Sentimental Analysis: Final Project Report

Zujun Peng (zp2224)
Jingyuan Wang(jw4000)
Wen Chen(wc2787)
Rahul Subramaniam (rs4128)
Sung Jun Won (sw3049)

Problem Statement

In this project, we work on sentiment140 dataset on tweets contents and our goal is to make sentiment analysis to predict whether it is a positive or negative word. We take the advantages of natural language processing techniques to preprocess the text data and construct machine learning and deep learning models such as logistic regression, decision tree, lstm, and so on to conduct sentiment classification.

Data Cleaning and Data Exploration Analysis

We collect the sentiment 140 dataset from kaggle. Since the dataset has equal numbers of positive and negative labels, we obtain a balanced dataset. Moreover, the dataset has no missing values. Except for text data, the data cleaning and preprocessing we do on the whole dataset includes reorganizing the posting time of tweets, changing the positive/negative labels to 1 and 0. We will explain how we process text data in the next section.

We also conduct data exploration analysis on the distribution of the labels, sentiment change over time and different users. We also draw two word cloud images for positive tweets (left) and negative tweets (right). The left includes most frequently used words in positive tweets including *thank*, *good*, *day*, *love*; the right includes most frequently used words in negative tweets including *work*, *now*, *go*, *day*. Overall, the frequent words shown by the word cloud are good representations of the emotion. For example, people might think working is tedious and therefore make negative tweets on that.

Text Data Preprocessing

1. ***TFIDF Vectorizer***

We use Tf-idf Vectorizer and Word2Vec word embedding to preprocess text data. Later in the model training and tuning process, we will try to compare which preprocess method performs better while conducting the sentiment analysis.

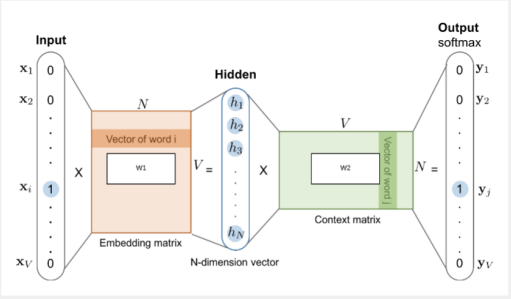
Tf-idf vectorizer counts the token frequency considering the document frequency of each token. It scales down the impact of tokens that occur very frequently in the text dataset. This can help we learn more features on the tokens that occur in a small fraction of the training text dataset. The formula of Tf-idf can be expressed as:

$$\mathbf{idf(t) = \log [\ (1 + n\ ) / (1 + df\ (t))] + 1}$$

Where n is the total number of documents in the document set and df(t) is the document frequency of t. In our project, we use a n-gram range from 1 to 3, we also set the max_df = 0.8 to further filter out the terms that have a document frequency strictly higher than the given value of max_df.

2. Word2Vec

To enrich our representation of text data, we also use Word2Vec model to train our word embedding. The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text by taking information in the context window into account. After training the model, we get a dense vector representation for each word in the text data by extracting the weights matrix updated after the training process. The following graph shows the basic structure of how a word2vec model works. In our word embedding training model, We utilize the gensim library to train our Word2Vec model. Regarding the parameter setting, we set the context word window-size to be 3, dimension of the word embedding to be 100 to obtain an informative word embedding matrix of our input text data.



	about	all	cent	cents	money	new	old	one	two	
doc	1	1	3	1	1	1	1	1	1	In theory (a)

Index	0	1	2	3	4	5	6	7	8	
doc	1	1	3	1	1	1	1	1	1	In practice (b)

Since sentences in different samples have different lengths, we take both the mean value and the maximum value of word embedding over the whole sentence. In this way, we can obtain a sentence representation of dimension 100 (same dimension as word embedding) for each sentence in the dataset, and these sentence representations can then be used in the classification model for sentiment analysis.

