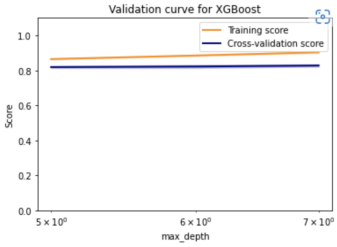


Figure 5: `max_depth` from 5 to 7

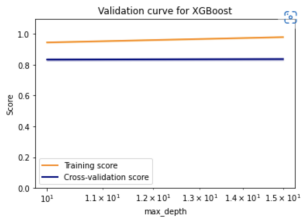


Figure 6: `max_depth` from 10 to 15

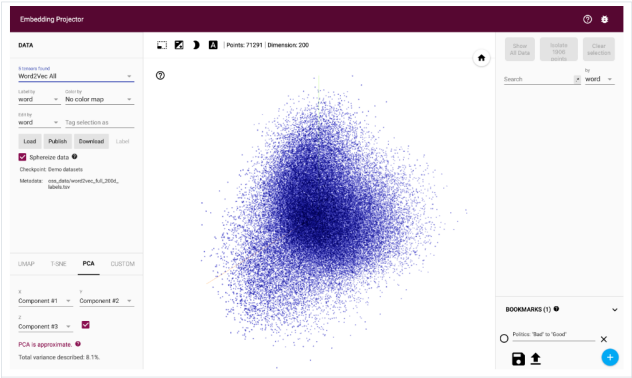
4 Deep Learning

4.1 Pretrained Word2Vec x Mini-batch KMeans

Cleaning and tokenizing data: this usually involves lowercasing text, removing non-alphanumeric characters, or stemming words.

Generating vector representations of the documents: this concerns the mapping of documents from words into numerical vectors—some common ways of doing this include using bag-of-words models or word embeddings. In this repository we used pre-trained fasttext-wiki-news-subwords-300 mix IMDb data to generate word embeddings.

Applying a clustering algorithm on the document vectors: this requires selecting and applying a clustering algorithm to find the best possible groups using the document vectors. Some frequently used algorithms include K-means, DBSCAN, or Hierarchical Clustering.



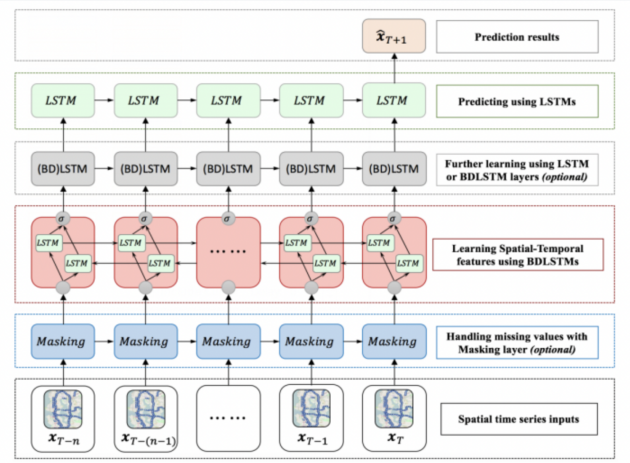
Word2Vec

4.2 Deep Bi-directional LSTM

Bidirectional RNNs is based on the idea that the output at each time may not only depend on the previous elements in the sequence, but also depend on the next elements in the sequence. For instance, to predict a missing word in a sequence, we may need to look at both the left and the right context.

