Sentiment Analysis of Movie Reviews

Cody Perakslis(perak005), Calvin Roth(rothx195)

April 2018

Abstract

Increasingly, more products and applications must be designed to interact with with how people naturally communicate. This field of natural language processing has a diverse range of applications. Figuring out if something has a negative or positive sentiment is an important subproblem to understanding language. In this paper, we investigated six different methods of sentiment analysis: Naive Bayes, K-nearest neighbors, Logistic regression, Decision trees, Linear regression, and a deep neural network. We test these methods on a collection of IMDB movie reviews where we consider below 5 to be negative and above 7 to positive. We also discuss the results and various avenues of future work.

Introduction

Some of the most prominent recent develops in AI are programs that can interact seamlessly with humans. Examples such as Apple's Siri, Amazon's Alexa, and, revealed this week, Google's Duplex all attempt to interact with humans natural speech. This area of ongoing research, processing and reacting appropriately to human language, is known as natural language processing. One of the main branches in this field is sentiment analysis.

Sentiment analysis involves processing a segment of language to determine polarity (positive or negative feeling). This is a important field of research for personal assistants like Alexa, because knowing if something was said in a positive or negative way should influence how a truly intelligent AI program would act. Sentiment analysis has uses beyond just personal assistants. It can also be utilized in a different field known as recommender systems, predicting what items a person would like based on previous reviews. For example, we choose to do sentiment analysis on a dataset of the text of 50,000 IMDB reviews provided by Maas et al[7]. This is a good dataset for sentiment analysis as some of the pre-processing such as placing each review is in it's own .txt file with its score (out of 10) has been done for us. Movie reviews are a natural domain for sentiment analysis as they have a clear and measurable overall sentiment. The review dataset is divided into two categories of 25,000 reviews for training and another 25,000 reviews for testing. In each of those categories, there are 12,500 positive and 12,500 negative reviews. For this paper, we regard a review as positive if it's score is greater than 7 and negative if it is less than 5. All the reviews in the dataset fall within this range. This is important because there are no neutral movie reviews, any review is either good or bad.

This paper investigates how effective various machine learning techniques are at sentiment analysis on this dataset. The six major avenues we investigate are Naive Bayes, K-nearest neighbors, Logistic regression, Decision trees, Linear regression, and a deep neural network. This paper goes over background information and research others have done, how these algorithms were implemented, and a discussion of the results found for each.

Background

Sentiment analysis was first explored by the team of Pang, Bo and Lee [12].The researchers in this paper, describe briefly 3 methods used in other areas of natural language processing. There is naive Bayes filtering, a statistical based approach using Bayes' theorem and the assumption that all features are independent, maximum entropy, another stats based approach that does not need the assumption of linear independence, and support vector machines, a method that tries to create a mathematical that separates positive and negative reviews by the maximum amount. They found that SVMs performed best.

One intuitive approach that Whitelaw et al [19] based on the intuitive idea of tracking how many positive and negative words or phrases are in a document to determine the document's overall sentiment. In this approach they characterize a sentence as a unit called an appraisal group which consists of Attitude, orientation, Graduation, and polarity. Attitude is the distinction between if the phrase is an emotional claim, "this makes me sad", an intrinsic property, "This pillar is slender", or a societal view, "that man is idiotic". Orientation is if it is positive or negative. Graduation describes how intense and how broad this group is. Polarity describes if it contains words like not that can change the meaning dramatically. This processing is down using machine learning, a small set of prior known words, and access to a thesaurus. The results of this experiment were very positive, with the best parameters this scored over 90

The researchers of Serafin et. al. use a probabilistic method known as LSA(Latent sentiment analysis) to tackle a related problem, dialogue classification. As they say "LSA can be thought as representing the meaning of a word as a kind of average of the meanings of all the pas- sages in which it appears, and the meaning of a passage as a kind of average of the meaning of all the words it contains"[15]. Basically, they use a bag of words methods to store the occurrence of each word in each document they analyze and then algorithms are used to find the best statistically representation of a document of a vector of words from which it is classified based on what category it most represents, from earlier training.

Next up we have the papers of Young et al and of Kim both of which discuss the use of neural networks applied to natural language processing. Kim specifically uses a convolutional neural network[5]. Representing each word as a vector of k dimensions, Kim chooses to represent a feature, the inputs to the neural network, as a window of h words represented by their vector representations. The work of Young et al[20] discusses more in depth how the vectors are formed and other variants of neural networks like recursive neural networks along with for each type of neural network the network's motivation.

A different approach is to model language generation with a system Soricut et. al. implemented as an IDL expression[16]. An IDL expression is a set of some words that can have the following operations on: $\cdot(x, y)$ concats string x and y, $||(x, y)$ generates all interweavings of x and y so here xy and yx are generated, $\vee(x, y)$ generates x and y, and finally $Lock(x)$ treats the expression x as an atomic unit in a bigger expression.

From here, we generate a graph representing paths to go from an initial set to a final expression. They give expressions a weight based on how much information is required to encode them.

