Sentiment Analysis on Movie Reviews

Using Support Vector Machines with Linear Kernel

Benny Tieu, Robert Yousif

April 2016

Abstract

Lorem ipsum...

Contents

1 Introduction

Sentiment analysis is the study in machine learning to analyze people's opinions on certain topics or objects. People have increasingly voiced their opinions on the web, for instance on social media discussing trends or writing reviews about the latest movies they have seen in theaters. Gathering such data is important to organisations and companies to better understand what satisfies or dissatisfies customers, in order to further improve their products. Using peoples text and analyze its sentiment is an application of NLP (Natural Language Processing), the study in computational linguistics [11]. Analyzing sentiments faces many challenges and is a complex matter, since natural language is complex in itself. Words that are generally considered positive, such as "happy", "party" or "vacation" may have a negative meaning used in a context. For instance, the words "scary" or "creepy" have a generally negative meaning in our daily lives, but have possibly a positive meaning describing a horror movie.

This paper will cover each step involving implementing a model that will predict a sentiment value, given a phrase in a certain context. The method is referred to as supervised classification or supervised learning. The concept of supervised learning is given a labelled data set, called training set, identify the features and learn a model that will predict labels given training set as input. The flow chart below illustrates each step of supervised learning. In the case of giving a sentiment analysis on movie reviews, the features are the preprocessed movie reviews and the labels are the sentiments.

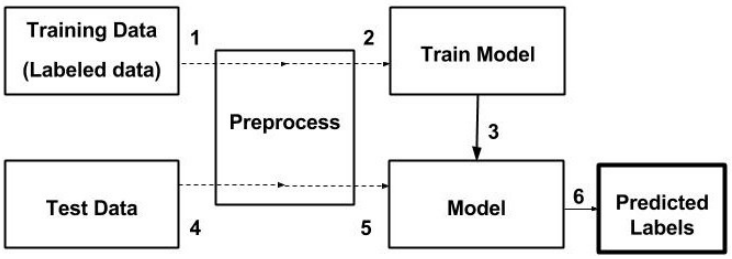

Figure 1: Flow chart of training a model and use it to predict labels.

There are several methods to classify text into sentiments. There are lexicon-based approach or using supervised learning. Most popular methods includes Decision Trees, Naive Bayes, Neural Network and Support Vector Machines (SVM). In this project, we will be focusing on SVM using a linear kernel and comparing two different strategies, One-VS-All and One-VS-One.

The training- and test sets are provided by Kaggle as part of the competition named "Sentiment Analysis on Movie Reviews" that ended 28 Feb 2015[1]. The evaluation of the predicted labels are evaluated through the competition website.

2 Related works

SVM was first introduced by Burges at late 1990's and have since then been proven to be a working method for sentiment analysis in different domains [5]. Wang and Manning compares different methods for text classification and sentiment analysis. One comparison made is between Naive Bayes and SVM. They stated that the performance varies significantly depending on several factors, i.e. alternations of the algorithms, the features selected and the dataset used. It is difficult to know in advance which classifier that performs the best for a specific domain [6]. In this project we are provided by snippets of movie reviews in a relative large dataset. Moilanen and Pulman suggest that statistical methods works well for datasets with hundreds of words in each example, while they handle snippets of movie reviews poorly [10]. However, Wang and Mannings research concluded that SVM is better at full-length reviews.

As for feature selection, a general increase of performance is to only consider letters, not using stop words [14] and using lemmatization[9].

Multiple research papers have been comparing One-VS-All and One-VS-One strategy for classification and concludes that neither strategy performs better generally and depends on the domain [8][14]. Milgram, Cheriet and Sabourin concludes that the performance depends on the number of training samples, number of classes and application constraints. They noticed however that One-VS-All performed better on smaller datasets than One-VS-One [8].

3 Motivation

This project's goal is is to satisfy the course objective by using "real-world data sets to deeply understand how apply machine learning techniques to solve real world application". This is achieved by implementing a model that is able to predict *accurate* sentiments of given phrases from movie reviews. The sentiment labels are the following [1]:

0 = Negative

1 = Somewhat negative

2 = Neutral

3 = Somewhat positive

4 = Positive

The goal is also implement an accurate model for the test data, in order to get the highest possible score in the Kaggle competition. Analyzing other evaluation strategies are not in the scope for this paper.

