Application of Machine Learning Techniques to Sentiment Analysis

CS4824 Final Project

Brian Lee

Abstract

Sentiment analysis is been an increasingly useful application of machine learning techniques. With the growing amount of textual data circulating on the internet and across devices, there is a need to process and classify it all. Also known as text analysis or opinion mining, sentiment analysis is the task of feeding textual data into machine learning algorithms to construct models that attempt to classify new texts into "positive" and "negative" sentiments.

The aim of this paper is to demonstrate how known machine learning techniques are applied to non-numeric data. This includes the process of data preparation, feature extraction, and how model construction using the support vector machine algorithm. The results and findings from various iterations of the support vector machine algorithm will then be presented, along with observations of interest and conclusions.

1. Introduction

The growth of the Internet has granted access to remarkable capabilities to ordinary people. One such capability is sharing sentiments and opinions on various platforms, such as sites dedicated to user-written reviews, blogs, and most notably, social media. Understanding public opinion, now easily expressible as text, can be incredibly valuable data. They can alter decision-making by giving targets for improvement and catering. Due to the subjective nature of sentiments, these texts must be read, analyzed, and understood [1]. One can begin to see the problem that exists when there are many texts to examine or the language used in them is rich and complex. Therefore, it would be beneficial to automate this process.

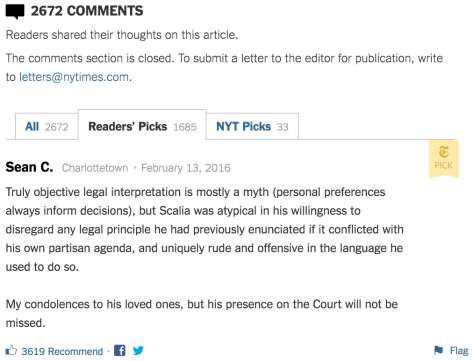

Figure 1. An example of user-written text containing a sentiment [5]

This is an incredibly powerful application of machine learning. By preprocessing textual data and converting it to numerical data by a process known as encoding [2], we can apply machine learning algorithms in an attempt to a classify examples as positive or negative. The autonomy of this process is what makes it special. This might be beneficial to businesses that depend on customer input, or government gaining public sentiment on a certain issue. The challenges of this process today are numerous. Texts that use sarcasm might be misclassified, while negations or indifferent opinions complicate the classification process. In addition, there are only rudimentary ways to calculate the polarity, or strength of the classified sentiment.

2. Data

The dataset being used is scraped from the "User Reviews" section on the IMDB website pages of various movies. This data was then aggregated and labeled posted on Kaggle by user *@yasserh* [3]. The dataset exists as a `.csv` file containing 40,000 examples of reviews, each labeled with 0 for a negative sentiment,

and 1 for a positive sentiment. Given the constraints of the program environment, only half of these examples will be considered to be the dataset.

3. Approaches

3.1. Data Preprocessing

On its own, the machine, or computer, cannot understand the textual data. However, the inherent challenge that exists with text data, at least in the English language, is that there are many forms of words and even more ways of forming sentences that are full of many tones, styles, and levels of detail. This is also irregardless of elements such as punctuation and spelling errors. Therefore, a necessary and crucial step is text preprocessing. Text preprocessing is the method by which data is cleaned and normalized in order to have the strongest encoding [4]. Among the preprocessing steps that were taken include:

- Conversion of text to lowercase
- Cleaning of text of single characters, special characters, numbers, and punctuation.
- Stopword removal
- Tokenization
- Part-of-speech tagging
- Lemmatization (normalization)

One component worthy of notice is stopword removal. In natural language processing, stopwords are words that lack much contribution to overall text sentiment. Many of these stopwords are subject words, such as ”I,” ”he,” ”she,”, and ”they,” as well as many others such as ”how,” ”are,” and ”doing.” Removing these words allows us to analyze more important and sentiment-revealing words in the text. More than previously thought, a lot of words in the English language are considered to be stopwords, even though intuitively, one might think that subject identifiers such as ”I” carry a more intense sentence than if ”they” was used. Another important component to mention is lemmatization. Lemmatization can be described as getting the ”base word”, or ”dictionary form” of words which might, in the text, be in a different form [4]. For example, ”changing”, a verb, lemmatized would be ”change,” a noun.