With the various methods and representations mentioned, there needs to be a way to test sentiment analysis. This means measuring proportion as well as polarity, and this is provided through the work of Maas et. al.[8] The work uses the vector model with an unsupervised learning component to capture word similarities, then a supervised portion on the IMDB dataset they provide. The dataset contains a list of movie review with the associated rating. This way polarity can be captured with some measure of proportion, while still giving a numerical value to that sentiment. The method of doing supervised learning after some measure of interpretation is one that we have found very interesting in this area, and these results will be a great baseline for our own.

Often the data given isn't wholly accurate or relevant. These methods are less effective when having to classify words as new when they are simply misspellings. Kolak et. al. defines a method of training a weighted finite state model with N-best scoring to probabilistically match a sequence of characters to a word in the dictionary.[6] Their use case was for even more error-prone data that comes from interpreting images of text into text. This method of pre-processing data could allow the effective use of more datasets where there are forms of review that are less well written. Another method of pre-processing done by Pang et. al. involves minimal cut to remove objective sentences. [11] Removing object (neutral) sentences allows for the sentences with real polarity to have more focus. The pre-processing also reduces further processing times after it. The danger of miss-classify a polar sentence as objective would be a great risk, however, so this is best for the purposes of compressing your data. It would be prone to miss subtle sentences as "This movie is a movie," which is an analytic sentence, as objective as possible, but seems to imply some lack-luster 2 to 3-star review. This implication is by saying an obvious objective sentence when more information is expected from the context of the review. Sentences of this kind are quite interesting but are far outside the scope of this area. These pre-processings have their place, and are great tools to aid in sentiment analysis.

Word chunks are one difficulty with the vector word model. Take for example "yeah right" takes on a negative polarity even though both of its constituent words have positive polarity. This difficulty more heavily plagues complex semantic processing, but it is a concern even here as well. For LSA, Deng et. al. incorporated an n-gram component to the algorithm to account for such issues. [3] An N-gram works to analyze not just single words, but word combinations. In this way "yeah right" would be represented by a 2-gram. A 2-gram formulation would account for certain negations as well and how the polarity often flips with a different proportion. For example "not bad" is more on par with "alright" then it is with "good". One difficulty with N-grams is determining the maximum value for N you will allow, and often those word combinations occur with less frequency than each constituent word, so learning from the data can be more difficult.

The N-gram model has other applications as well. Bojanowski et. al. used N-grams not at the word level but at the character level. [2] This approach can better catch entomological considerations of words through common roots. This has a great impact, especially with a smaller dataset, to find word associations with greater ease. Character associations, or in this case sub-word associations, are interesting avenues of representation. Sentiment analysis relies less upon the common word associations than other forms of semantic processing which may benefit more from this method.

In order to train systems for language processing, a lot of data is needed. This creates a problem in the time taken to train, often with parts of the data adding little benefit to the resulting word representations. Mikolov et. al. use neural nets to capture the multiple degrees of similarities that words have in less time with arguably greater accuracy. [10] They found that certain techniques in the area of NLP used in conjunction with a recurrent neural network brought significant results. They also offer a method of training a continuous bag of words and a skip gram model in a distributed fashion, allowing parallel processing to greatly reduce the time needed, and for the data itself to spread across multiple machines. This horizontal scaling makes sophisticated uses of these models more reasonable.

There are many avenues by which to explore sentiment analysis. Each representation and method helps solves for a specific set of challenges. The approaches in this paper covered those possible representations, processing, and pre-processing to better determine the base sentiment tied to our words.

Word Representations

Since we are working with computers not humans we can't just feed in an unprocessed paragraph and have the machine work on that raw text. Instead, we need a a way to represent words and text in a manner a computer can manipulate and use effectively. The first step is to create what is known as a bag of words model. A bag of words representation of text is simply each word, or possibly n-gram of words, and the number of times that it appears. This basic method was extended for our use in two different directions.. The first, word embeddings, represents each word has a real valued vector with the hope that words of similar meaning have similar vectors. The second option is TD-IDF which attempts to control for common words "the" or "I" dominating the data.

Word Embeddings

A common technique to represent words is as word embeddings. Given a fixed sized vocabulary of possible words, a word embedding represents each word has a vector of real values. The dimension of these vectors are often of dimension from 10 up to several hundred. The goal is to have words with similar meanings have similar representations. This captures meaning in language that a traditional bag of words model does not. For example, a bag of words model doesn't capture that 'wonderful' and 'fantastic' actually mean similar things, and as such they should be represented by similar vectors. We represent a whole review as a matrix where one line corresponds to the vector of a word from that review. This leads to the other advantage of this method: It is more efficient to work with this compact representation than a large sparse matrix. How to best create and train a system to do word embedding correctly is an ongoing research topic itself, so we will be using a prebuilt tool supplied by TensorFlow which was trained on a massive corpus of Google news articles and represents each word has a 128 dimensional vector[1].