4 Background

4.1 Pre-processing

Pre-processing is a phase for feature extraction. It is a method done to *transform* or *filter* noise in the text. Removing or transforming this noise can result in a more accurate model and also minimize the amount of data being processed when predicting the test data. There are several methods in order to filter and clean the text. The methods involved are dependent on how the training data is formatted, and also for what purpose the model is used for. There are some phrases that are more obvious to remove than others. Examples of obvious noise are HTML-tags or advertisements, these may occur when retrieving online data and can be filtered out directly. There are however some less certain pre-processing methods that must be investigated. These methods are considered under the presumption that the transformation or filtering gives a more accurate sentiment value prediction by the model [9]. The following methods are considered for this paper:

- Only considering letter symbols
- Only consider lowercase symbols
- Remove numbers or use a placeholder (e.g. use $\{num\}$ for every number)
- Remove punctuations (i.e. symbols like , . ? ! ;)
- Remove stop words (words like: a, an, the, me, you)
- Stemming and Lemmatization

Stemming and lemmatization are methods to reduce words and transform it to a common base form. Stemming refers to a method that uses a heuristic process that removes end of words, e.g. *eating*, *eating*, *eating* will result in the word *eat*. In some cases, the word that is reduced does not have to be a correct lexical word. For instance, the words *octopus* and *octopie* may be reduced to *octop*. Stemming is less sophisticated than lemmazation in a sense. Lemmazation refers to transforming the word into its basic, dictionary form, a lemma. It uses an algorithmic process that considers vocabulary and natural language analysis.

4.1.1 Bag of words

Bag of words is a data structure to represent the features in. Consider that each phrase has been transformed and filtered. The phrase can further be processed and divided into n-grams that is put in a set. Each element in this set is associated with a value. In sentiment analysis, this value is commonly the frequency count of the n-gram in the document [9].

4.1.2 Term Frequency–Inverse Document Frequency

Term Frequency–Inverse Document Frequency, short TF-IDF is a method to determining the relative frequency of words in a specific document compared

to the inverse proportion of that word over the document [9]. Informally this means weighing down terms that occurs too frequent and weighing up terms that occurs less frequent proportionally. As formally defined in Formula 1, the TF-IDF weight of a term is the product of its TF weight and its IDF weight. TF is the frequency a term in a document.

$$TF\text{–}IDF = TF \times IDF$$

Formula 1: TF-IDF

IDF estimates the rarity of a term in a document and is defined in Formula 2. w_i is i:th term, $|D|$ is the cardinality of documents, $DF(w_i)$ the count of documents that contain the term w_i. If a term is not in the corpus, $DF(w_i)$ will result in zero-division. It is therefore common to add 1 in the denominator or ignore these cases.

$$IDF(w_i) = log(\frac{|D|}{DF(w_i)})$$

Formula 2: IDF

In larger corpuses the TF can be biased to a larger document, meaning that a term can have a higher count regardless of the importance of the term in the document. This is solved by normalize the TF, see Formula 3. $n_{i,j}$ is the occurrences of the term t_i in document d_j.

$$TF\text{–}Normalized_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

Formula 3: Normalized TF

4.2 Super Vector Machine

Super Vector Machine, SVM, is a supervised learning algorithm that classifies objects into an n-dimensional space. The classes in the space will be separated by a n-dimensional hyperplane. The purpose of SVM is to create a model for mapping of input-output data, SVM is proven to be a very powerful classifier for sentiment analysis. [12]

The central task of SVM is to define the hyperplanes in space for the best separation of the different classes. Linear SVM will separate the classes linearly with hyperplanes, however data instances can be mapped non linearly to an inner product space, the classes can then be linearly separated with hyperplanes. [7]

Each phrase of the training data is plotted as a point in the n-dimensional space, where n is the number of features, every phrase with their own features as a representation of a particular coordinate. Five classes would result in a 5

dimensional space where each phrase/word has 5 specific features representing a coordinate mapping that particular phrase/word in the space. To separate the classes classification is preformed setting the hyperplanes.

