Prediction of the positive and negative reviews on Yelp and Amazon data sets via LSTM, CNN LSTM and SVM classifiers

Karina Karapetyan (596226)

June 10, 2018

Abstract

Text analysis and classification has been growing as s field for study in the recent years. It is critical that adequate techniques are used to evaluate and study content. Different techniques and architectures from machine learning are utilized to determine which has the best accuracy, and compare pros an cons of said techniques, when performing classification tasks on text reviews that can be labeled as positive or negative. A data set with over 5 million reviews is employed and the experiments are performed with 3 different models. A comparison and discussion is made between gained results and results from other recent studies made in the same field and using similar techniques.

1 Introduction

Classification is used to predict the class or label of data entries. Specifically using supervised learning, a model is created by learning the patterns of a data set and afterwards the label of new data samples can be predicted[10]. During classification, data set is divided into two subsets. One contains the features, which describe each entry of data. Meanwhile the second one consists of the labels for the instances[10]. Concretely, Yelp reviews are classified into two categories, positive and negative. This is a binary classification problem, the classifier should find a boundary that divides the data set according to the two values that the label can take.

I use Long Short Term Memory (LSTM) neural networks to make the classifier, given the results these networks have while performing sequence processing tasks. LSTM architecture utilizes memory cells to store data, usually it is better at storing and accessing information [7] in comparison to standard recurrent neural networks. The data set I am using consists of 5,200,000 reviews from the Yelp open data set, each instance contains its respective ids, votes which are three categories (funny, useful and cool), stars and text review. The features extracted from the text will be used,

and stars as label to determine whether a review is positive or negative. The purpose of this project is to prove that LSTM architectures have a better accuracy compared to a Support Vector Classifier when classifying text into binary classes. But LSTM take longer to train and have an overall slower performance.

2 Related work

There has been several studies on text analysis. Specifically social media has been the subject of analysis over the recent years. One example is Deep Health Care Text Classification published in October 2017 where a data set consisting of tweets was analyzed in two different tasks. One of them was to classify Twitter posts depending on whether they mentioned adverse drug reactions or not. The next task aimed to classify tweets which commented about personal medicine intake, possible medication intake or non-intake[**?**]. This was implemented with two approaches, a Recurrent Neural Network and a Long Short Term Memory Neural Network. Experiments were conducted using back-propagation through time, due to the limited size of the data set performance was low.[16]

There is a large amount of work done about sentiment analysis on text, for instance Measuring, Understanding and Classifying News Media Sympathy on Twitter after Crisis Events, published in January 2018, aimed to answer questions regarding the different reactions Western and Arab media had regarding the 2015 Beirut and Paris attacks[5]. There was preprocessing work done on the data set like term filtering, near-duplicates removal, language identification, among other techniques. Then the problem was treated as a binary classification problem, and the model aimed to classify a tweet as sympathetic or unsympathetic. This was accomplished by training a Convolutional Neural Network (CNN) with a single channel and three layers. The overall balanced accuracy was 0.725.[5]

3 Method

I have used a Support Vector Machine (SVM) as a baseline model for the experiments. A SVM is used to perform distribution-free learning to linearly separate the two classes in the underlying problem. I have utilized a Support Vector Classifier with a linear kernel[14]. Reviews are transformed into large sparse vectors, and the classifier is trained two times. In SVMs there is some unknown and nonlinear dependency between an input vector and a scalar ouput, however there is no information about the joint probability functions[14].

In linearly separable data I are aimed to find a hyperplane that minimizes the training error or empirical risk. During training the machine finds a

vector of weights and a bias, with which I find a hyperplane. SVMs are nonparametric models, which means the parameters are picked to match the model capacity to data complexity. SVMs keep the value of the training error fixed and minimize the confidence interval while RNNs one has to choose an appropriate structure for the model, and keeping the estimation error fixed, minimize the training error.[14].

LSTM architecture works similarly to Recurrent Neural Networks (RNN) where an input vector is passed through weighted connections to N recurrently connected hidden layers that compute the first hidden vector sequences and output vector. Each output vector is used to parametrize a predictive distribution over the next inputs[7]. In this architecture information passes vertically and horizontally and will be affected by multiple weight matrices and nonlinearities (**Appendix, Figure 3**). **Figure 3** illustrates that the inputs are 'skipping connections' to all hidden layers, and from hidden layers to the outputs.

These make it easier to train deep networks by reducing the number of processing steps and migrating the 'vanishing gradient problem'[7]. The hidden layer activations are calculated by using the following equations from $t = 1$ to T and from $n = 2$ to N:

$$h_t^1 = H(W_{ih^1} x_t + W_{h^1 h^1} h_{t-1}^1 + b_h^1) \tag{1}$$

$$h_t^n = H(W_{ih^n} x_t + W_{h^1 h^1} h_{t-1}^1 + b_h^1) \tag{2}$$

Where the W denotes the weight matrices, the b denotes the bias vectors and H is the hidden layer function. LSTM instead of using the RNNs hidden function, uses memory cells to store information (**Appendix, Figure 4**), so the hidden layer is implemented by the following composite function[7].

$$i_t = \sigma(W_{xi} x_t + W_{hi} h_{t-1} + W_{ci} c_{t-1} + b_i) \tag{3}$$

$$f_t = \sigma(W_{xf} x_t + W_{hf} h_{t-1} + W_{cf} c_{t-1} + b_f) \tag{4}$$

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc} x_t + W_{hc} h_{t-1} + b_c) \tag{5}$$

$$o_t = \sigma(W_{xo} x_t + W_{ho} h_{t-1} + W_{co} ct - 1 + b_o) \tag{6}$$

$$h_t = o_t \tanh(c_t) \tag{7}$$

Where σ is the logistic sigmoid function and *i, f, o* and *c* are the *input gate, forget gate, output gate, cell* and *cell input* respectively, all of which are the same size as the hidden vector *h*. In the model I used 150 as the size of the word embeddings, and a 20% of dropout to create a more generalizable model.