Document Vectors

Document vectors represent each word as a scalar value, and all known words in a document, a review in this case, as elements of that document vector. Each new word found in the training set is added to the known

vocabulary. Any word in the testing set not in the known vocabulary is ignored, since it has no knowledge of the sentiment of that word.

For each review, a count is taken of each word in that review that is in the vocabulary. The count is the term frequency (tf) for that word. The more times it occurs, the more weight it should be given in determining that review's sentiment. To account for words that are simply common, and thus should be given less weight, the term frequency is multiplied by the inverse document frequency (idf). The idf is the $\log(\frac{|\mathcal{I}|}{|\mathcal{I}_j|})$ where $|\mathcal{I}|$ is the number of documents in the training set, and $|\mathcal{I}_j|$ is the number of documents in the training set that contain term j. The idf is computed for each term in the vocabulary given the documents in the training set.

For the testing set, each review is parsed to determine the counts of each word in the review that is in the vocabulary. Then, that count (the term frequency) is multiplied by the associated idf, given what is known as the tf-idf of each word, a scalar value for that word and a sparse matrix for the entire review.

Tensorflow

Panda Dataframes

An important implementation detail is how we handle this dataset and load it into training. We use a python package known as Pandas that eases the complexity of these initial task of loading the data into train. We load the into our program using with the following code

```
def load_directory_data(directory):
    data = {}
    data["review"] = []
    data["score"] = []
    for file_path in os.listdir(directory):
        with tf.gfile.GFile(os.path.join(directory, file_path), "r") as f:
            data["review"].append(f.read())
            #This regex will get from the name of a file like '250_1.txt', 1(numbers after
            data["score"].append(re.match("\d+_(\d+)\.txt", file_path).group(1))
    return pd.DataFrame.from_dict(data)

# Merge positive and negative examples, add a polarity column and shuffle.
def load_dataset(directory):
    pos_df = load_directory_data(os.path.join(directory, "pos"))
    neg_df = load_directory_data(os.path.join(directory, "neg"))
    pos_df["polarity"] = 1
    neg_df["polarity"] = 0
    return pd.concat([pos_df, neg_df]).sample(frac=1).reset_index(drop=True)
```

Here we create a table with one column labelled review which is the text of the review, one column labeled score, and one column for polarity (good or bad). This code loads the reviews in from directories

	review	score	polarity
0	I've always liked Fred MacMurray, andalthough...	4	0
1	I heard about this film and knew it wasn't rea...	1	0
2	There are a lot of pretentious people out ther...	1	0
3	i liked this film a lot. it's dark, it's not a...	9	1
4	With few exceptions, most of George Bernard Sh...	8	1
5	Legendary movie producer Walt Disney brought t...	10	1
6	Minimal script, minimal character development,...	2	0
7	Depardieu's most notorious film is this (1974)...	10	1
8	I rented this movie because it sounded pretty ...	1	0
9	** HERE BE SPOILERS ** The governme...	2	0
10	French film directors continue to amaze with t...	10	1
11	I think the comments regarding the show being ...	10	1
12	I've seen this movie when I was young, and I r...	10	1
13	Phil the Alien is one of those quirky films wh...	4	0
14	A friend once asked me to read a screenplay of...	3	0
15	Like wearing a hair shirt. Positively, absolut...	1	0
16	Though the title includes the word "zombies", ...	4	0
17	Joan Crawford had just begun her "working girl...	8	1
18	Over several years of looking for half-decent ...	2	0
19	I never bothered to see this movie in theaters...	9	1
20	As far as parody films go, there are few that ...	7	1
21	"Fido" is to be commended for taking a tired g...	7	1
22	This is possibly the best short crime drama I'...	10	1
23	Let me state first that I love Westerns & Civi...	4	0
24	I saw this on the big screen and was encapsula...	10	1
25	Now, I've seen a lot of bad movies. I like bad...	1	0
26	This is a truly awful film. What they have don...	1	0
27	Steven Spielberg wanted to win an Oscar so bad...	10	1
28	I first saw Thief as a child which makes me al...	10	1
29	Intriguing. Exciting. Dramatic. Explosive. Com...	10	1
...	...	...	...
24970	i rate this movie with 3 skulls, only coz the ...	3	0
24971	Well, What can I say, other than these people ...	10	1
24972	I am a fairly big fan of most of the films tha...	10	1
24973	1st watched 6/18/2009 2 out of 10 (Dir- Pete...	2	0
24974	All Hype! What better way to describe a movie ...	2	0

Figure 1: How a panda dataframe is represented. The first column is what index of the dataframe we are looking at, next the review itself, and finally the polarity of the review

pos and neg, joins them and then shuffles the data.