SVM are essentially binary classifiers. However this technique has been adapted to work on multiclass problems. There are different multiclass SVM approaches, the two most common ones are One-VS-One and One-VS-All. The One-VS-All approach contains of N-binary classifiers, meaning one classifieras have positive and negative examples. For the i:th classifier, let the positive examples be all the data points in class i, and negative examples be all the points not in class i. f_i will then be the i:th classifier. Classifying using One-VS-All approach will be defined as [9]:

$$f(x) = arg\_max_i(f_i(x))$$

Formula 4: Classifying using One-VS-All

As illustrated in Figure 2 the classifiers will be evaluated as *Class 1* or not *Class 1*, *Class 2* or not *Class 2* etc.

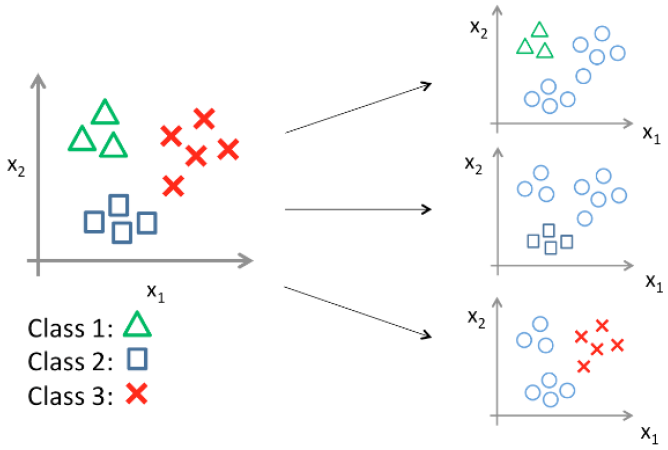

Figure 2: One-VS-All example with three classes

The One-VS-One approach, also called the all-pairs approach, will construct classifiers for each pair of classes. This gives the total number of $N(N-1)/2$ classifiers. Let $f_{i,j}$ be the classifier where i represents positive- and j negative examples. Classifying using One-VS-All approach will be defined as [9]:

$$f(x) = arg\_max_i(\sum_j f_{i,j}(x))$$

Formula 5: Classifying using One-VS-One

As illustrated in Figure 3, each binary classifier will be evaluated as *Class 1* or *Class 2*, *Class 1* or *Class 3* etc. This results in a heavier computational work due to the number of classifiers. To determine what class the testing point has every classifier will vote for their winning class, the class with the most votes will then be assigned to the test point. [2]

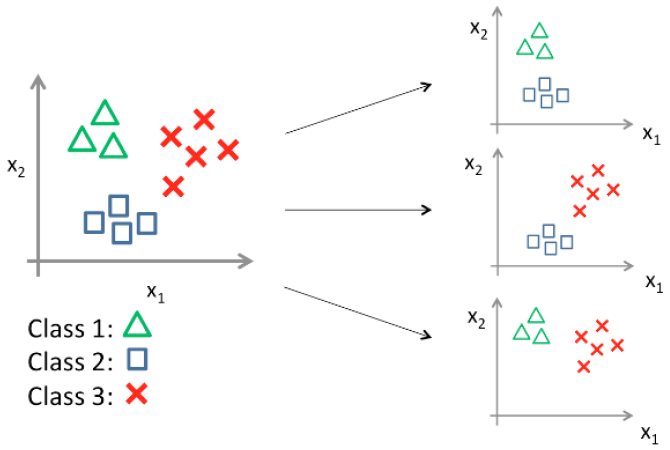

Figure 3: One-VS-One example with three classes

4.3 The C-parameter

C can be described as the soft margin constant, a kernel parameter that influences the hyperplanes between the classes in respect of the penalty assigned from the margin errors. A large C would result in a hyperplane letting the points with small margin have a heavy weight on the orientation of the hyperplane as illustrated on the left panel of figure 4. With a small value of C the points with small margin has less effect on the orientation of the hyperplane as seen in the right panel on the same figure, resulting in an overall better margin for the majority of the points. To find the best value of C is therefore important due to its effect on the seperation of classes. Moreover to large C value would result in overfitting and to small value of C would result in underfitting. [3]

