COMP30027 Machine Learning 2020
Sentiment Analysis
Group report

Anonymous
word count: 2430

1. Introduction

Sentiment prediction is widely used to systematically identify, extract, quantify, and study affective states and subjective information. In this project, we aim to develop several Machine Learning systems to predict the star rating of a review (1, 3 or 5). The full dataset is from the two research: *'What Yelp fake review filter might be doing? A. Mukherjee, V. Venkataraman, B. Liu, and N. S. Glance, ICWSM, 2013.'* and *'Collective Opinion Spam Detection: Bridging Review Networks and Metadata. S. Rayana, L. Akoglu, ACM SIGKDD, Sydney, Australia, August 10-13, 2015.'* It contains the meta features and text reviews. In different systems we come out, we have used different text encoding methods to transform the text documents (using CountVectorizer, TF-IDF and Doc2Vec with 50, 100 and 200 features). Depending on the form of encoded text, we also applied feature selections on them. Followed by conducting several simple or ensemble classifiers: Logistics Regression, Naive Bayes, Stacking and Support Vector Machines, to train the processed data. Our experiment has shown that this decision making process is effective in improving the performance of classifiers and building an ideal Machine Learning system for text classification problems.

2. Methods

A complete machine learning system for this case study includes encoding method, feature engineering and classifier. Generally, we decide the classifiers at the start so we can determine which encoding method does the best and follows by feature selection.

2.1 Classifiers

2.1.1 Logistic Regression (LR)

The first classifier we have implemented is Logistic Regression (LR). Since our task is to predict the ratings with three classes (1,3 or 5). The LR implemented here is a Multinomial logistic regression which is based on LR. The difference is that instead of building one LR model, it will build **(C-1)** models and predict according to the best score (C is the number of classes). We choose LR because it suits frequency-based features, and we expect it to perform well using CountVectorizer and TF-IDF as the encoding methods. This is also the reason that it is ideal for this text classification task.

Critically analyses the format of the features and critically weighs it against each classifier's strengths and weaknesses

2.1.2 Support Vector Machines (SVM)

SVM finds a hyperplane to separate two classes. In our task, the built feature spaces using various encoding methods are complicated, but SVMs can be applied to non-linearly-separable data without considering the dimensions. In Wang, M., Zhang, H. and Ding, R.,'s research in 2011, they also proved SVMs to work well on text categorization. Hence, we consider SVM as a good approach for our task.

2.1.3 Naive Bayes (NB)

NB classifiers classify an instance into one of the possible classes according to the joint probability. It is based on the assumption that all attributes are *independent*, conditional on the class. In our task, the review text we want to learn and predict may contain the words which never appear in the trained text features.

NB can still make a valid prediction by ignoring the new words and calculate the probability for each class. Inspired by the research conducted by Mowafy, M., Rezk, A., and El-bakry, H. M. in 2018, Multinomial NB classifier works very well with TF-IDF encoding method. We want to test this point in our task as well.

2.1.4 Stacking

The ensemble stacking learner we have applied includes three base classifiers: Logistic Regression, AdaBoost and Decision Tree. The reason for choosing them is that the mechanisms of learning behind these three classifiers are different. Hence, we expect them to have varying performance with text classification. After combining the base classifiers, we trained the meta classifier, Logistic Regression, over the outputs of them to achieve a less biased result. Since a nested cross validation is applied, we expect Stacking to outperform simple LR because the bias has been reduced.

2.2 Encoding Methods

2.2.1 CountVectorizer

This is also known as the Bag-of-Words model. The model will convert documents to vectors which represent word counts. Hence, after transformation, the row of the matrix represents the count of each word in the general word dictionary inside that specific review. However, this model ignores the relative position information of the words and hence the order and context are not used.

2.2.2 Doc2Vec

Doc2Vec is based on word2vec, which maps words to a high-dimensional vector space so that words which appear in similar contexts will be close together in the space. Unlike CountVectorizer, it considers the position of words under the context in the whole text document. Currently, it is hard for us to explain the real meaning of numbers inside the generated features. Hence, we had made an assumption of Gaussian distribution for each feature to carry out some experiment.