3. Countvectorizer using Keras Tokenizer

Countvectorizer with Keras Tokenizer allows to vectorize a text corpus, by turning each text into either a sequence of integers (each integer being the index of a token in a dictionary) or into a vector where the coefficient for each token could be binary, based on word count. Keras tokenizer also gives an option to use the tfidf method of vectorization by specifying text_to_matrix(mod='tfidf') instead of text_to_sequence(). The tokenizer allows limiting the number of embedding entries for each sentence. The project uses the Keras Tokenizer for the LSTM modelling. Above one (diagram to the right) explains how the Keras Tokenizer uses the Count Vectorizer.

Model and Techniques

Our problem is a binary classification problem, so naturally we focus on applying classification models to solve it. The models are classified into three main categories: traditional machine learning models, tree-based models and Neural Network related models.

1. Traditional machine learning models

1.1 *Logistic Regression*

Logistic regression is a classification algorithm and it is widely used to predict a binary outcome given a set of independent variables. It is easier to implement, interpret, and very efficient to train so we firstly try to fit the logistic regression model on datasets. Then we tune our hyperparameters for l1_ratio and c values by grid search technique. We basically tried three regularization methods, including L1 norm, L2 norm and Elastic-Net. Last we fitted the model with the best set hyperparameters C = 0.01 and l1_ratio = 1 on the test dataset.

1.2 *Support vector machine (SVM)*

Support Vector Machine (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. It is relatively memory efficient and is more effective in high dimensional space. In this algorithm, we plot each data item as a point in n-dimensional space (where n is the number of features in raw data) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes.

Since SVM fits slowly on large datasets, and only data near the decision boundaries benefit, we extract the first **10k** to accelerate training, also validation and test. We tuned the C parameter on our validation dataset by grid search. The parameter C regulates different penalties for the misclassified data points, and as a result affecting the margin for the decision boundary. Finally we get our optimal hyperparameter C = 1.3.

1.3 ***KNN(k-nearest neighbors)***

K-nearest neighbors (KNN) is a type of supervised learning algorithm used for both regression and classification. It does not have any parameters to learn and is also easy to implement. It tries to predict the correct class for the test data by calculating the distance between the test data and all the training points. Then select the K number of points which is closest to the test data. For similar reasons to SVM, we reduce the data size to **10K** for fast training. Tuning k from range one to forty and performing the grid search on it, we get our optimal parameter k = 33 and then fit it on the test set.

1.4 ***Comparison***

Training SVM and KNN on the full dataset takes hundreds of times compared to the logistic model, so we decided to reduce the data size while keeping the balance. As a result, the data size we use to train the three models is 96:1:1(logistic regression:SVM:KNN) to reach similar training efficiency.

Under such a situation, logistic regression performs the best in terms of accuracy.

2. Decision Tree and XGBoost (Explain why not including Word2Vec)

2.1 Decision Tree

Decision Tree is a supervised machine learning algorithm that splits the data according to some parameters (max_depth, min_samples_split, etc) and makes decisions on the leaf nodes. Decision Tree is used for both regression and classification. The training takes a huge amount of time if the data is big, and it is very prone to overfit if the tree branches out too far. Thus, we reduced the data size to 10K, just like SVM and KNN models. To find the optimal decision tree model, we did hyperparameter tuning on max_depth parameter and found the optimal parameter max_depth = 100 with the validation accuracy=0.653 (which is quite low). With this parameter, we fit the model again and evaluated the model on the test data.

2.2 XGBoost

XGBoost is gradient boosted decision trees, mainly designed to outspeed and outperform the sklearn's GradientBoostingClassifier. It has a more regularized model formalization, which means that it is able to deal with overfitting better than GradientBoostingClassifier. XGBoost is used for both classification and regression. Similar to Decision Tree(as it is a group of boosted decision trees), training takes a huge amount of time if the data is big. Thus, we reduced the data size to 10K again. We then did hyperparameter tuning on learning rate, and we chose a combination of relatively high learning rates because generally, XGBoost works well with learning rates of

0.05~0.3. Then we found the optimal parameter learning_rate = 0.3 with the validation accuracy=0.721. With this parameter, we fit the model again and evaluated the model on the test data.

