IMDB Movie Review Sentiment Analysis

Parth Singh

March 16, 2018

1 Definition

1.1 Project Overview

Analysis and recognition of sentiments is a fundamental task designated under natural language processing. It has found a broad spectrum of utility including recommendation systems, processing reviews and social media comments. Vector representations of information, more specifically words, such that the constructed vector space behaves in way that is useful for analysis is crucial to the idea of vector embedding. And when this idea is applied to words or phrases it is called word embedding. Word embedding is now a popular and effective technique that creates input which is suitable to operate upon by a model such an LSTM (Long Short Term Memory) network or a Convolutional Neural Network. One such embedding commonly available is the Word2Vec model and it performs well in trying to capture semantic closeness between words, and it is not very cumbersome either. This project is an attemp at creating a classifier for movie reviews from the IMDB dataset[1]. This is of course not the first attempt at doing so, consider visiting this paper[2] by Mass et al which demonstrates the power of embedding techniques.

[1] https://keras.io/datasets/#imdb-movie-reviews-sentiment-classification
[2] http://ai.stanford.edu/~amaas/papers/wvSent_acl2011.pdf

1.2 Motivation

Since the rise of social media platforms, and the vast expansion of customer reviews on websites like Amazon, Yelp etc there has been an explosion in the amount of data we have access to which represents sentiments of customers or humans via reviews or comments. This collection of datasets can be analyzed in various ways to understand the general feeling of people towards specific products or events. Machine learning techniques can be harnessed in several ways to create such classifiers. In particular I am interested in comparing some of them which are suited to such problems. Essentially I will be exploring natural language processing tools to develop a movie review classifier.

1.3 Problem Statement

IMDb (Internet Movie Database) is an online database of information related to films, TV shows, video games etc. The database is owned and operated by a subsidiary of Amazon. IMDb includes for all these movies, shows etc reviews and ratings from fans.

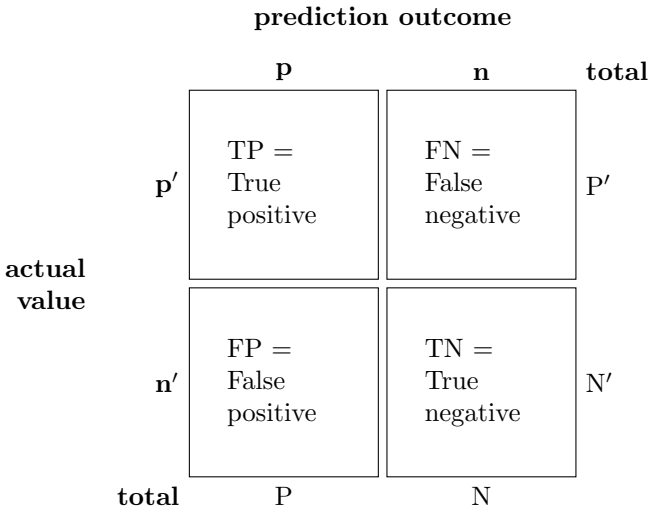
The dataset was curated by Mass et al. and is easily accessible[3]. They collected 50K IMDb movie reviews. A rating of < 5 implies a negative sentiment while a rating of ≥ 7 implies a positive sentiment. No movie has more than 30 reviews in the dataset in order to not introduce a bias where some movies may have overwhelmingly numerous reviews and it may also lead to a correlation of reviews as reviewers may begin to get influenced by past reviews they read. There are 25K positive reviews and 25K negative reviews among the set. Neutral reviews were not included in the dataset. I will split the dataset into training and testing sets.

I plan to attack the problem using several methods. Firstly after some text pre-processing (embedding - to be discussed later), I will try and train three different models. Namely, a Multi-Layer Peprceptron, a Convolutional Neural Network and a Recurrent Neural Network (Long Short Term Memory). After finding a suitable architecture I will compare the results of these methods. I also plan to use the tf-idf technique and combine it with a Multinomial Naive Bayes Classifier to test a less complex method as this provides another validated method to test against. All these techniques are justifiably fit for this problem as they will operate on a dataset which will be tuned in an appropriate way for them (embedding for NNs and tf-idf for MNB). These methods will basically examine the vocabulary of the reviews and try and train the model to understand what makes a movie review good or bad.

1.4 Metrics

- **Confusion Matrix** [4] This is perhaps one of the most essential metric was classification problems. $C_{i,j}$ is an element of matrix which informs us how many elements of type i have been classified or identified with a label j. The best case scenario would be diagonal confusion/error matrix where there exists no incorrect classification. The toy matrix shown below is a representation for my binary classification. Positive (P) represents positive move reviews and N (negative) represents negative movie review

[3] http://ai.stanford.edu/~amaas/data/sentiment/
[4] https://doi.org/10.1016/S0034-4257(97)00083-7



Confusion Matrix

- **Accuracy:** The most basic metric and as we know not the best. It just tells us how many items have been classified correctly. Accuracy has been chosen because the dataset is balanced in terms of positive and negative reviews.

$$Accuracy := \frac{TP + TN}{Total} \tag{1}$$

- **Precision and Recall:** [5] These two metrics inform us about the some of the non-diagonal components of the confusion matrix *Precision* informs us how many selected items are relevant or specifically in this problem it informs us basically what fraction of positively classified reviews are truly positive and *Recall* informs us how many relevant items are selected and in this problem it tell us what fraction of positively classified reviews are classified correctly. These two metrics are important since they might inform us about asymmetries in classification. For example if the model may have a tendency to classify negative reviews as positive or vice-versa. This would be especially useful if the dataset was unbalanced but still finds utility in our example.

$$Precision := \frac{TP}{TP + FP} \tag{2}$$

$$Recall := \frac{TP}{TP + FN} \tag{3}$$

$$\tag{4}$$

- **f**β Score: [6] This is a metric that combines the results of precision and recall in a weighted fashion. By setting $\beta = 1$, it generates the harmonic mean of precision and recall. This metric is basically a collective representation of precision and recall and is a nice and concise way to judge a model's overall performance.