2.2.3 TF-IDF

TF-IDF is known as term frequency–inverse document frequency vectorizer. Its output has a similar structure as the CountVectorizer. However, TD-IDF transforms the count vector to a normalized form by reducing the weight of terms that occur frequently and increasing the weight of terms that occur rarely. Hence, we treat TF-IDF as an improved version of CountVectorizer.

2.3 Feature Selection

2.3.1 Meta Features

Before going to the text feature engineering, we have done some correlation calculation on the meta features to the class feature ‘rating’. Firstly, the IDs should be totally useless for prediction as they are independent of the case. The three vote features labelled 'vote_funny', 'vote_cool' and 'vote_useful' have around -0.048, 0.051 and -0.054 correlation with ‘rating’ respectively (using Pearson). This suggests that the meta features do not contribute to the prediction, and we can treat the text reviews as the main features for our systems.

Properly justifies feature selection with correlations and time efficiency

2.3.2 Filtering Methods

For the feature selection, we choose to use chi-square (X2) instead of mutual information (MI) since we found out that SelectKbest() runs much faster using X2 than MI, while the performances of both methods are almost equal. As the parameter k (number of the best features) increases, the time efficiency of MI becomes worse. Hence, X2 is preferred for feature selection.

2.4 Hyperparameters Tuning

After deciding the classifiers, we have also involved consideration of hyperparameters. First thing we came up with is the kernel type of SVM classifiers. Since the relationship between features are not necessarily linear, we have tried all kernel types for SVM (Table 1).

SVM kernel type	Accuracy
LinearSVC	87.08%
SVC with linear kernel	87.06%
SVC with rbf kernel	86.72%
SVC with polynomial kernel	69.24%

Table 1. Hyper-parameter tuning for SVM with different kernels

Critically analyses model bias

Two SVC kernels (rbf and polynomial) both have the problem of overfitting and their performances are worse than LinearSVC and SVC with linear kernel. Hence, LinearSVC should be used as the final model.

Another hyperparameter we have considered is the regularization parameters for both LinearSVM and LR (Table 2).

	Best C	*Accuracy with Best C*
LinearSVM	1	87.08%
LR	10	86.85%

Table 2. Hyper-parameter tuning for Cs and the corresponding accuracies

For LinearSVC, the default value of C (=1.0) seems to be the best parameter, while C= 10 is the best regularization parameter for the LR model.

3. Evaluation

As mentioned in "Section 2 Methods", we have done a series of experiments to decide the most ideal encoding and feature engineering method for each classifier we have chosen. After this, we are able to conduct the evaluation on every part of these methods.

3.1 Encoding: Results and Error Analysis

We test the classification accuracy of every classifier with five vectorization methods: CountVectorizer, TD-IDF and Doc2Vec with 50, 100 and 200 features. The sparse matrix will be split into "train" and "test". A holdout strategy with 33% test set and 67% train set has been used here for evaluation (see Table 3).

Inductive Learning Hypothesis satisfied though not explicitly stated

	CV	*TF-IDF*	*D2V-50*	*D2V-100*	*D2V-200*
LR	84.27%	85.64%	81.79%	82.76%	83.28%
NB	83.61%	69.24%	72.05%	67.18%	61.50%
Stack	84.14%	86.45%	75.68%	75.98%	75.76%
SVM	81.59%	86.63%	81.58%	82.68%	83.19%

Table 3- Classification accuracy with vectorization methods.

LR classifier averagely shows a very good performance based on all of the chosen encoding methods. We expect LR to work well with CV and TF-IDF as mentioned in 2.1.1 and the results meet the expectation. Another observation is that the accuracy is improved with the increasing number of features generated by Doc2Vec. However, TF-IDF still outperforms the D2Vs by more than 2% accuracy.

For NB classifiers, we used Multinomial NB (MNB) for CV and TF-IDF, and Gaussian NB (GNB) for Doc2Vec since MNB fails to process negative values generated by the Doc2Vec. We observe that MNB works as well as LR using CV, but GNB doesn't perform well on Doc2Vec, and the accuracy goes down with increasing number of features generated by Doc2Vec. This implies that numbers inside the Doc2Vec features do not really follow a Gaussian distribution, and hence the assumption in 2.2.2 is invalid.

Relates to previous assumptions very well and has concise justification, validating and invalidating own assumption/hypothesis

Surprisingly, MNB does not perform well with TF-IDF. The point in 2.1.3 is hence invalid for our task. Our explanation is that TF-IDF's normalization is not prefered by MNB since it undervalues the higher frequency words. By doing so, the likelihood of some words changes and more instances are classified wrongly.