Deep Neural Networks

Deep neural networks are powerful tools of machine learning. Neural networks borrow from biology and try to mimic the way that neurons learn. A neural network is a series of layers of neurons where all the neurons in one layer feed into the next layer. The first layer of the network, the input, here a movie review. The final layer is the output, here representing if that review is positive or negative. In between we have what is called the hidden layers. Hidden layers apply transformations to the information coming the the previous layer to potentially model more complex behavior. The way it works is that every connection has some weight, some measure of how important the value of that earlier neuron is too the value of that neuron in the next layer. If a_i is the value of neuron i in layer 1 and the weight of importance $w_i$they have on a neuron in layer 2 called p_i is w_i then the value of this neuron is

$$(\sum_i w_i a_i) + b$$

where b is known as a bias, some constant some we add to our value of p_i. Typically we want to value of each neuron to have a range between 0 and 1. We do this by using the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. The process of changing these the weights and biases to minimize the output layer's error is how our network 'learns' and gradually performs better. To figure out how we should adjust our the weight and

bias for each neuron we look at how wrong our answer once on a training sample. We define our cost function C to be $\sum(Observed - Expected)^2$. From calculus we know that the gradient of this function is the direction which increases this function fastest, so we move in the negative gradient direction where each dimension corresponds to one connection in our neural network. This process how adjusting the weight of our connections based on the gradient of some cost function is known as gradient descent. Using the gradient also has the feature that a connection that is very wrong will have a large gradient and change more as compared to a connection that is mostly right, that is we adjust connections proportionally to how much they negatively affect the output.

To create this neural network first we initiate the neural network. Here h_unit is an array representing the hidden layers, [100,300] would be two hidden layers with the first layer having 100 neurons and the 2nd having 300. We choose to have two hidden layers one of 600 neurons and one of 200 neurons. Interestingly, in general there is very little insight into how many neurons and layers a particular problem should optimally have. The values we chose are semi arbitrary, large networks didn't produce notably better results, typically with at best performing 1% better but with a significant increase in time to run. The feature column variable is our input layer which we represent as an embedding column as described above. The n_classes variable is the number of output options, here two as we only care if a review is good or bad. Finally, we use the adam optimizer that helps us optimize this network in a more easy computational way.

```
classifier = tf.estimator.DNNClassifier(
    hidden_units= h_unit,
    feature_columns=[embed_col],
    n_classes=2,
    optimizer=tf.train.AdamOptimizer(learning_rate= l_rate))
```

Next we have to train our model, where step is how many times to step through this model. input_fn is a function that gives of access to the data which we are using Pandas to represent.

```
classifier.train(input_fn=train_input_fn, steps=step);
```

Finally, we have to evaluate how accurate our model is. We do this by retesting how well our model does against the training model as a whole and how well it does against test data, reviews it has not seen before. This will give us of a percentage of the cases where our output is correct.

```
train_eval_result = classifier.evaluate(input_fn=predict_train_input_fn)
test_eval_result = classifier.evaluate(input_fn=predict_test_input_fn)
```

Optimizers

The task of optimizing thousands of connections is computationally daunting so frequently more advanced tools are used. Known as optimizers, these build off of gradient descent to help a cost function converge quicker. We use a popular one called Adam which uses previous gradients to adaptively optimize certain variables to converge more quickly for many applications. The learning rate parameter is used in this method and affects how much a single connection can change in any one iteration.

Scikit-Learn

Sci-kit learn is a Python library for machine learning. We used the library to construct the document vectors to apply K-Nearest Neighbor, Multinomial Naive Bayes, Logistic Regression, Decision Trees, and Linear Regression algorithms to the data as well as do some pre-processing. The default values for the algorithms are used.

K-Nearest Neighbors

The K-Nearest Neighbors (KNN) algorithm for classification takes the k nearest classified neighbors to a new data point to determine that data point's classification. The computational complexity comes from running the algorithm, not training it, since training involves just giving points coordinates for distance determination. The number of neighbors is 5, and the distance is the Minkowski distance , determined by $(\Sigma_{i=1}^{k}(|x_i - y_i|)^q)^{1/q}$ for k-dimensions per point with a default of $q = 2$. When $q = 1$ the Minkowski distance is the Manhattan distance, and when $q = 2$ the Minkowski distance is the Eucledian distance of $\sqrt{\Sigma_{i=1}^{k}(x_i - y_i)^2}$. The point weighting is uniform, so whichever polarity has 3 or more neighbors in the 5 closest neighbors decides the polarity of the new data point. Weighting can be done in more complicated ways, more commonly by inverse of the distance within the k-nearest neighbors. Trtenjak et. al. found that K-nearest neighbors worked well with tf-idf in natural language processing.[17] This follows from Minkowski distances requiring continuous data to produce effective results.