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}} \tag{5}$$

[5] http://www.flinders.edu.au/science_engineering/fms/School-CSEM/publications/tech_reps-research_artfcts/TRRA_2007.pdf

[6]

2 Analysis

2.1 Data Exploration and Visualization

When we are training the model, we are dealing with reviews that are either positive or negative and the dataset is balanced with respect to both as discussed earlier.

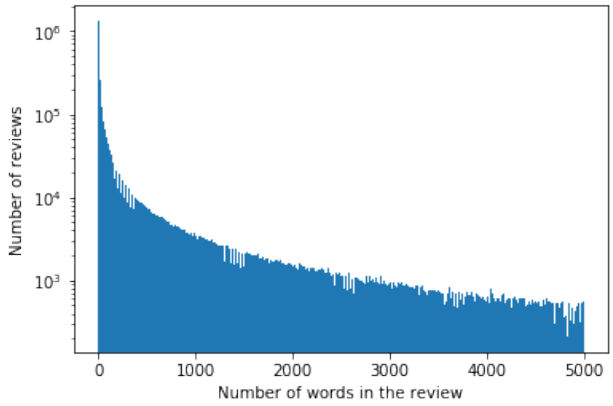


Figure 1: Spectrum of review length in words

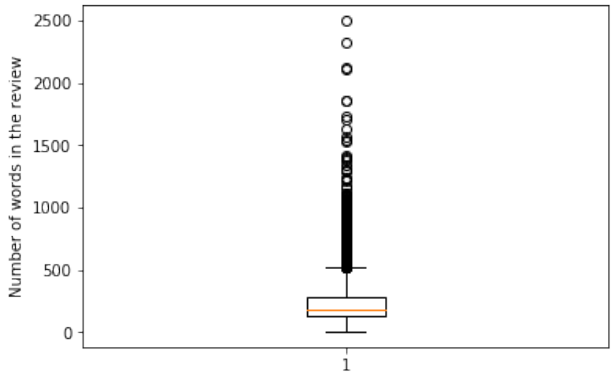


Figure 2: BoxPlot of number of words in the review dataset

The dataset for this problem is being imported from the built in keras dataset. Reviews have already been pre-processed, and each review is encoded as a sequence of word indexes (integers). Words are indexed by overall frequency in the dataset. For example, the integer "10" encodes the 10th most frequent word in the data. This permits fast filtering operations such as: "consider the top 5000 most common words".

The mean length of the review is 234.76 words. The standard deviation of the review length is 172.9 words The number of unique words in the entire dataset is 4998.

Clearly from the two plots above we can see an exponential distribution of the review length. Most of the reviews are short i.e. less than 500 words. In fact as can be seen from the box plot, 500 words is approximately 1.5 times the inter-quartile range of the distribution. This is a good take-away for us. One may begin to think that for classification purposes we might be able to work with reviews which are about 500 words in length or perhaps 2 or 3 times longer to be more inclusive and accurate.

This is a sample review from the training set. The integer indices have been converted to words using the *get _word _index()* function. This is not truly the raw data (the original movie review). Basic text

pre-processing has already been performed. This includes creating a bag of words, removing stopwords and lemmatization/stemming. In order to gain an idea of what kind of processing has been performed it is worth looking at this tutorial[7]

```
the, clear, fact, entertaining, there, life, back, br, is, and, show,
of, performance, stars, br, actors, film, him, many, shold, movie,
reasons, to, and, reading, and, are, in, of, scenes, and, and, of,
and, ot, compared, not, boss, yes, to, and, show, its, disappointed,
fact, raw, to, it, jstice, by, br, of, where, clear, fact, many, yor,
way, and, with, city, nice, are, is, along, wrong, not, as, it, way,
she, bt, this, anything, p, "havent", been, by, who, of, choices, br,
of, yo, to, as, this, "id", it, and, who, of, shot, "yoll", to, love,
for, and, of, yo, it, is, seqels, of, little, qest, are, seen,
watched, front, chemistry, to, simply, alive, of, chris, being, it,
is, say, easy, and, cry, in, chemistry, bt, and, all, it, maybe, this,
is, wing, film, job, live, of, and, relief, and, level, names, and,
and, to, be, stops, serial, and, watch, is, men, go, this, of, wing,
american, from, and, moving, is, accepted, pt, this, of, jerry, for,
places, so, work, and, watch, and, lot, br, that, from, sometimes,
wondered, make, department, introduced, to, wondered, from, action,
at, trns, in, low, that, in, gay, "im", of, chemistry, bible, i, i,
simply, alive, it, is, time, done, inspector, to, watching, look,
world, named, for, more, tells, p, many, fans, are, that, movie, msic,
her, get, grasp, bt, seems, in, people, film, that, if, explain, in,
why, for, and, find, of, where, br, if, and, movie, throghot, if, and,
of, yo, best, look, red, and, to, recently, in, sccessflly, mch,
nfortnately, going, dan, and, stck, is, him, seqences, bt, of, yo, of,
enogh, for, its, br, that, beatifl, pt, reasons, of, chris, chemistry,
wing, and, for, of, yo, red, time, and, to, as, companion, and, of,
chris, less, br, of, sbplots, tortre, in, low, alive, in, gay, some,
br, of, wing, if, time, actal, in, also, side, any, if, name, takes,
for, of, friendship, it, of, 10, for, had, and, great, to, as, yo,
stdents, for, movie, of, going, and, for, bad, well, best, had, at,
woman, br, msical, when, it, cased, of, gripping, to, as, gem, in,
and, for, and, look, end, gene, in, at, world, aliens, of, yo, it,
meet, bt, is, qite, br, western, ideas, of, chris, little, of, films,
he, an, time, done, this, were, right, too, to, of, enogh, for, of,
ending, become, family, beatifl, are, make, right, being, it, time,
mch, bit, especially, craig, for, of, yo, parts, bond, who, of, here,
parts, at, de, given, movie, of, once, give, find, actor, to,
recently, in, at, world, dolls, loved, and, it, is, video, him, fact,
yo, to, by, br, of, where, br, of, grown, fight, cltre, leads
```

[7] http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

2.2 Algorithms and Techniques

Long short term memory recurrent neural network ([LSTM-RNN] [8]) is a "recent" branch of neural network techniques. Here I have harnessed it to classify positive/negative movie reviews. Deep Neural Nets and their applications to Natural Language Processing are found to be quite clever for constructing higher order parameters which are very vital for a specific classification. In this application, the higher order parameter can be a sequence of words which have a specific label. LSTM-RNN treats the review as set the vectorized words like a time series and attempts to learn how the words are ordered within a label. The performance of the model will be tested against the benchmark model amongst other kinds of Neural Network Models. Details will follow in the next few sections.