We can observe the difference between raw text data and preprocessed text data. It is now free of much of the fluff and unnecessary characters.

```
>> When this first came out, my dad brought it home- we were amazed
by it- It was so different from anything we had seen before. I was
looking for a specific movie last night, and I found "The Mind's Eye"
again. The box is falling apart, and I am surprised that the tape
still works! Although it is not 'Finding Nemo' quality graphics,
it is still very good. They should sell this again- it is a landmark
for computer animation imagery. Highly recommended!<br /><br />
This is what it is: <br /><br />"The Mind's Eye"is a spectacular odyssey
through time.

>>  first come dad bring home amaze different anything see
look specific movie last night find mind eye box fall apart surprise
tape still work although find nemo quality graphic still good
sell landmark computer animation imagery highly recommend
mind eye spectacular odyssey time
```

Figure 2. Raw text vs. preprocessed text [3].

3.2. Text Featurization

Now that we have obtained preprocessed text containing only the most important words, the next step was to find a way to featurize our text so that a model can be trained, constructed, and tested. This is a process known as text encoding [2], where we wish to encode individual words as various numerical values. This is also commonly known as word vectorization or text vectorization. There are many means to this encoding process.

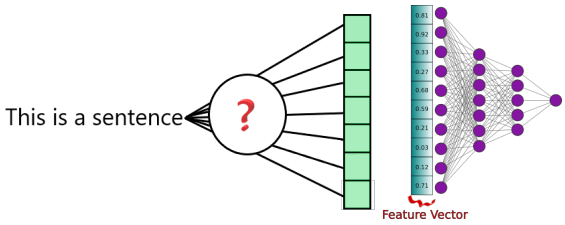

Figure 3. An illustration of how every word in a sentence can be encoded with numerical values [2].

However, I decided to use the term frequency-inverse document frequency (TF-IDF) model of word vectorization to give values to my features. There are other models of feature extraction from text data, such as the Bag of Words (BoW) model and the Word2Vec model [2]. However, the TF-IDF model of feature extraction is the most mathematically intuitive way to assign meaningful values to the text.

The TF-IDF algorithm aims to assign weights to each words in a body of text by its semantic importance [2,5]. By its name, "TF" stands for "term frequency," and "IDF" stands for "inverse document frequency [2,5]. Term frequency defined as the number of times a word appears in a text divided by the total number of words in that text, or the proportion of total words in that text, in other words [5]. Inverse document frequency is defined as the natural logarithm of the ratio of the total number of texts being examined plus one to the number of texts in which the term appears plus one [5]. Mathematically:

$$TF(term) = \frac{k}{N}$$

$$IDF(term) = ln\left(\frac{T+1}{\tilde{T}+1}\right)$$

where k denotes the number of times a term appears in a text, N is the total number of words in that text, and T denotes the total number of text documents being examined, while $\tilde{T}$ denotes the number of text documents in which the term appears. This algorithm is then applied to the data. It is applied row-wise to every text document example and column-wise to literally every word that will sequentially appear in the text. This means that the data's set of features is the set of unique words that exist in the text. The important result is that there is now a numerical value attached to a feature for every data point, with each unique word in the processed text being the features.

Note that without setting a limit on the number of features, the maximum number is equal to the number of unique words that appear across all of the text examples. In the movie reviews dataset this maximum is 80,506 features, far too many for any one analysis while far too unhelpful. I will set the maximum number of features to 100 because this is a reasonable number of words that might appear in one text example.

Due to the random access memory limitations of the problem environment, Google Colab, the program would crash every time the TF-IDF algorithm would attempt to run, despite cuts to the total number of examples [4]. This crashing was not unique to TF-IDF, but also occurred in model-fitting as well. So although the TF-IDF algorithm is implemented, we have opted for a more stable and efficient function from sklearn's 'TfidfVectorizer' module. Observe the distribution of the encoded text data:

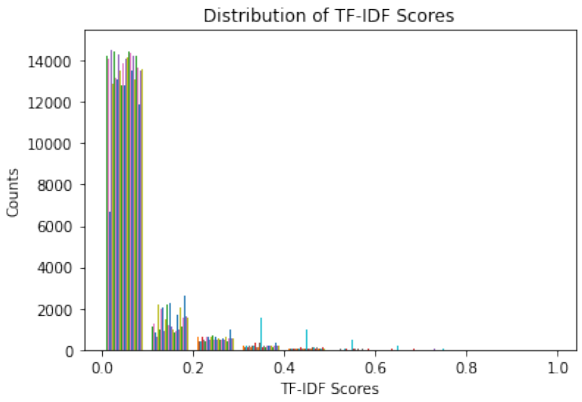

Figure 4. Distribution of feature values by TF-IDF calculation.