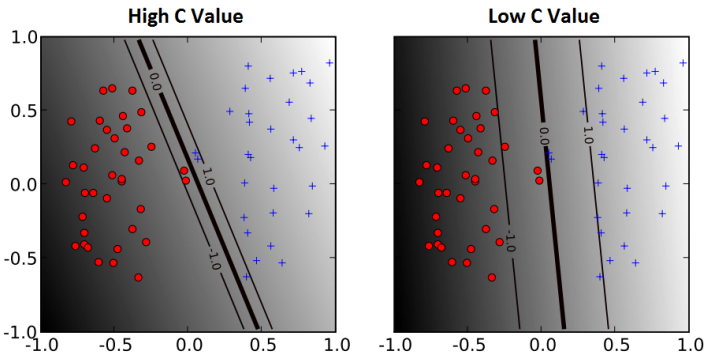

Figure 4: Soft margin constant C, the thick line is the hyperplane and the lighter lines are the margins [3]

5 Methodology

The implementation was written in Python using various functions in the *scikit-learn* [13] and NLTK[4] library. Each phase, from pre-processing to building the model is passed down in a `pipeline`, where the output of one function is the input of the following function.

5.1 Data Sets

There were two data sets provided for this project, one training set and one test set. The training set contained $156,060$ rows. Each row contained a *PhraseId*, a *SentenceId*, a *Phrase* and a *Sentiment* ranging from 0 to 4. *SentenceId* was omitted for this implementation. The test data set contained $66,293$ rows. Each row contained a *PhraseId*, a *SentenceId*, and a *Phrase*. The sets of data are collected from the Rotten Tomatoes movie review dataset [1].

5.2 Pre-processing

The bag of words was created by using the function `CountVectorizer`. This function have parameters to specify options to filter and transform phrases. The filtering and transform methods were presented in Section 4.1 and considered in this phase. Removing punctuations was done in a separate function (specified in the tokenizer parameter) by removing all phrases that contained punctuations. The set of punctuations is provided by `string.punctuation`. The `min_df` parameter removes terms that have a document frequency strictly lower than the given threshold. This is sometimes refereed to cut-off.

As for the lemmatization and stemming, both is considered. The lemmatiazation was achieved by calling the function `lemmatize` for each word. The function is part the package `nltk.stem.WordNetLemmatizer`.

It is also possible to choose the n-gram size for each element in the bag of words. `CountVectorizer` converts the words in a set where each element is the count frequency of the n-gram and the index corresponds the word. After getting bag of words set, the `TfidfTransformer` function was applied. The function was set to default to be normalized since the document in this case is very large as discussed in Section 4.1.2, TF-IDF without normalization was also considered.

5.3 Building and using the model

After running the training data through the functions `CountVectorizer` and `TfidfTransformer` the feature extraction is completed. The script proceeds to build the model. Two implementation of the model was considered for this paper, `LinearSVC` and `SVC` specified to have a linear kernel. The difference is that `LinearSVC` uses the One-VS-All strategy and `SVC` uses the One-VS-One strategy. The C-value was specified when the object was initialized.

Prediction was generating using the function `predict` in respective SVC object.

5.4 Removing Duplicates

There exists duplicates between the training data and the test data. The sentiments of these duplicates was directly transferred for respective phrase in the submission file. This method is not a practice in machine learning, but definitely contributes to the goal of get a higher score in the competition.

6 Result

6.1 Pre-processing

This section presents different settings of the pre-processing phase and the percentage of improvement by using each method. All other parameters are set to a default, fixed value when tested and the results presented are relative to this default settings. The default settings are `1-gram`, `C=1.0` and `min_df=1`.

Pre-processing method	Score improvement
Only consider letters	**+0.487 %**
Only consider lowercase	**+0.322 %**
Remove numbers	+0.002%
Remove punctuations	+0.293%
Placeholder for numbers	−0.005%
Remove stop words	±0.000%
Use stemming (PorterStemmer)	**+11.235 %**
Use lemmatization (WordNet)	+10.895%
TF-IDF	−7,534%
TF-IDF (Normalized)	**+0.967 %**

Table 6.1: Improvements of pre-processing settings. Marked in bold are the settings considered for the best result.

Observing the results of Table 6.1, we can see that the best combination is by **only consider letters and lowercase** and using **stemming**, all these marked in bold in the Table. Testing the values of `min_df` showed that 0.000040 gave the best results which is a +0.482% improvement.