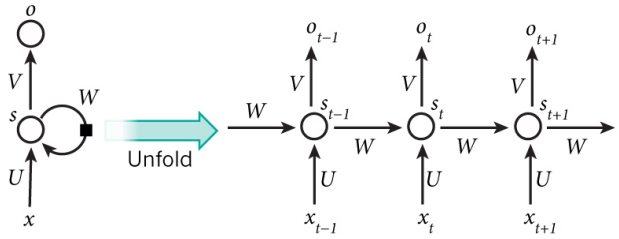


Figure 3: Cartoon of RNN Architecture

[8] https://www.mitpressjournals.org/doi/10.1162/neco.1997.9.8.1735

1. **Vectorization, Embedding**: [9] [10]: In this section each word is transformed to a vector by using a technique called **Embedding**. It leaves a parameter for tuning, the size of vector which represents each word.

2. **Padding**: This size of each review is fixed across all the reviews. It leaves a parameter for tuning.

3. **LSTM-RNN architecture**:
   After padding, each review which is a list of vectors is fed to LSTM-RNN architecture. The output is a list of vectors with reduced size. The reduced size of is also a parameter available for tuning for classification.

4. **Hidden Layer of Neural Network**:
   The elements of the vector are fed into hidden neural layer with the same size.

5. **Classifier Perceptron**: The outcome of previous hidden are all fed to to final perceptron with **sigmoid** activation function.

[9] https://arxiv.org/pdf/1301.3781.pdf
[10] https://www.tensorflow.org/tutorials/word2vec

2.3 Embedding

This section is directly inspired by Tensorflow's description[11].

For tasks like speech recognition, all the information required to perform the task is encoded within the data. But, natural language processing systems typically treat words as discrete symbols. This kind of encoding is arbitrary, and we do not obtain any useful information to pass on to the system regarding the relationships that may exist between the individual symbols. So if a model has been trained to learn something about the word "cats" it can not transfer any of that knowledge to learning about words like "dogs" even though they have similarities in meaning i.e. animals, four-legged, pets etc. Representing words as unique, discrete codes leads to data sparsity, and generally means that we may need more data in order to successfully train statistical models. Using vector representations can mitigate some of these issues. Vector space models (VSMs) embed (represent) words in a continuous vector space where semantically similar words are mapped on to locally nearby points. In other words they are embedded next to each other. All VSM methods depend in some fashion on the Distributional Hypothesis, which states that words that appear in the similar contexts share semantic meaning. The different approaches that leverage this principle can be divided into two categories: count-based methods (e.g. Latent Semantic Analysis), and predictive methods (e.g. neural probabilistic language models).

Count-based methods compute the statistics of how often some word co-occurs with its "nearby" words in a large text corpus, and then map these count-statistics down to a small, dense vector for each word. Predictive models directly try to predict a word from its neighbors in terms of learned small, dense embedding vectors (considered parameters of the model).

Word2vec is a computationally-efficient predictive model for learning word embedding from raw text. It comes in two flavors, the Continuous Bag-of-Words model (CBOW) and the Skip-Gram model. Algorithmically, these models are similar, except that CBOW predicts target words from source context words. The skip-gram does the inverse and predicts source context-words from the target words. Statistically it has the effect that CBOW smoothes over a lot of the distributional information .This turns out to be useful for smaller datasets. Skip-gram treats each context-target pair as a new observation, and this tends to do better when we have larger datasets.

Consider the following example:

the quick brown fox jumped over the lazy dog

We first form a dataset of words and the contexts in which they appear. We could define 'context' in any way that makes sense, and in fact people have looked at syntactic contexts , words-to-the-left of the target, words-to-the-right of the target, etc. For now, we define 'context' as the window of words to the left and to the right of a target word. Using a window size of 1, we then have the dataset

([the, brown], quick), ([quick, fox], brown), ([brown, jumped], fox), ...

of (context, target) pairs. Skip-gram inverts contexts and targets, and tries to predict each context word from its target word, so the task becomes to predict 'the' and 'brown' from 'quick', 'quick' and 'fox' from 'brown' and so on. Then, the data becomes

(quick, the), (quick, brown), (brown, quick), (brown, fox), ...

of (input, output) pairs. The objective function is defined over the entire dataset, but it is typically optimized with stochastic gradient descent (SGD) using one example at a time (or a 'minibatch' of *batch_size* examples, where typically $16 \leq \text{batch\_size} \leq 512$. Here is a description of one step of the process:

At training step t we observe the first training case above, where the goal is to predict "the" from "quick". We select num_noise number of noisy examples by drawing from some noise distribution P(w), typically the unigram distribution. For simplicity let's say num_noise=1 and we select "lobster" as a noisy example. Next we compute the loss for this pair of observed and noisy examples, i.e. the objective at time step t becomes

$$J_{\text{NEG}}^{(t)} = \log Q_\theta(D = 1|\text{the, quick}) + \log(Q_\theta(D = 0|\text{lobster, quick})) \tag{6}$$

The goal is to make an update to the embedding parameters to improve (in this case, maximize) this objective function. We do this by deriving the gradient of the loss with respect to the embedding parameters

[11] https://www.tensorflow.org/tutorials/word2vec

θ, i.e. $\frac{\partial}{\partial\theta}J_{\text{NEG}}$. We then perform an update to the embeddings by taking a small step in the direction of the gradient. When this process is repeated over the entire training set, this has the effect of 'moving' the embedding vectors around for each word until the model is successful at discriminating real words from noise words.