Multinomial Naive Bayes

Multinomial Naive Bayes (MNB) uses Baye's theorem the assumption that the order of words is irrelevant to the sentiment. This would not be as effective for semantic analysis, where order matters more, but can work well for classifying words by their polarity. It treats each word of the document as a multinomial, which generally follows from discrete integers, but as found by Rennie et. al., Multinomial Naive Bayes over tf-idf can produce effective text classification.[13] The general algorithm as expressed by Rennie et. al. is

$$p(d|\vec{\theta_c}) = \frac{(\Sigma_i f_i)!}{\Pi_i f_i!}\Pi_i(\theta_{c_i})^{f_i}$$

where $\vec{\theta_c}$ is the parameter vector for each word i in the vocabulary of class c. In this case there are two classes, one for each polarity. f_i is the frequency of word i in document d, tf-idf instead of frequency in this case. Both classes are calculated, and the one with the higher probability is chosen for classification. The default also incorporates additive smoothing, which gives a value $\alpha = 1$ to words not in a given document in the following form:

$$\theta_i = \frac{f_i + \alpha}{\alpha|\mathcal{I}| + \Sigma_{j\in\mathcal{V}} f_j}$$

where $\mathcal{I}$ is the set of documents, $\mathcal{V}$ is the vocabulary.[18] This equation is specific to the parameter of a single class relative to a given document.

Logistic Regression

Logistic Regression (LogReg) fits the data to a sigmoid curve to give a binary response (in this case sentiment). Sachan et. al. focused on the use of lexicons, but the work demonstrates the power of logistic regression, especially over this same dataset.[14] The parameters for logistic regression are determined by gradient descent, finding the derivative of the squared error to determine in which direction to adjust the variable, then moving by that amount multiplied by some learning rate α. This converges to some local optimum. By default $\alpha = 1$, and no change occurs when the magnitude of the parameter is below the default tolerance of 10^{-4}. The sigmoid function works well for this regression, since $\sigma'(x) = \sigma(x)(1 - \sigma(x))$.

Decision Trees

Decision trees (DTree) involve splitting the dataset by criteria into sub-decision trees until the leaf nodes give the categorical output. Decision trees often support natural language processing for grammar categorization.[9] In this paper, we train a decision tree to determine polarity based on tf-idf, but the literature would not suggest this as an effective method. A split parameter is chosen as the one that would most reduce the impurity, where impurity is $1 - (p_0^2 + p_1^2)$ where p_i is the fraction of items labeled i in a given group. With an impurity of 0, all items are of the same label. The defaults don't stop until all the leaf impurity's are 0 or there are no more parameters to split on, which creates the danger of overfitting. This can be handled by post-growth phase pruning, or by setting values in scikit for the minimum impurity decrease to split or the maximum depth.

Linear Regression

Linear regression (LinReg) fits a line (in multiple dimensions) to the data. Joshi et. al. found success using linear regression over movie text and other features to predict opening week revenue.[4] Joshi et. al. used bi-grams and tri-grams over stop words, which we do as well for sentiment analysis. Linear regression can also be solved using gradient descent in the same manner as logistic regression. The difference is that the equation is of the form $f(\vec{x}) = \theta_0 + x_1\theta_1 + ... + x_n\theta_n$ for n words in the vocabulary $\mathcal{V}$. Linear regression is better suited for making predictions within a continuum, instead of the classification problem being given.

Results

Tensorflow Results

Steps	Training Set Accuracy	Test Set Accuracy
100	0.792360008	0.785520017
200	0.799799979	0.793919981
300	0.798080027	0.789799988
400	0.805320024	0.794080019
500	0.809159994	0.796119988
600	0.808719993	0.794839978
700	0.796720028	0.782880008
800	0.813279986	0.795239985
900	0.810479999	0.790799975
1000	0.821839988	0.800040007
2000	0.841880023	0.79272002
3000	0.862879992	0.783959985
4000	0.864799976	0.756160021
5000	0.906599998	0.760599971
6000	0.915279984	0.762759984
7000	0.920920014	0.75308001
8000	0.944920003	0.762600005
9000	0.944320023	0.750320017
10000	0.939559996	0.762239993
11000	0.95231998	0.76384002
12000	0.961199999	0.749080002

Table 1: This table is the number of iterations that the model was trained for and how that affected the accuracy against the Training Set of previous seen data and against the test set of new data

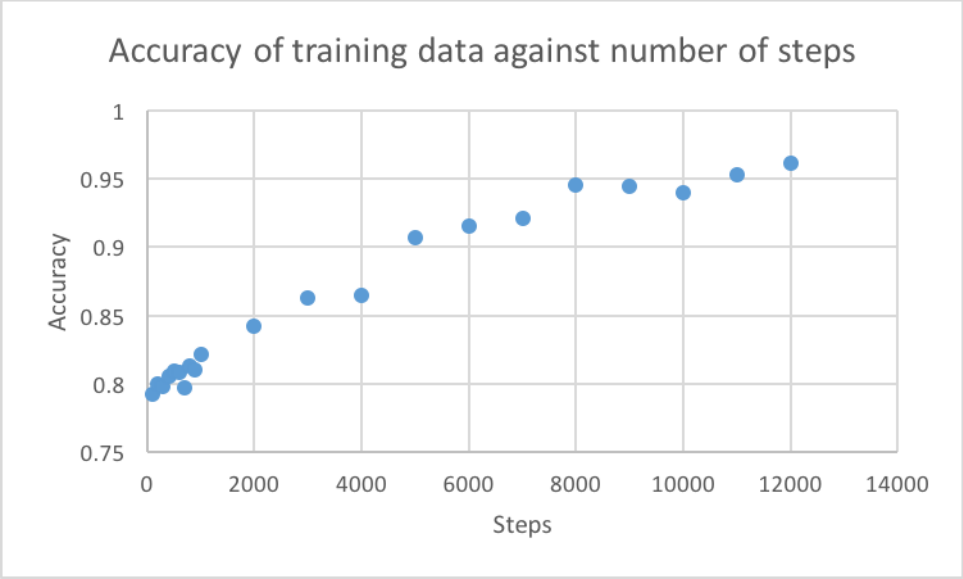

Figure 2: This graph shows how the accuracy of the model tested against the training data performs as a function of the number of training steps