Table 6.2 presents the score improvements of different N-gram settings. All of the results are relative to 1-gram.

N-gram range	Score improvement
1-1	±0.000%
1-2	+**0.539**%
1-3	+0.337 %
1-4	+0.144 %

Table 6.2: Results of different settings of N-gram. Best result marked in bold.

6.2 One-VS-All

Table 6.3 presents the results in the Kaggle competition using settings marked in bold in table 6.1 and 6.2. The functioned used was `LinearSVC` which uses One-VS-All strategy. Training the model and generating the predictions for the test data takes about 1 minute.

C-value	Score improvement
10	−4.144%
1.0	0.0%
0.9	+0.165%
0.8	+0.404%
0.7	+0.583%
0.6	+0.819%
0.5	+1.088%
0.4	+1.272%
0.3	+1.438%
0.2	+1.511%
0.17	+**1.661**%
0.1	+1.174%
0.01	−5.246%

Table 6.3: Results of various C-value for One-VS-All strategy. Best result marked in bold.

The best result is using `C=0.17` which results in a 0.65377 accuracy score in Kaggle. This results in placement 115 out of 861 submission, meaning top 13.36%

6.3 One-VS-One

Table 6.3 presents the results in the Kaggle competition using settings marked in bold in table 6.1 and 6.2. The functioned used was `SVC` specified to linear kernel which uses One-VS-One strategy. Training the model and generating the predictions for the test data takes about 1 hour.

C-value	Score improvement
10	−4.072%
1.0	0.0%
0.9	+0.285%
0.8	+0.453%
0.7	+0.722%
0.6	+0.935%
0.5	+1.134%
0.4	+1.281%
0.32	+**1.402**%
0.3	+1.366%
0.2	+1.024%
0.1	+0.217%
0.01	−8.266%

Table 6.4: Results of various C-value for One-VS-One strategy. Best result marked in bold.

The best result is using `C=0.32` which results in a 0.66266 accuracy score in Kaggle. This results in placement 30 out of 861 submission, meaning top 3.48%

7 Discussion

7.1 Evaluation Criticism

As mentioned in the motivation section, the goal of the project is to implement a predictive model that generates the highest possible score in the Kaggle competition. This should not be confused that the model is generally considered accurate. The question to consider is: How is the accuracy measured? The methodology focuses on a high accuracy relative to the test data. The settings that generated the highest score in Kaggle may generate another accuracy score by using for instance the 10-fold cross validation, which is not in the scope for this project. Also using another test data set in another domain may generate a different score. The model implemented for this project can be considered biased to this specific domain.

The phrases and sentiments in the training data was provided by Rotten Tomatoes movie review dataset as for the test data. This is reliable source of data, since the website is used for reviewing and expressing sentiments on movies by the general public. However, different people may express sentiments

differently, with this in mind it is therefore impossible to create a perfect predictive model. Judging by the Kaggle score board, the top accuracy is around 67-70%. This means that people generally agrees on text 67-70% of the time.

7.2 Analysis of pre-processing

Some of the tested pre-processing methods resulted in a positive outcome and was used for the best result. Namely, *only consider letter symbols, use only lowercase symbols*. By this it can be concluded that all symbols, that is not in the alphabet, have a sentiment value that is not relevant for the test data.

It was unclear initially if removing punctuations would result in a more accurate model. This was considering signs like "???", "!!!" and "..." may possibly have a sentiment value that is not just neutral. It showed however in the results that removing punctuations resulted in a higher score.

A surprising result was that using stemming gave a better result than lemmatization. This is since stemming only uses a crude function to reduce the words. Lemmatization is considers natural language analysis and is more sophisticated in a sense. As mentioned in the *Evaluation Criticisim* section, the model is biased to this domain, meaning that `WordNet` is possibly not best suited for this domain compared to `PorterStemmer`. Another surprising result was that removing stop words did not affect the scoring at all. This may have been an effect of using normalized TF-IDF, which weighted down the most frequent words, in this case stop words. The 0.00004% least frequent words were also removed.