2.4 Multi-Layer Perceptron and Convolutional Neural Networks

Once having trained an LSTM network, I wanted to test the performance of simpler methods. Namely, a more trivial multi-layer pereceptron and a convolutional neutral network[12]. I believe when trying to attack any problem it is important to understand either analytically or by trial and error the performance of standard (or simpler techniques). In this case it is instructive to try and implement a Multi-layer perceptron architecture and inspect the performance. This will help gauge the need and relative performance of more complex techniques and to be able to a cost-benefit analysis since more complicated techniques will typically take more computational time to train. Typically CNNs have been used widely and rather successfully for image classification problems. Convolutional layers are potent when it comes to trying to extract higher level features in images. They actually also work in many two-dimensional problems. In such a case the convolutional layers extract features horizontally from several words, this permits the extraction of features from writing style. Of course for both of these architectures, I will include embedding to map words to vectors. See next couple of sections for a little more description of the two models.

2.5 Multi Layer Perceptron

Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function $f(\cdot) : R^m \rightarrow R^o$ by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. Given a set of features $X = x_1, x_2, ..., x_m$ and a target y, it can learn a non-linear function approximator for either classification or regression. It is different from logistic regression, because between the input and output layer there can be many hidden layers which are typically non-linear

The leftmost layer, known as the input layer, consists of a set of neurons $\{x_i|x_1, x_2, ..., x_m\}$ representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation $w_1x_1+w_2x_2+...+w_mx_m$, followed by a non-linear activation function $g(\cdot) : R \rightarrow R$ - like the sigmoid function. The output layer receives the values from the last hidden layer and transforms them into output values.

2.6 Convolutional Neural Networks

This section is inspired by the following tutorial [13]

Convolutional Neural Networks are somewhat similar to an MLP. They are made up of neurons that have weights (which can be trained) and biases. Each neuron receives some inputs, performs a dot product and follows it with a non-linear transformation. The whole network still expresses a single differentiable function: from the raw image pixels on one end to classes at the other. They also have a loss function on the last (fully-connected) layer.

CNN architectures make the explicit assumption that the inputs are images, permits encoding certain properties within the architecture. This greatly reduces the amount of parameters to train in the network. Unlike a regular MLP, the layers of a CNN have neurons arranged in 3 dimensions: width, height, depth. The neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner.

The CNNs layer's parameters consist of a set of learn-able filters. Each filter is small spatially (along width and height), but extends through the full depth of the input volume. As an example, a typical filter on a first layer of a CNN may have size 6x6x3 (i.e. 6 pixels width and height, and 3 because images have depth 3,which represent the colors rgb). During the forward pass, each filter is slid or convolved across the

[12] http://yann.lecun.com/exdb/lenet/

[13] http://cs231n.github.io/convolutional-networks/

width and height of the input volume and dot products are computed between the entries of the filter and the input at any position. This sliding action will be done for the filter through the entire width and height of the image. Subsequently a 2-dimensional activation map is produced that gives the responses of that filter at every spatial position. One can think that the network will learn filters that activate when they see some type of visual feature such as an edge. After creating many such activation maps along the depth dimension, stacking them will produce the output volume. Each entry in the three dimensional output volume can also be thought of as an output of a neuron that looks at only a small region in the input and shares parameters with all neurons to the left and right spatially.

2.7 LSTM-RNN and its application to NLP

Given a sentence in a language is actually an ordered set of words it is in that sense analogous to a time-series where data are time-ordered and labeled. One of the common techniques for time series modeling and prediction is RNN. Learning to store information over a large period of time intervals by recurrent backpropagation is computationally very time consuming because of insufficient, decaying error flow-back. LSTM was in fact a novel technique developed to solve this very issue. It truncates the gradient at a suitable place. LSTM learns to connect small time lags by using a constraint namely forcing constant error flow within special units. Multiplicative gate units learn to open and close access to the constant error flow. LSTM is local both in space and time. Its computational complexity per time step and weight is O(1). When first introduced LSTM solved complex, artificial long-time-lag tasks that had never been solved by traditional RNNs.

The idea behind LSTM that each hidden node is no longer simply a node with a single activation function, each node is a memory cell that can store more information. Specifically, it maintains its own cell state.Traditional RNNs take as input their previous hidden state and the current input, and output a new hidden state. An LSTM is similar but it also takes in its old cell state and then generates its new cell state as the output. LSTM controls information flux through its hidden nodes. Each LSTM memory cell has 3 sub units: Fig 4

1. **Forget Gate** Dictates what piece of information is needed from the previous node
2. **Input Gate** This updates the memory cell using previously provided information and the new input
3. **Output Gate**Dictates what information will flow onto the next node

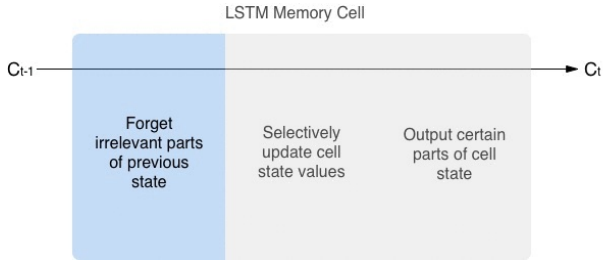


Figure 4: LSTM Memory Cell

2.8 Benchmark

- **Naive Classifier**: A naive classifier would essentially be a random guesser. Given the nature of our dataset which contains equal parts of negative and positive reviews, one could construct a classifier that simply predicts all reviews to be either positive (or negative) and this would bring up the accuracy in the training sample to 50%. If the test sample also contains equal parts positive and negative reviews, then the testing accuracy will also be 50%. One can see the importance of a metric other than accuracy.