The Stacking classifier, as expected, gives the best accuracy score of 86.63% with TF-IDF and performs equally well as LR when CV is applied. Accuracies of around 75.7% are observed for all three Doc2Vec methods. This suggests a limited influence

on Stacking's performance by feature numbers of the Doc2Vec.

SVM performs as good as LR generally. A notable observation is: unlike LR, SVM seems to prefer TF-IDF much more than CountVectorizer. This suggests that by increasing the weight of rare words and decreasing the weight of common words, SVM works better at separating the classes.

After examining the performances of every vectorization-classifier combination, we decided to apply TF-IDF on LR, Stacking, SVM and CountVectorizer on MNB.

3.2 Feature Selection: Results and Error Analysis

By utilising CV as the encoding method and conducting SelectKbest() using chi-square, with different k values (indicating selecting different numbers of most correlated features to the class feature 'rating'), we observed a variation of accuracy with different k values (Hyperparameters) (Figure 1).

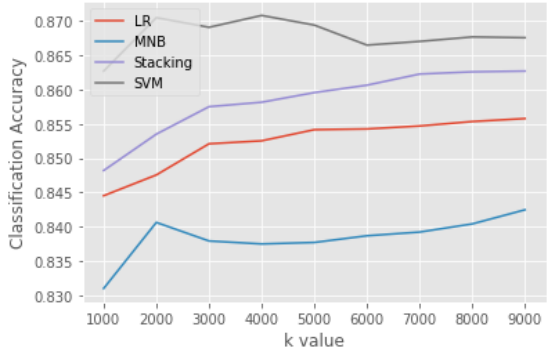

Figure 1- Accuracy vs. k value.

LR and Stacking have a very similar variation trend. This is due to the fact that LR is also one of the level-0 classifiers under the stacking model. Instead of simple LR, the learning process of Stacking also involves AdaBoost and DecisionTree models, which produce a variety of biases and variances, and this will let Stacking outperform LR with the same feature engineering.

MNB, on the other hand, shows a small range of variation of around 0.5% accuracy. Its performance is generally worse than LR and Stacking in terms of accuracy.

Surprisingly, SVM works much better than Stacking despite its simplicity. The accuracy can go to above 87% with barely 4000 features. We think this is due to the stacking model we have applied does not include SVM as one of the base models. Theoretically, by doing so, we are supposed to achieve a higher accuracy with stacking.

The final decision is: we take k= 4000 for SVM and k= 8000 for the rest of models.

3.3 Evaluation of the full models

3.3.1 K-fold Cross Validation

We have already done some basic evaluation on the incomplete models by conducting holdout strategy. However, we have no idea whether the accuracy we have gotten is valid or not since the models can possibly overfit or underfit the training set. Hence, K-fold cross validation (k=5) has been used as a further evaluation technique here (Table 4).

	HOLDOUT	K FOLD CROSS VALIDATION
LR	85.64%	85.66%
NB	83.61%	83.75%
SVM	86.63%	86.27%

Table 4- Classification accuracy with holdout and K-fold cross validation strategies

After utilising every part of the datasets as training and testing sets, we found out there is not any huge variation in accuracies between the two strategies. The previous evaluation is therefore valid. This can also be supported by the results we have gotten from Kaggle. The scores we received for every model we have applied are very close to what we got from the evaluation part.

3.3.2 Confusion Matrix

To explore what the four models have done on each class of the dataset, we have implemented the confusion matrix (Figure 2). According to the distribution of classes:

label 0= rating 1

label 1= rating 3

label 3= rating 5

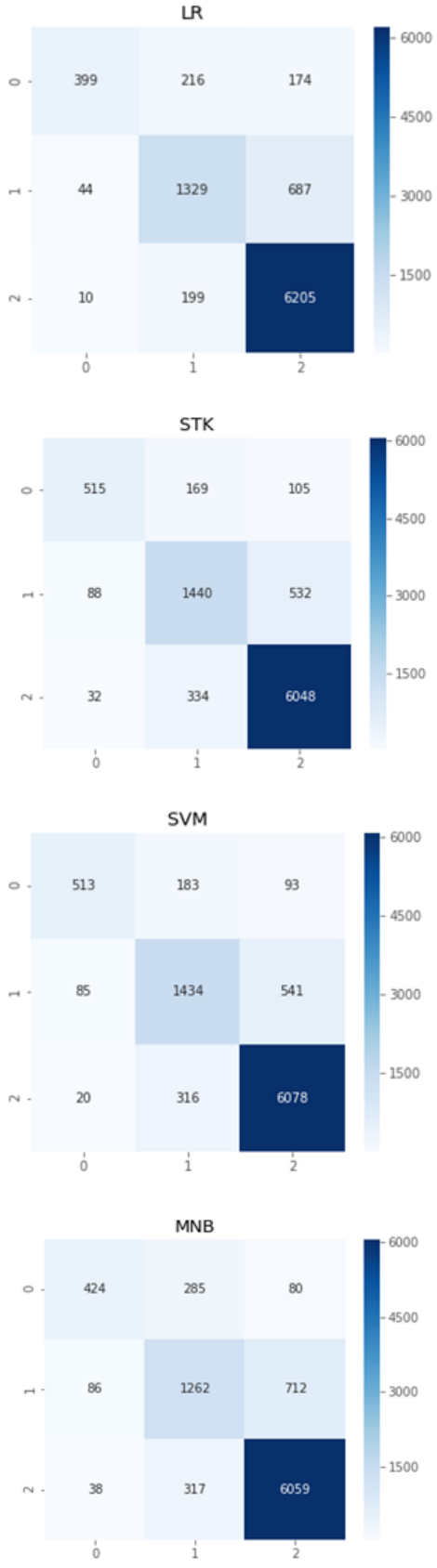

Figure 2- Confusion Matrix for the four models.

A large percentage of reviews with rating '1' are wrongly classified as '3' or '5' by every model. At the same time, around 50% of rating '3' reviews are predicted as '5' using LR and MNB, and this proportion goes to around 30% using STK and SVM. Lastly, most of the reviews with rating '5' are classified correctly by all models.

The different ratios of right and wrong prediction are likely caused by the imbalanced training dataset itself. Out of 28068 instances, 2336 are '1', 6444 are '3' and '19288' are '5'. The distribution of the prediction output by our models will tend to follow the distribution of original 'rating'. Therefore, more instances will be classified as '5' and less will go to '1' and '3'. This can be considered as a systematic error. The proof is the fact that in Kaggle competition, we always got a close but slightly lower accuracy score than what we have gotten using the validation set.

4. Conclusions

In summary, among the four machine learning systems we experimented for this sentiment analysis task, SVM with Frequent Count vectorization TF-IDF stands out, ending up with an accuracy of 87.08% under 33% hold-out and 4000 selected features. In particular, TF-IDF also improves the accuracy of Logistic Regression and stacker. In contrast, although MNB works better with CountVectorizer, the accuracy Naive Bayes achieves is still lower than most of its counterpartners.

A high ranked result in Kaggle competition proves that the decision making process we have conducted in the Method section is effective in improving the machine learning systems for this specific text classification task.

Future improvement for this task should adjust the level 0 models of the ensemble stacking classifier, such as adding a SVM classifier due to its well performance in this problem. In addition, since we found out the great performance of Frequency Count vectorizer, we can improve the systems by involving

n-gram tokenization. So the words combinations like ‘not bad’ are added into the training process and can hopefully improve the systems.

References

Mukherjee, A., Venkataraman, V., Liu, B. and Glance, N., 2013, June. What yelp fake review filter might be doing?. In *Seventh international AAAI conference on weblogs and social media*.

Rayana, S. and Akoglu, L., 2015, August. Collective opinion spam detection: Bridging review networks and metadata. In *Proceedings of the 21th acm sigkdd international conference on knowledge discovery and data mining* (pp. 985-994).

Mowafy, M., Rezk, A. and El-bakry, H.M., 2018. An Efficient Classification Model for Unstructured Text Document. *Am J Compt Sci Inform Technol*, *6*(1), p.16.

Wang, M., Zhang, H. and Ding, R., 2011. Research of text categorization based on SVM. In *Proceedings of the 2011, International Conference on Informatics, Cybernetics, and Computer Engineering (ICCE2011) November 19–20, 2011, Melbourne, Australia* (pp. 69-77). Springer, Berlin, Heidelberg.