3.3. Machine Learning Algorithm: Support Vector Machine

The support vector machine algorithm, or SVM, is a supervised machine learning algorithm with the intention to classify data of binary classes [6]. To understand the intuition of SVM, imagine binary data, in the case that there are two features.

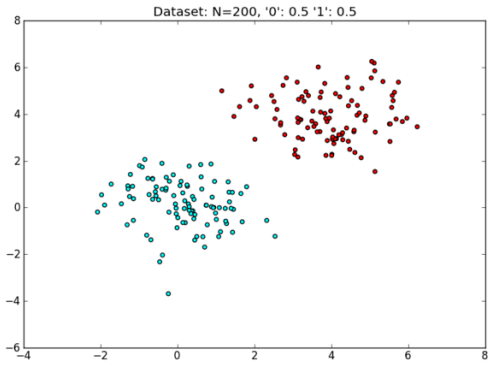

Figure 5. Binary-classified data [7].

The question then becomes as to what hyperplane is best suited to separate the data, including data which may have more than just two features.

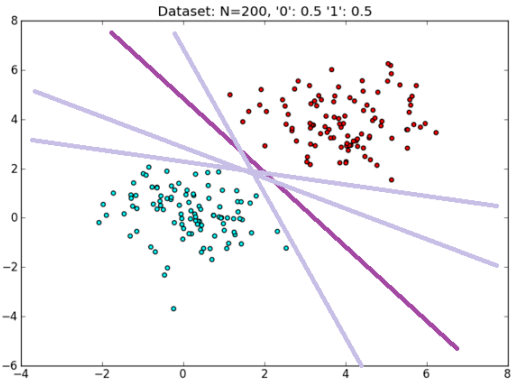

Figure 6. Binary-classified data, separated by various hyperplanes [7].

The answer is the hyperplane which maximizes the margin, γ, a distance between the hyperplane and the nearest positive and negative examples at the same time, with $\mathbf{w}$ denoting the model vector, applied to $\mathbf{X}$, the data [8]. Using these variables, observe a mathematically sound graphical illustration of the hyperplane and its support vectors:

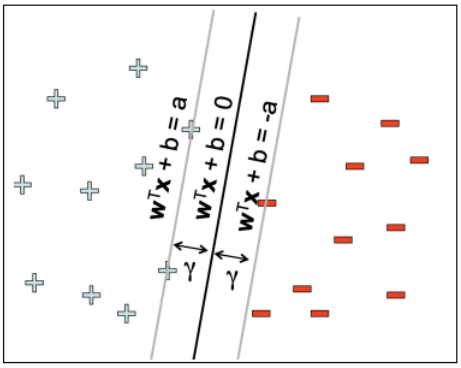

Figure 7. SVM Classification [8].

We can express this as an maximum optimization problem known as the "hard margin" optimization problem [8,9]. We would like to solve:

$$arg\max_{\mathbf{w},b} \frac{1}{||\mathbf{w}||}, (\mathbf{w}^T\mathbf{X}_l + b)(\mathbf{y}_l) \geq 1 \forall l$$

To take this a step further, we will want to solve the "soft margin" optimization problem [9]. The idea of the "soft margin" comes from the idea that, on its own, the data might not be so easily linearly separable, but instead have misclassifications among each of the classes separated by the hyperplane. For example, a positive example might be in the class of negative examples and without the kernel trick, may be difficult to correctly classify. The hyperparameter C is known as the "stretch parameter", the amount of classifications that are allowed in either class [9].