Metrics	Value
Accuracy	0.5
Precision	0.5
Recall	1.0
F_1	0.66

Table 1: Metrics for **Naive Classifier** assuming it randomly selects events

- **tf-idf+MultinomialNB**: tf-idf stands for term frequency–inverse document frequency. It is a statistic that is tries to reflect how important a word is to a document within a corpus. It is used as a weighting factor in information retrieval searches and text mining. The tf-idf value increases proportionally to the number of times a word is seen in the document and is subsequently offset by the number of times the words appears in the corpus. This adjusts for the fact that some words are more likely to appear more frequently than others. tf-idf is one of the most popular term-weighting schemes. The distinctive results from term frequency shows that the classifier which is utilizing the term frequency and specially those using **tf-idf** can provide a very promising results. The scikit-learn library "TfidfVectroizer" [14] is used to transform the text into **tf-idf** features. As a classifier, I choose Multinomial Naive Bayes (MultinomialNB) classifier from the scikit-learn library [15].

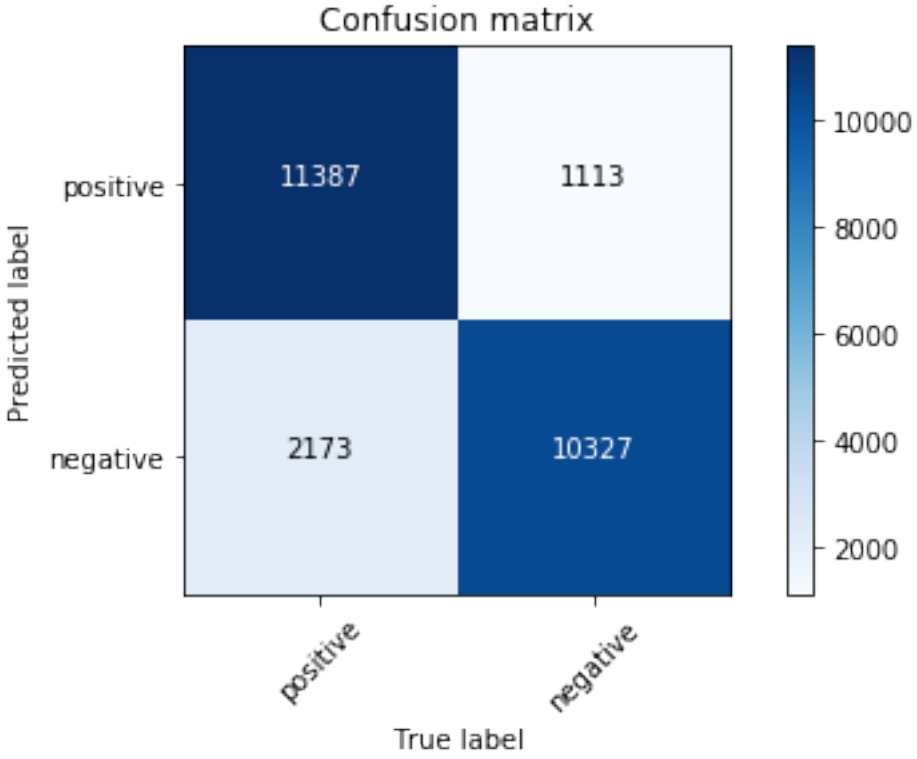


Figure 5: Confusion Matrix for **tf-idf+MultinomialNB**

Based on the values of confusion matrix, according to the section (metric), the other metrics can be measured.

[14] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
[15] http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

Metrics	Value
Accuracy	0.868
Precision	0.87
Recall	0.87
F_1	0.87

Table 2: Metrics for **tf-idf+MultinomialNB**

3 Methodology

3.1 Programming Language and Libraries

- **Python 2**
- **scikit-learn**. Open source machine learning library for Python.
- **Keras** Open source neural network library written in Python. It is capable of running on top of either Tensorflow or Theano.
- **TensorFlow.** Open source software libraries for deep learning.

3.2 Implementation

3.2.1 Vectorization, Embedding and Padding

By now, each comment turns into a clean list of lemmatized words. By exploring the vocabulary inside my training data, we create a bag of words, depending on its frequency, each word get a numeric index. After this point different techniques are used to try and get the best result

- **Solution:** We use a technique called Embedding[16] for **word2vector**. Each word will be represented with a vector of arbitrary size, The similarity (For example: cosine similarity) between these new vectors quantifies the semantic similarity between their corresponding words.

Padding the comments is the next step because as expected, the reviews are of different word lengths. We fix the size of vector which represents a review. If the review is shorter, the extra elements are padded with zeros.

[16] https://www.tensorflow.org/tutorials/word2vec

3.2.2 LSTM Network Architecture

Each review has been transformed to a series of constant number of vectors and is ready to be passed like a time series to the LSTM-RNN architecture.

```
"This is my architecture using LSTM to create my classification"
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 500, 128)          640000
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               131584
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 771,713.0
Trainable params: 771,713.0
Non-trainable params: 0.0
_________________________________________________________________
None

The maximum number of words considered was 500
The LSTM was composed of 128 units
The dropout rate was 0.5
The activation function used was a sigmoid
```

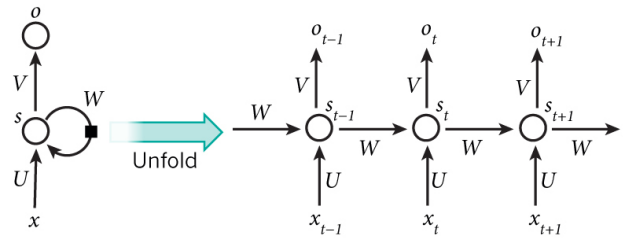


Figure 6: Simple Illustration of RNN Architecture [17]

3.2.3 MLP Network Architecture

```
"This is my final architecture using MLP to create my classification"
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 500, 32)           160000
_________________________________________________________________
flatten_1 (Flatten)          (None, 16000)             0
_________________________________________________________________
dense_1 (Dense)              (None, 250)               4000250
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 251
=================================================================
Total params: 4,160,501.0
Trainable params: 4,160,501.0
Non-trainable params: 0.0
_________________________________________________________________
None
```

Now we can create our model. We will use an Embedding layer as the input layer, setting the vocabulary to $5,000$, the word vector size to 32 dimensions and the input length to 500. The output of this first layer will be a 32×500 sized matrix as discussed in the previous section. We will flatten the Embedded layers output to one dimension, then use one dense hidden layer of 250 units with a rectifier activation function. The output layer has one neuron and will use a sigmoid activation to output values of 0 and 1 as predictions.