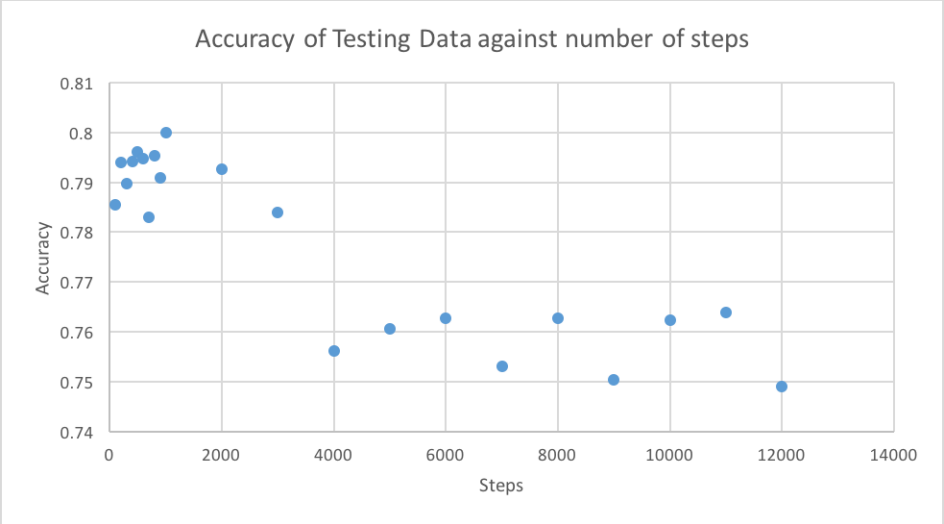

Figure 3: This graph shows how the accuracy of the model tested against new data varies as a function of steps in the training

Testing on the deep neural network(DNN) was done with two hidden layers, the first being 600 neurons and the second having 200 neurons and a learning rate of 0.03. The results for the test against the training data itself make intuitively make sense, we have low accuracy that improves steadily as the number of training steps increases with some diminishing returns for the large step sizes. But more interestingly, for the against the test data our results get notably worse when the model has been trained longer. This behavior on new

data from more training is known has overfitting. We are learning too much about the particular details of our training data, which helps our model do well against training data but worse against general data, because we are trying to apply those specific patterns to data where it does not fit.

Sci-Kit Learn Results

N-grams

Often single terms cannot reflect the sentiment of a phrase. Take "not bad" as an example. Both those words alone carry a negative sentiment, but together the sentiment becomes positive. N-grams can account for these phrases by taking words in groups of N, instead of simply in groups of one. The larger the value of N, the more data to sift through, so classification can take longer and have reduced efficacy due to some N-grams producing extraneous noise into the decision.

Algorithm	N-gram	Training Efficacy	Testing Efficacy	Time Taken
KNN	1	0.8598	0.6615	183.59
KNN	2	0.8473	0.6683	242.41
KNN	3	0.8364	0.6718	217.61
MNB	1	0.9089	0.8296	**000.50**
MNB	2	0.9703	0.8683	001.31
MNB	3	0.9904	0.8768	002.17
LogReg	1	0.9333	0.8832	001.26
LogReg	2	0.9572	**0.8862**	004.95
LogReg	3	0.9696	0.8775	009.82
DTree	1	1.0000	0.7042	044.30
DTree	2	1.0000	0.7001	254.14
DTree	3	1.0000	0.6877	**757.15**
LinReg	1	1.0000	**0.0390**	057.74
LinReg	2	1.0000	0.6060	029.60
LinReg	3	1.0000	0.6186	049.79

Table 2: Machine learning algorithms in uni-, bi-, and tri- grams listing accuracy on training data and testing data, as well as the time taken to train and then test both data sets.The best and worst testing efficacy scores and timings are bolded.