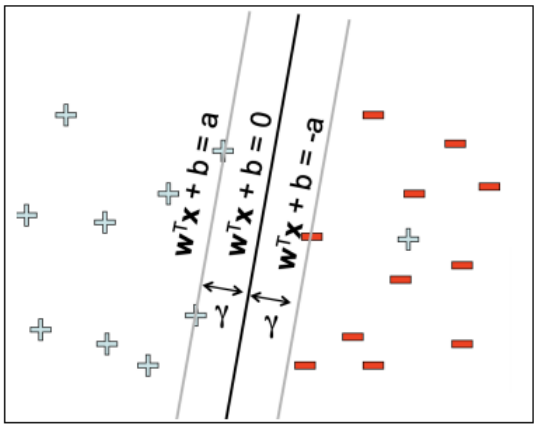

Figure 8. SVM Classification with misclassifications [8].

Then, our new optimization problem is the soft margin optimization problem. We would like to solve [9]:

$$arg\min_{\mathbf{w},\mathbf{b},\epsilon} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{l=1}^{M} \epsilon_l$$

This problem's dual form to which we can apply a kernel trick is [8,9]:

$$arg\max_{\alpha_1 \ldots \alpha_n} \sum_{l}^{M} \alpha_l - \frac{1}{2}\sum_{j=1}^{M}\sum k = 1^M \alpha_j \alpha_k y_j y_k K(X_j, X_k)$$

$$s.t.\ \alpha_l > 0\, \forall l \in \{training\},\ \sum_{l=1}^{M} \alpha_l y_l = 0,\ 0 \leq \alpha \leq C$$

Such a problem is solvable by minimizing the objective function [9]:

$$arg\min_{\mathbf{w},\mathbf{b}} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{l-1}^{M} max\{0,\ 1 - y_l(\mathbf{w}^T\mathbf{x}_l + \mathbf{b})\}$$

Concerining the loss function, this problem can be solved through what is known as quadratic programming. In short, quadratic programming is the method of solving an optimization problem by optimizing a quadratic objective function [10].

The Python implementation was soundly inspired by Randeep Ahlawat, who also utilized quadratic programming to solve this optimization problem [9]. The SVM algorithm is defined as a class, requiring the desired value for C and the type of kernel as hyperparameters to be used on in the algorithm. The choices of kernel presented are the linear kernel, the polynomial kernel, which requires another parameter, the polynomial degree, d, and the Gaussian kernel, which also requires another parameter, σ, to control the size of the kernel. Then, in the fitting function, to perform quadratic programming, the Python library CVXOPT was used [9]. CVXOPT has a module called 'solvers', which can solve the quadratic optimization program, if given five matrices and a bias vector [9]. Then, the model is calculated as above, with the optimal model, **w** and bias term, **b** being stored as instance variables of the SVM object. The fitting function requires two NumPy arrays, so the feature vectors and labels had to be converted before fitting the data. Then, using the optimal model created, the predict function lets the SVM object attempt to classify on the test data, its output another vector.

Using this SVM object in Python, I wanted to perform multiple experiments assessing how the accuracy changes depending on the parameters that are defined. For the values of $C = 0.1, 0.5, 1.0, 2.5, 4.0$ and using a linear kernel, I wished to examine how, as C changes, how the model accuracy changes as well. In other words, I wanted to examine if there was a relationship between C and the accuracy. In addition, for each kernel, linear, polynomial, and gaussian while holding C constant at 0.5, I wanted to see which kernel in the dual form performed the best at classification with the best model accuracy. Accuracy, the metric for performance of these models, will be calculated using a function. The accuracy function will check the test labels against the predicted test labels for performance assessment, totaling the correct predictions, and dividing the total by the length of the prediction vector.

4. Experiments and Results

To set up the experiments I wanted to run, I initialized SVM objects for each of them, so that they and their data could be stored and easily accessed. For each experiment, depending on the model, I fitted each model to the data, with 100 features and training-test splits of 16000 to 4000. My "base" support vector classifier was defined with a C-value of $\frac{1}{2}$ and the linear kernel, which are considered to be fairly common parameters.

The base model achieved an accuracy of 75.73%.

By the sheer high-dimensionality of the data, it is not possible to view the chosen hyperplane in 3D space. However, we can assess the performance of models from various values of C and various kernels to see if anything changes from using nonlinear kernels or different values of C.