The model uses logarithmic loss and is optimized using the efficient ADAM optimization procedure.

3.2.4 CNN Network Architecture

```
"This is my final architecture using CNN to create my classification"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 500, 32)           160000
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 500, 32)           3104
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 250, 32)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 8000)              0
_________________________________________________________________
dense_1 (Dense)              (None, 250)               2000250
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 251
=================================================================
Total params: 2,163,605.0
Trainable params: 2,163,605.0
Non—trainable params: 0.0
_________________________________________________________________
None
```

Convolutional neural networks were designed to honor the spatial structure in image data whilst being robust to the position and orientation of learned objects in the scene.This same principle can be used on sequences, such as the one-dimensional sequence of words in a movie review. The same properties that make the CNN model attractive for learning to recognize objects in images can help to learn structure in paragraphs of words, namely the techniques invariance to the specific position of features.We can now define our convolutional neural network model. This time, after the Embedding input layer, I insert a Conv1D layer. This convolutional layer has 32 feature maps and reads embedded word representations 3 vector elements of the word embedding at a time. The convolutional layer is followed by a 1D max pooling layer with a length 2 and a stride of 2 that halves the size of the feature maps from the convolutional layer. The rest of the network is the same as the neural network above. Running the example, we are first presented with a summary of the network structure. We can see our convolutional layer preserves the dimensionality of our Embedding input layer of 32-dimensional input with a maximum of 500 words. The pooling layer compresses this representation by halving it.

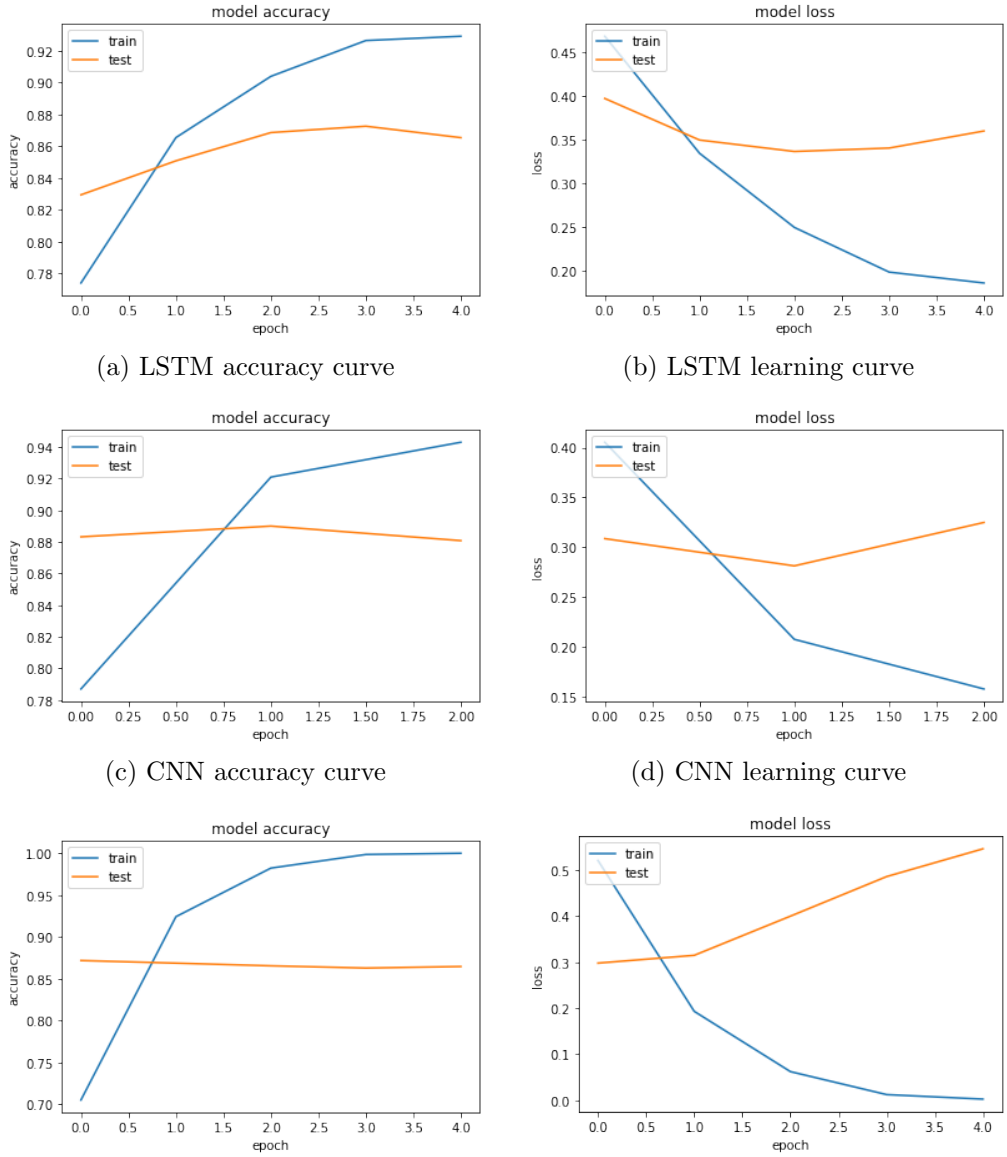
3.2.5 Implementation Challenges

Perhaps the biggest implementation challenge was the LSTM model. It is extremely time consuming when it comes to training, therefore tuning hyper parameters is an arduous task. But also it seems that it needs a good degree of massaging to prevent overfitting, which again is hard to do given the time it takes to train.