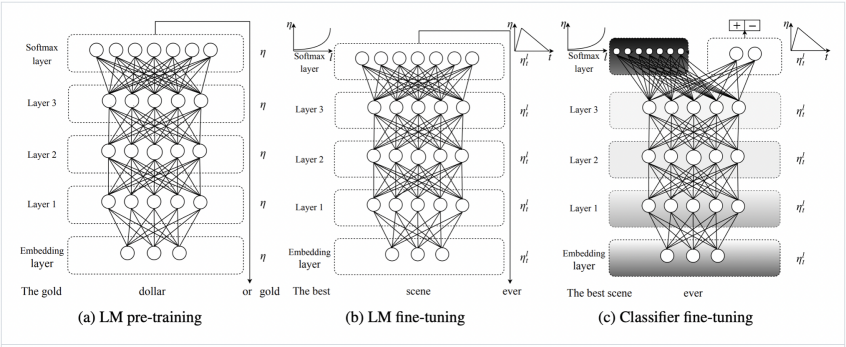
A Bidirectional LSTM, or biLSTM, is a sequence processing model that consists of two LSTMs: one taking the input in a forward direction, and the other in a backwards direction, which are stacked on the top of each other. The one that processes the input in its original order and the one that processes the reversed input sequence. The output is then computed based on the hidden state of both LSTMs. biLSTMs effectively increase the amount of information available to the network, improving the context available to the algorithm (e.g. knowing what words immediately follow and precede a word in a sentence).

Deep bidirectional LSTM is similar to bidirectional LSTM. The only difference is that it has multiple layers per time step, which provides higher learning capacity but needs a lot of training data.



Deep Bidirectional Long Short Term Memmory

4.3 Universal Language Model Fine-tuning



(a) LM pre-training (b) LM fine-tuning (c) Classifier fine-tuning

Howard et al. in Universal Language Model Fine-tuning for Text Classification

Universal Language Model Fine-tuning, or **ULMFiT**, is an architecture and transfer learning method that can be applied to NLP tasks. It involves a 3-layer AWD-LSTM architecture for its representations. The training consists of three steps:

Step 1: General language model pre-training on a Wikipedia-based text.

Step 2: Fine-tuning the language model on a target task.

Step 3: Fine-tuning the classifier on the target task.

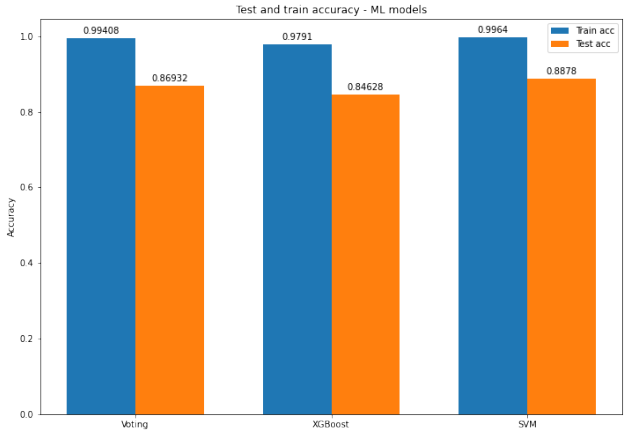
As different layers capture different types of information, they are fine-tuned to different extents using **discriminative fine-tuning**. Training is performed using **Slanted triangular learning rates** (STLR), a learning rate scheduling strategy that first linearly increases the learning rate and then linearly decays it.

Fine-tuning the target classifier is achieved in **ULMFiT** using gradual unfreezing. Rather than fine-tuning all layers at once, which risks catastrophic forgetting, ULMFiT gradually unfreezes the model starting from the last layer (i.e., closest to the output) as this contains the least general knowledge. First the last layer is unfrozen and all unfrozen layers are fine-tuned for one epoch. Then the next group of frozen layers is unfrozen and fine-tuned and repeat, until all layers are fine-tuned until convergence at the last iteration.

5 Evaluation

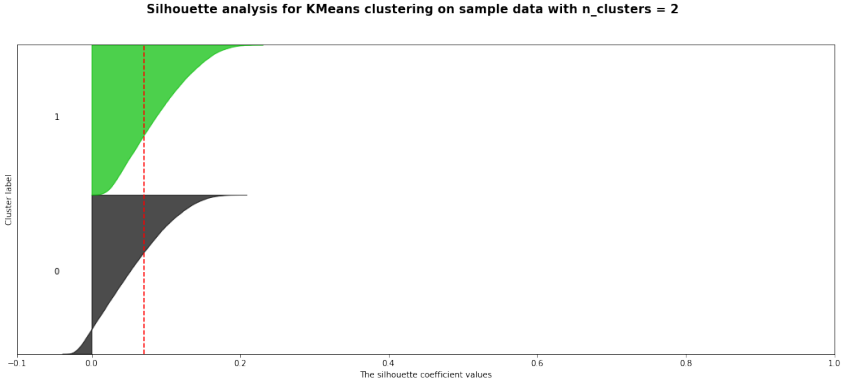
5.1 Machine Learning

Below is the plot of training and test accuracy of Machine Learning models. It can be seen that all three models achieved quite good accuracy of about 88 percent in average. However, all three models seem to overfit the training dataset and may require further hyperparameters tuning.