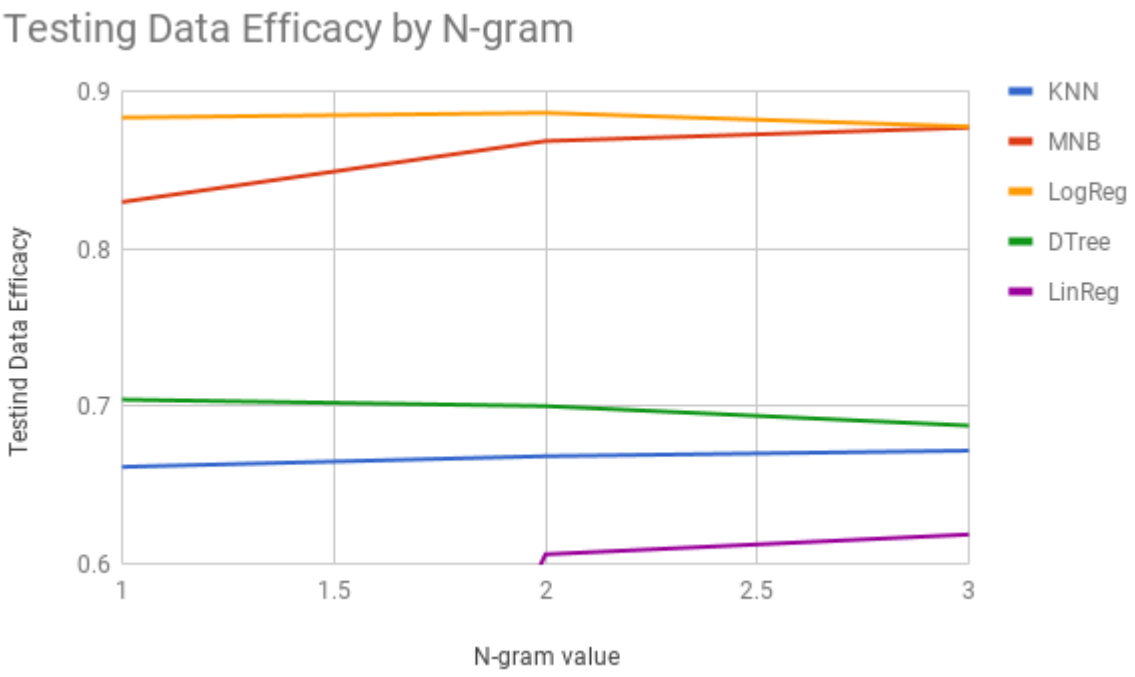

Figure 4: Comparing N-gram values of 1,2, and 3 to the testing efficacy of the five machine learning algorithms we are testing. The data comes from Table 2.

The testing results tend to improve using a bi-gram, but have marginal or reduced benefit beyond that. The most dramatic increase came from the linear regression algorithm, whose testing efficacy increased over 15 fold between a uni-gram and a bi-gram. Bi-grams allow for reversals such as "not bad," as mentioned above, so these results fit with our understanding of natural language.

Stop Words

Another pre-processing step could be to remove stop words. Stop words are words such as "the", "and", "a", etc. which are so common to add little value to keep track of, so they are ignored. With more complex semantic processing, removing these words could seriously damage comprehension, but it can benefit sentiment analysis by removing these neutral terms.

Algorithm	Training Efficacy	Testing Efficacy	Time Taken
KNN	0.8473	0.6683	242.41
MNB	0.9703	0.8683	001.31
LogReg	0.9572	0.8862	004.95
DTree	1.0000	0.7001	254.14
LinReg	1.0000	0.6060	029.60

Table 3: Scikit Algorithms as bi-grams without removing stop words.

Algorithm	Training Efficacy	Testing Efficacy	Time Taken
KNN	0.8602	0.6302	114.84
MNB	0.9844	0.8544	000.80
LogReg	0.9626	0.8735	002.59
DTree	1.0000	0.7060	269.03
LinReg	1.0000	0.5616	016.98

Table 4: Scikit Algorithms as bi-grams with stop words removed.

The removal of stop words reduces the effectiveness of these algorithms, but also reduces the training time. Stop word removal gives too little information to be as effective as standard bi-grams. Stop word removal would be the strategy if training and classification speeds where more important than a marginal efficacy increase.

Discussion

Logistic regression gave the best testing accuracy with 88.62% on bi-grams without removing stop words, which is close to the 90.53% achieved using logistic regression with n-grams on tf-idf by Sachan et. al.[14]. We see that the deep neural network(DNN) also gets respectable results at 80% and a moderate amount of time taking 48 seconds for the test achieving the highest results, 2 hidden units of 600 and 200 neurons with 1000 steps of training. Comparing all of the scikit algorithms over bi-grams without removing stop words to the neural net, we found the following orders for testing data efficacy and timing results:

Algorithm	Training Efficacy
LogReg	0.8862
MNB	0.8683
DNN	0.80
DTree	0.7001
KNN	0.6683
LinReg	0.6060

Table 5: Algorithms in order of training efficacy. Scikit algoritms as bi-grams without removing stop words. DNN of two hidden layers (600 and 200 neurons), 1000 training steps, and a learning rate of 0.03.

Algorithm	Time Taken
MNB	001.31
LogReg	004.95
LinReg	029.60
DNN	48.37
KNN	242.41
DTree	254.14

Table 6: Algorithms in order of completion speed. Scikit algoritms as bi-grams without removing stop words.DNN of two hidden layers (600 and 200 neurons), 1000 training steps, and a learning rate of 0.03.