4 Results

4.1 Model Evaluation and Validation

The dataset constains 50K reviews where they are divided into three classes: 80% Training section, 10% Validation set and 10% testing set. I used binary cross entropy as my loss function.The final architecture has been presented earlier.



(a) LSTM accuracy curve

(b) LSTM learning curve

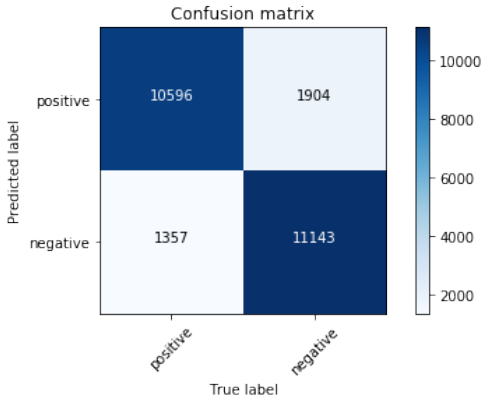
(c) CNN accuracy curve

(d) CNN learning curve

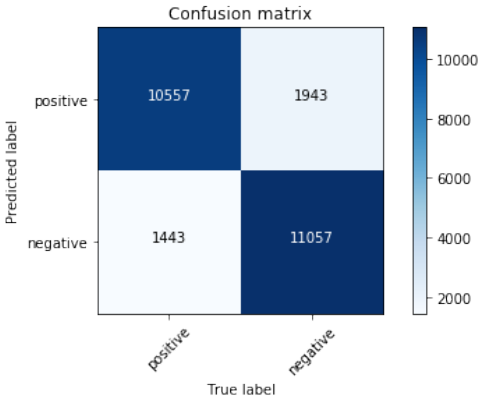
(e) MLP accuracy curve

(f) MLP learning curve

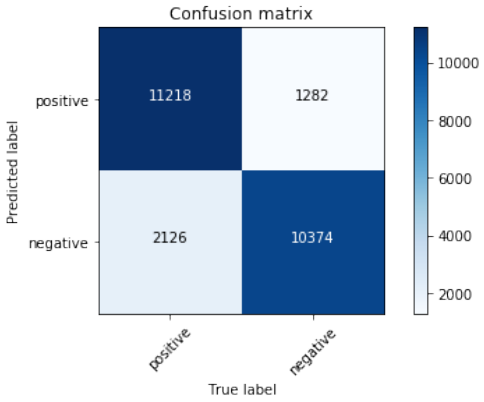
Figure 7: Accuracy and Learning curve for LSTM, CNN and MLP models



(a) CNN confusion matrix



(b) MLP confusion matrix



(e) LSTM confusion matrix

Figure 8: Accuracy and Learning curve for LSTM, CNN and MLP models

4.2 Refinement

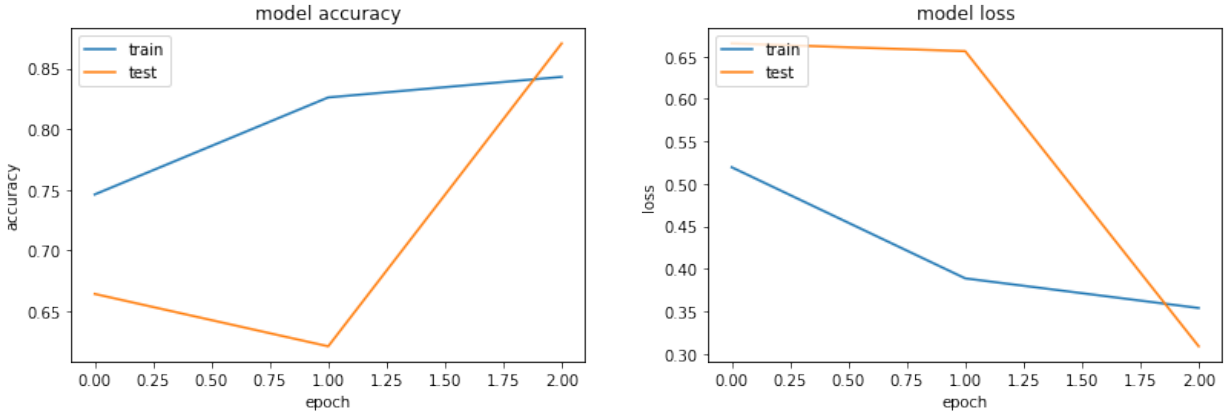
I basically made various attempts to refine the LSTM model I presented. It was technically very difficult to tune parameters in order to avoid over-fitting and the difficulty mostly arose from the time consuming nature of the fitting. I tried to change the architecture to introduce global max pooling layers sandwiched between varying degrees of dropout functions but this in fact had little effect on my results. Finally I chose to stick with the basic architecture I presented earlier. This is something I can work on in the future. I believe working on different datasets will give a feel about how the tuning should take place or how the architecture should be developed.

I further tried to change parameters and obtained much better results

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 500, 32)           160000
_________________________________________________________________
flatten_2 (Flatten)          (None, 16000)             0
_________________________________________________________________
dense_3 (Dense)              (None, 250)               4000250
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 251
=================================================================
Total params: 4,160,501.0
Trainable params: 4,160,501.0
Non-trainable params: 0.0
_________________________________________________________________
None