2.3 Comparison

Training Decision Tree and XGBoost on the full dataset also takes much more time compared to the logistic model, similar to SVM and KNN. Thus, we again decided to reduce the data size while keeping the balance. The data size we use to train the three models is also 96:1:1
(logistic regression:Decision Tree:XGBoost) to reach similar training efficiency.
Logistic regression performs the best in terms of accuracy.

3. LSTM

3.1 Overview

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. It is an improvement over the traditional RNN Model as it helps solve the popular vanishing gradient problem. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three *gates* regulate the flow of information into and out of the cell. It has great applications in the field of Time Series Modelling, Image Captioning, and several others.

3.2 Model

The modelling is implemented using Keras, and have the following layers

- Embedding Layer: Layer to turn the TFIDF/ Tokenizer embeddings to dense vectors of fixed size.
- SpatialDropout : Drops 2D feature maps instead of normal dropout that drops single elements
- LSTM layer: We chose 176 units, but even tried with different numbers of units. (not shown in code since param search is causing crashes)
- Dense Layer: Softmax Activation with 2 nodes that returns probability of being 0 or 1 (Zero: -ve One: +ve).

Note: Theoretically a bidirectional LSTM should work better, but the epochs takes really long, hence we couldn't go ahead with it.

3.3 Results

We tried with the 2 kinds of vectorization, the keras tokenizer and the tfidf vectorizer (we could try the tfidf by specifying it as a mode parameter). The model is trained on a limited number of epochs (between 5-10) and accuracies are as shown in the results. The model achieved an AUC score of 0.82 when Keras tokenizer is applied with a single LSTM layer. We limit the number of words in both cases to be 500, since otherwise Colab is unable to handle the original sparse representation with 70,000 odd features.

Results

Since the data are perfectly balanced, we can use accuracy as one of our metrics to measure the model performance. Besides, we also introduce precision score, recall score and F1 score. For the two embedding methods, the results are shown below:

Model	Vectorization Technique	Accuracy	Precision	Recall	F1 Score
Logistic Regression	TF-IDF	**0.795**	**0.803**	**0.784**	**0.794**
	Word2vec	0.733	0.736	0.731	0.734
SVM	TF-IDF	0.755	0.77	0.728	0.748
	Word2vec	0.722	0.731	0.701	0.716
KNN	TF-IDF	0.757	0.769	0.736	0.752
	Word2vec	0.672	0.690	0.623	0.655

Decision Tree	TF-IDF	0.646	0.649	0.632	0.641
XGBoost	TF-IDF	0.704	0.715	0.677	0.695
	Keras Sequencing	0.742	0.736	0.756	0.7564
LSTM	TF-IDF	0.69	0.7	0.691	0.695

Conclusion

Compared with different models, logistic regression has the best performance than the other models followed by LSTM. Decision tree has the lowest performance. But considering the efficiency, we use only part of the dataset when training SVM, KNN and decision tree and full dataset for the other models. We also utilized three vectorization techniques: one is TF-IDF another one is Word2vec and third one is COuntVectorizer (Keras Tokenizer). Model tends to perform better after we use the TF-IDF vectorization technique than the other one (Word2vec). Exception is observed incase of LSTM models.

Reflection and Improvement

- The major limitation of Logistic Regression is the assumption of linearity between the dependent variable and the independent variables.
- Both SVM and KNN are not suitable for large and high dimensional datasets.
- WRT LSTM, we would prefer using Bidirectional LSTM over a simple LSTM. However, given the training contains 960,000 rows, coupled with the lack of GPU resources, we have to settle with a single LSTM layer.
- The entire set of sparse representations is not considered in case of LSTM and is limited to 500. Ideally we would like a system that has the capability to run as many representations.