Overfitting

Overfitting the training data leads to reduced accuracy over the testing data. Within the scikit algorithms, decision trees and linear regression demonstrate supreme overfitting. Both score 100% on the training data,

but achieve at most 70.32% and 61.86% respectively. The purpose of the testing set is often to measure and reduce such overfitting, either pruning or hampering growth of decision trees or by using fewer parameters for linear regression. These scikit algorithms are on the extremes, but all of the algorithms fit the training set better than the testing set, often substantially. As mentioned above the neural network has pronounced overfitting too for step sizes larger than 2000 results become noticeably worse.

Future Work

Each of these algorithms could be adjusted to give different results. Pruning algorithms could be applied to the decision tree to produce better testing efficacy. Similarly, more data on a smaller vocabulary could result in less overfitting. There is also method called cross validation, a method that can sometimes help reduce overfitting be also be worth exploring in future work. The K-value for K-Nearest Neighbors could be variable instead of fixed. More combinations of n-grams, stop words, and other pre-processing methods could be used to enhance the performance further. In particular, only one grams were considered in the neural network, considering two grams could very possibly prove a fruitful track to explore.One of the may avenues that we could explore preprocessing is by in each review and removing sentences that are purely objective. A sentence such as "I saw this movie on Thursday" for example could probably be ignored. By ignoring this sentences that shouldn't factor into sentiment, our training would focus only on the sentences that actively contribute to the overall sentiment. Another area of potentially advanced is a deeper knowledge of deep neural networks how getting a different number of layers and neurons that produces better results. The literature mentions using Support Vector Machines, which would be an important algorithm to compare, but it took over 35 minutes to train to 65.62% accuracy on uni-grams, so was left out of the analysis here.

Contributions by Team Member

Cody Perakslis

- Coded and tested scikit algorithms
- Wrote sections over the scikit algorithms
- Co-wrote background section
- Wrote document vectors word representation section
- Co-wrote discussion section and its subsections

Calvin Roth

- Coded and tested the TensorFlow neural net
- Wrote sections over TensorFlow and neural nets
- Co-wrote background section
- Wrote word embedding word representation section

- Co-wrote discussion section and its subsections
- Wrote introduction and abstract

References

[1] Module google/nnlm-en-dim128/1. https://www.tensorflow.org/hub/modules/google/nnlm-en-dim128/1. Accessed: 2018-05-10.

[2] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[3] Y. Deng and S. Khudanpur. Latent semantic information in maximum entropy language models for conversational speech recognition. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 56–63, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[4] M. Joshi, D. Das, K. Gimpel, and N. A. Smith. Movie reviews and revenues: An experiment in text regression. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 293–296, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[5] Y. Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751, 2014.

[6] O. Kolak, W. J. Byrne, and P. Resnik. A generative probabilistic OCR model for NLP applications. In M. A. Hearst and M. Ostendorf, editors, *Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, HLT-NAACL 2003, Edmonton, Canada, May 27 - June 1, 2003*. The Association for Computational Linguistics, 2003.

[7] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[8] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pages 142–150. Association for Computational Linguistics, 2011.

[9] R. MarilenaăLAZ and D. Militaru. The role of decision trees in natural language processing. 2015.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[11] B. Pang and L. Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*, page 271. Association for Computational Linguistics, 2004.

[12] B. Pang, L. Lee, and S. 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.

[13] J. D. M. Rennie, L. Shih, J. Teevan, and D. R. Karger. Tackling the poor assumptions of naive bayes text classifiers. In *In Proceedings of the Twentieth International Conference on Machine Learning*, pages 616–623, 2003.

[14] D. S. Sachan, M. Zaheer, and R. Salakhutdinov. Investigating the working of text classifiers. *CoRR*, abs/1801.06261, 2018.

[15] R. Serafin, B. Di Eugenio, and M. Glass. Latent semantic analysis for dialogue act classification. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: Companion Volume of the Proceedings of HLT-NAACL 2003–short Papers - Volume 2*, NAACL-Short '03, pages 94–96, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[16] R. Soricut and D. Marcu. Towards developing generation algorithms for text-to-text applications. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 66–74, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[17] B. Trstenjak, S. Mikac, and D. Donko. Knn with tf-idf based framework for text categorization. *Procedia Engineering*, 69:1356 – 1364, 2014. 24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013.

[18] D. Valcarce, J. Parapar, and A. Barreiro. Additive smoothing for relevance-based language modelling of recommender systems. In *Proceedings of the 4th Spanish Conference on Information Retrieval*, CERI '16, pages 9:1–9:8, New York, NY, USA, 2016. ACM.

[19] C. Whitelaw, N. Garg, and S. Argamon. Using appraisal groups for sentiment analysis. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 625–631. ACM, 2005.

[20] T. Young, D. Hazarika, S. Poria, and E. Cambria. Recent trends in deep learning based natural language processing. *CoRR*, abs/1708.02709, 2017.