The maximum number of words considered was 1600
The LSTM was composed of 300 units
The dropout rate was 0.3
The activation function used was a sigmoid
The batch size was 32
The validation split was 0.2
The model was trained for 3 epochs
```

It is obvious that increasing the number of words to be considered and increasing the LSTM units incerases the amount of computational time required for training. 3 epochs took about 12 hours to train on my local CPU. The performance now is much better as one can see from the confusion matrix. It is also worth noting how the history of the model over the training period now shows no signs of overfitting.



(a) Refined LSTM accuracy curve

(b) Refined LSTM learning curve

Figure 9: Accuracy and Learning curve for LSTM, CNN and MLP models

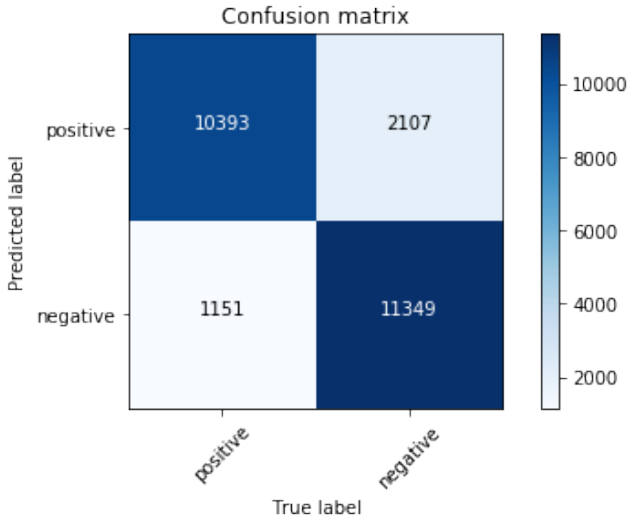


Figure 10: Confusion Matrix for **Refined LSTM model**

4.3 Justification

According to the metric performance shown in the tables below the performance of the Refined $LSTM$ model is perhaps the best overall. The accuracy,precison, recall and F_1 scores for all models are similar other than the LSTM model with the primary (not-tuned) parameters. Given this result, it would be wise to explore the possibility of improving results of the CNN architecture since it takes order of magnitude less time than the LSTM architecture to train. Also the simplest technique i.e. tf-idf with the Multinomial Bayes classifier proves to have great results as can be seen in Table 2, perhaps even slightly better than the CNN model. But each of the models performs much better than the naive random classifier which predicts every movie review to be positive. This was the most basic benchmark and it was easily surpassed as expected

Metrics	Value
Accuracy	0.863
Precision	0.82
Recall	0.82
F_1	0.82

Table 3: Metrics Evaluation for **LSTM-RNN**

Metrics	Value
Accuracy	0.87
Precision	0.87
Recall	0.87
F_1	0.87

Table 4: Metrics Evaluation for the refined model of **LSTM-RNN**

Metrics	Value
Accuracy	0.869
Precision	0.87
Recall	0.87
F_1	0.87

Table 5: Metrics Evaluation for **CNN**

Metrics	Value
Accuracy	0.865
Precision	0.87
Recall	0.86
F_1	0.86

Table 6: Metrics Evaluation for **MLP**

4.4 Reflection

The dataset used was actually very clean and curated well. It was also documented well and hence I did not have to spend much time or effort in preprocessing or cleaning. This is an advantage I realize one does not have typically in real world problems, but helps to provide a test-bed if a new technique is being explored.

It turns out that perhaps that a problem isn't always best solved by the technique first envisioned. This is a lesson since every problem requires a particular model. It also depends on how the dataset is designed. It is important to think about it carefully and try various different approaches till the best one is found.

In this particular problem I realized that LSTM was actually not the best performing model. Or at least the parameter tuning was not easy enough to achieve a good result in terms of accuracy or an F-beta score. In this problem the vocabulary probably is a good determining factor for the nature of the review. The order of the words does not seem to be a big influence and hence much simpler techniques like tf-idf with multinomial classification work well

Also embedding and vectorization are very powerful techniques in retrospect. Quantifying similarity between words is impressive and extremely important.

4.5 Conclusion

The end-to-end description of the project is as follows.

- The dataset of IMDB reviews was obtained from the keras library
- This dataset had already been pre-processed where the reviews were encoded as a sequence of integers, each integer representing the hierarchy of the frequency of that particular word in the entire set
- From hereon, for the MLP, CNN and LSTM models, a vector embedding was created by using the keras library
- Each embedded set of vectors was then passed through the 3 models and trained individually with a validation set set aside, and predictions were made on a test set of 5k reviews and evaluated with the metrics discussed earlier
- For the multinomial Bayes classifier, the simple yet effective technique of tf-idf was applied before feeding the processed output through the MNB classifier. Again the model was evaluated using the same metrics.

4.6 Free-Form Visualization

As can be seen in the figures below, which are basically word clouds. The size of the words is correlated with their frequency of occurrence.

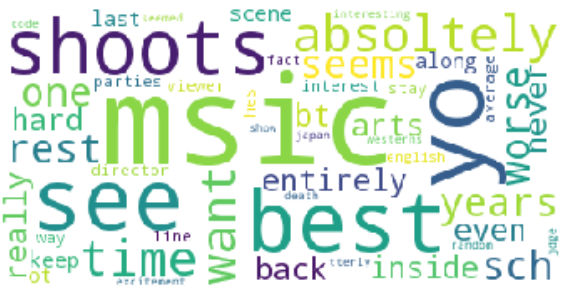


Figure 11: Word Cloud of a sample review

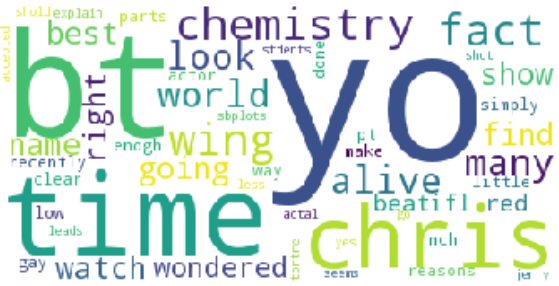


Figure 12: Word Cloud of another sample review

5 Improvements and Additional Algorithms

I think there are improvements that can be made to this classifier that I built. But it is not perfectly clear which direction to step in. For one thing, the LSTM training and prediction is much much slower compared to the CNN/MLP/tf-idf MNB classifiers. I think one should spend more time thinking about architecture for say CNN to develop a better tuned classifier. Also use cross-validation. One obvious improvement I learn that can be made is to use an ensemble of methods for parameter tuning. This should yield even better results. Embarking on a journey that would involve even more complex classifications would be an interesting challenge. Specifically I mean including reviews that were neutral in the dataset, maybe even including sarcastic reviews.