4.1. Testing Values of C

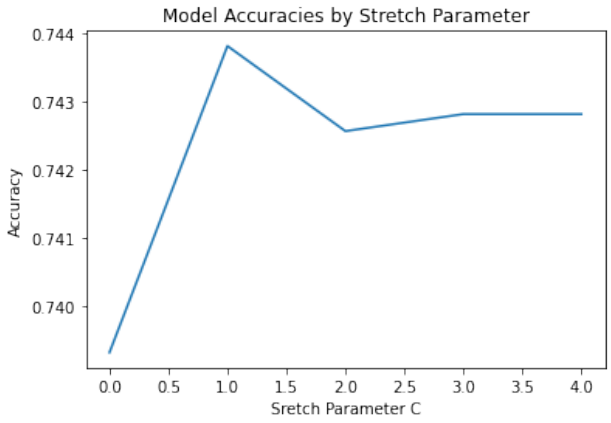

Figure 9. SVM Model Accuracies against value C.

As can be observed, the highest accuracy achieved for the linear kernel when the soft optimization problem was defined with the stretch parameter of $C = 1$. However, the difference between the highest and lowest accuracies for this collection of models is no more 0.4%.

4.2. Testing Different Kernels

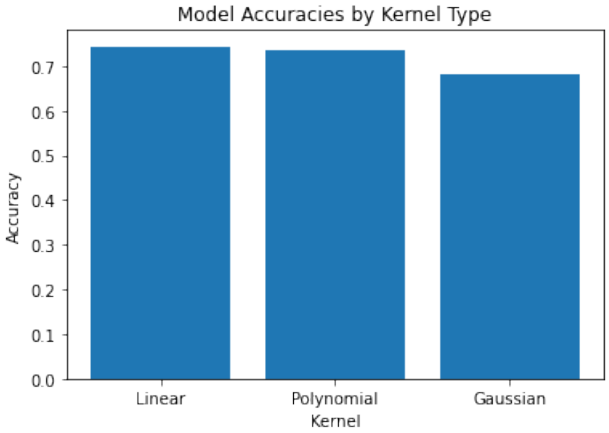

Figure 10. SVM Model Accuracies by various kernels.

As can be observed, the highest accuracy achieved for the linear kernel when the soft optimization problem was defined using the linear kernel. The Gaussian kernel performed the worst, with an accuracy of around 68%. Perhaps the data is most shaped in a linearly separable manner.

It should be noted that this differences in accuracy across these are relatively slim. The greatest difference between the linear kernel and the Gaussian kernel is 6%. Instead, accuracies fall in a ballpark of around 74.4%. Perhaps a combination of these parameters or a fundamental reworking of the model calculation can prove effective in raising the accuracy.

5. Availability and Reproducibility

I have published the Python program, data files, and resulting figures on my personal GitHub, *@leebri2n*. Data source and help resource credits are given to their respective owners.

This makes it so that with some tweaks to the data file paths, this program is replicable by anyone. It should be noted that any changes to the dataset of use requires that it should be structured such that the rows are the example data points, with the first column containing the text data and the second column containing the label, a 0 for negative examples, and 1 for positive examples. If the dataset is in order, then the models made in the program should be able to be constructed using any text.

The source code for this project can be found here:

References

[1] M. S. Neethu and R. Rajasree, "Sentiment analysis in twitter using machine learning techniques," 2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT), 2013, pp. 1-5, doi: 10.1109/ICCCNT.2013.6726818.

[2] T. Duque, "How to turn Text into Features," Towards Data Science, 2020. [Online].

[3] M. Yasser H., "IMDB Movie Rating Sentiment Analysis," Kaggle, 2022. [Online].

[4] G. Bedi, "A guide to Text Classification(NLP) using SVM and Naive Bayes with Python," Medium, 2018. [Online].

[5] A. Simha, "Understanding TF-IDF for Machine Learning," Capital One Machine Learning, 6 October, 2021. [Online].

[6] Q. Abbassi, "SVM From Scratch - Python," Towards Data Science, 7 February, 2020. [Online].

[7] CommonLounge, 2020. [Online].

[8] D. Bhattacharya, "CS4824/ECE4424: Support Vector Machine," Virginia Tech CS4824, 2022. [Online].

[9] R. Ahlawat, "SVM from scratch using Quadratic Programming," Medium, 24 May 2020. [Online].

[10] N. Manchev, "Fitting Support Vector Machines via Quadratic Programming," Domino Education, 8 June 2021. [Online].