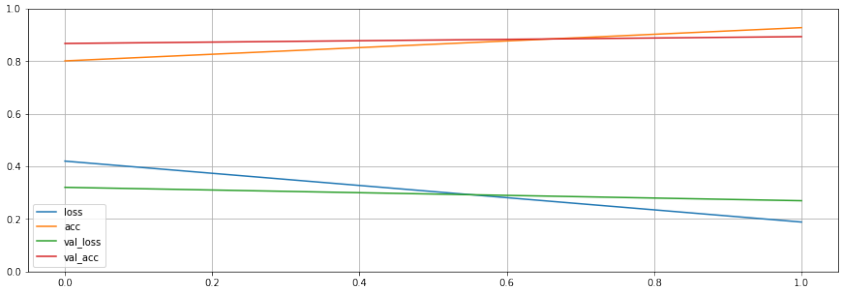
Silhouette Score is a metric to evaluate the performance of clustering algorithm. It uses compactness of individual clusters(intra cluster distance) and separation amongst clusters (inter cluster distance) to measure an overall representative score of how well our clustering algorithm has performed.

Silhouette coefficients near +1 indicate that the sample is far away from the neighboring clusters. A value of 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters and negative values indicate that those samples might have been assigned to the wrong cluster.

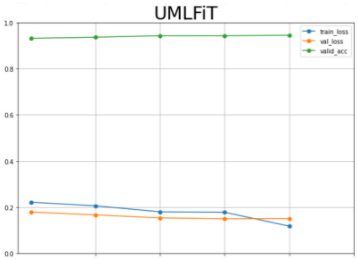


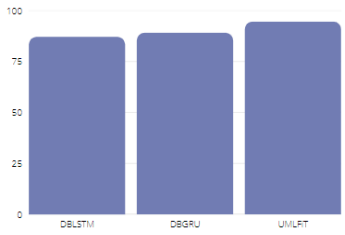
5.2 Deep Learning

5.2.1 DBGRU



5.2.2 UMLFit





References

[1] YONATHA WIJAYA, Komang Dhiyo; KARYAWATI, Anak Agung Istri Ngurah Eka. The Effects of Different Kernels in SVM Sentiment Analysis on Mass Social Distancing. JELIKU (Jurnal Elektronik Ilmu Komputer Udayana), [S.l.], v. 9, n. 2, p. 161-168, nov. 2020. ISSN 2654-5101

[2] Akın Özçift. Medical sentiment analysis based on soft voting ensemble algorithm.

[3] Soo YoungKima, ArunUpnejab. Majority voting ensemble with a decision trees for business failure prediction during economic downturns.

Appendix

DBLSTM - Deep Bi-directional LSTM

Lei Zhang, Shuai Wang, Bing Liu. Deep Learning for Sentiment Analysis: A Survey

Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, Jianfeng Gao. Deep Learning Based Text Classification: A Comprehensive Review

Mike Schuster, Kuldip K. Bidirectional Recurrent Neural Networks

Tomas Mikolov, Hya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality

Sepp Hochreiter, Jürgen Schmidhuber. LONG SHORT-TERM MEMORY

Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space

UMLFiT - Universal Language Model Fine-tuning

Stephen Merity, Nitish Shirish Keskar, Richard Socher. Regularizing and Optimizing LSTM Language Models

Jeremy Howard, Sebastian Ruder. Universal Language Model Fine-tuning for Text Classification

Leslie N. Smith. A DISCIPLINED APPROACH TO NEURAL NETWORK HYPER-PARAMETERS: PART 1 – LEARNING RATE, BATCH SIZE, MOMENTUM, AND WEIGHT DECAY