7.3 One-VS-All and One-VS-One

One-VS-All strategy took significant less computation time to train the model and get a prediction for the whole test set. However it generated a less accurate score than using the One-VS-One strategy. One-VS-One trains a separate classifier for each different pair of labels, meaning that it evaluates more classifiers than One-VS-All. The time complexity for One-VS-All is $O(N)$ and One-VS-One is $O(N^2)$. The latter can be problematic in a system that needs to train a model dynamically, since it takes time to re-train the model. As mentioned in the related works section, some researchers concluded that the difference of the different strategies are mostly the computational time. The different strategies may generate different accurate prediction on different data sets. One-VS-One is the better strategy for this domain. It can not be concluded that one strategy is better than the other generally.

7.4 The C-Parameter

Regulating the C-parameter showed that 0.17 was the better setting for One-VS-All strategy, while 0.32 was the better for One-VS-One. Too large or too low of C-Value generated a decreased score for both strategies, which is a result of

overfitting and underfitting respectively. This was an expected result mentioned in *Section 4.3*.

8 Conclusion

SVM with linear kernel using either One-VS-All or One-VS-One generates an accurate prediction for the goal mentioned in *Motivation* section. The prediction is accurate in terms of the domain for the Kaggle competition *"Sentiment Analysis on Movie Reviews"* and Rotten Tomatoes. It is possible that the model generates a different accuracy score using another evaluation strategy.

Pre-processing the data was done to extract the features. Only considering letter symbols and lowercase generated a +0.487% and +0.322% score improvement respectively. Using Stemming (PorterStemmer) generated a +11.235% score improvement compared to lemmatization (WordNet) that had +10.895%. Normalized TF-IDF generated +0.967% improvment, while TF-IDF without normalization gave a decreased score. Removing stop words does not change the score, when the pre-processing methods mentioned above are considered. Removing the 0.00004% of the least frequent words resulted in a +0.482% better score.

The best result of C-value was 0.17 for the One-VS-All strategy, which generated a 0.65377 score in Kaggle. The best result of C-value was 0.32 for the One-VS-One strategy, which generated a 0.66266 score in Kaggle. It can not be concluded that One-VS-One is the better strategy in the general sense, however it was the better for this specific domain.

8.1 Best Results

To summarize, the settings that generated the best results were the following:

- Only consider letter symbols (a-z)
- Only consider lowercase symbols
- Stemming
- Normalized TF-IDF
- `min_df` = 0.00004
- C = 0.32
- One-VS-One strategy

All this generated an accuracy in Kaggle of **0.66266 score units**, which corresponds a **30:th placement** in the leader board, out of 861 submissions. This means the **top 3.48 %** best results of all the submissions.

9 Distribution of work

Both members of the project worked with the code together using pair programming, both being programmer and observer. Research of the methods was done generally separately, Tieu doing pre-processing and Yousif doing SVM.

References

[1] Sentiment Analysis on Movie Reviews. https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews, 2015. [Online; accessed 2016-4-17].

[2] Gidudu Anthony, Hulley Gregg, and Marwala Tshilidzi. Image classification using svms: one-against-one vs one-against-all. *arXiv preprint arXiv:0711.2914*, 2007.

[3] Asa Ben-Hur and Jason Weston. A user's guide to support vector machines. *Data mining techniques for the life sciences*, pages 223–239, 2010.

[4] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python.* " O'Reilly Media, Inc.", 2009.

[5] Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[6] Kai-Bo Duan and S Sathiya Keerthi. Which is the best multiclass svm method? an empirical study. In *Multiple classifier systems*, pages 278–285. Springer, 2005.

[7] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal*, 5(4):1093–1113, 2014.

[8] Jonathan Milgram, Mohamed Cheriet, and Robert Sabourin. "one against one" or "one against all": Which one is better for handwriting recognition with svms? In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.

[9] IC Mogotsi. Christopher d. manning, prabhakar raghavan, and hinrich schütze: Introduction to information retrieval. *Information Retrieval*, 13(2), 2010.

[10] Karo Moilanen and Stephen Pulman. Sentiment composition. In *Proceedings of RANLP*, volume 7, pages 378–382, 2007.

[11] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002.

[12] Ms Gaurangi Patil, Ms Varsha Galande, Mr Vedant Kekan, and Ms Kalpana Dange. Sentiment analysis using support vector machine. *Indian, VIIT Engineering College, Pune*, 2014.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[14] Sida Wang and Christopher D Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 90–94. Association for Computational Linguistics, 2012.