4 Data

In the project I have utilized **Yelp** data set that is a set of reviews, businesses and user information collected for educational, commercial and individual objectives [11]. Yelp clients provide a feedback on businesses and services presented on Yelp in order for other Yelp clients to evaluate the rating and make a sustainable choice [9]. The feedback is beneficial for the general experience, even though it does not pass the scope that led a user to the corresponding experience [9].
For instance, consider a Yelp review about a hair salon with 5 stars: [11]
"Best place to get your haircut in town. I recently moved here from North Carolina and was looking for a good barber shop and this spot definitely exceeded my expectations. Just 100% *good guys who know how to cut hair. This will be the only spot I go to get my haircut for as long as I live in Scottsdale. You guys are the best, thanks.".*
Overall, the data set contains 4736897 records with 9 different properties where there are over 1,100,000 tips gained from 1,300,000 users [11]. The data set is accessible in JSON and SQL formats, in the project I have employed *review.json* file with the size 3.82 GB [11]. A general feedback contains a data regarding a user with *user_id* leaving a rating for the business with *business_id* where the total review is depicted with next properties [11]:

- "$review\_id$" : "$gnWl\_LB9dkQol4LaB4oKPK$" - a 22-character string that represents a review id,
- "$user_id$" : "$Na6lPq24HgqgAk-pGHo_3l$" - a 22-character string that represents a user id which connects the user id with the user information in the $user.json$ file,
- "$business\_id$" : "$jrpsBo4Qg6LpQTQASldVVh$" - a 22-character string that represents a businness id which connects the business id with the business data in the $business.json$ file,
- "$stars$" : 5 - an integer that quantifies star rating,
- "$date$" : "$2016-03-09$" - string that contains the date when review was left in *YYYY-MM-DD* format,
- "$text$" : "Great place to visit after work!", - a string with the text review,
- "$useful$" : 2 - an integer illustrating an amount of gained useful votes,
- "$funny$" : 1 - an integer demonstrating a number of obtained funny votes,

- "*cool*" : 3 - an integer deriving a number of received cool votes.

The *review data set* is structured with reviews in form of JSON objects where each object is allocated per line. In order to process the data, I have stored a JSON array into a Python list. **Figure 3** introduces the descriptive statistics of the data set that encapsulates the focal tendency, shape and dissipation of the set distribution (**Appendix, Figure 3**). **Figure 4** demonstrates the obtained histogram for the data set, depicting the frequencies of the class occurrences (star ratings) (**Appendix, Figure 4**). From the **Figure 4** we can make a conclusion that the majority class is 5 stars whereas the most uncommon rating is 2 stars.
Next step was to classify the ratings into *positive* and *negative* reviews. I have achieved that by converting the **reviews with 1-3 stars** into negative feedback (**class 0**) and **reviews with 4-5 stars** - *positive feedback* (**class 1**).
Our goal was to teach a classifier to distinguish positive and negative reviews, based only on the feedback text. It is generally considered that the classifiers lead to perform worse on the minority class rather that on the the majority class. The classification "regulations" that forecast the minority class are more likely to obtain a much higher error rate than those which predict the majority class. The test samples that belong to the minority class are more often to be miscategorized rather the ones, belonging to the majority class. [6] Thus, I have implemented a balanced training data set with the equal amount of positive and negative reviews with an eye to fairly train the neural networks where I have maintained 200000 samples with positive feedback as well as 200000 samples with negative feedback.
Further, I have performed tokenization and sequence padding on the reviews in the training set, utilizing Keras library. Keras Tokenizer API vectorizes the texts into sequences regarding the selected amount of common words where the most frequent word in the list has rank 1, second most common word belongs to rank 2, etc. [4]. To begin with, we select the amount of common words and determine the word frequencies in the training set, assigning corresponding ranks. Hence, we convert the text reviews into a list of numerical tokens.
For instance, consider the list of reviews: *text = ["The most common word", "So is", "the least common", "oxymoron is common word"].* After carrying out tokenization, we obtain next results: *'common': 1, 'the': 2, 'word': 3, 'is': 4, 'most': 5, 'so': 6, 'least': 7, 'oxymoron': 8.* The most common word is **"common"** with a rank 1 whereas the least common word is **"oxymoron"** with rank 9.
Eventually, I have implemented zero-padding of the sequences where all training samples had the same amount of words to train the neural networks efficiently. The size of all texts was converted to the size of the longest text via filling with zeros [3]

For the training data set, I have employed the most common 40000 words whereas the selected maximum length for padding of the sequences was 500. **The size of the training set is 200000 reviews**, **the size of validation set includes the same amount of samples as the training set**. Regarding the test set, I have selected the **Grocery and Gourmet Food** data set that contains reviews from **Amazon** according to the selected topic. Overall, **the test data set includes 151254 samples**. I have categorized the reviews in the test data set as **positive** and **negative** in the same manner as I have done with the Yelp **review** data set. *Overall, I have obtained that in the test data set there are **120044 positive reviews** and **31210 negative reviews**.*

5 Experiments

The goal of the project is to design and train Long Short-Term Memory neural network, CNN Long Short-Term Memory model and Support Vector Machine to distinguish positive and negative reviews and compare obtained evaluation metrics as well as time performance.
Figure 5 demonstrates implementation of the Long Short-Term Memory neural network for the Yelp review data set (**Appendix, Figure 5**). To start with, I have settled a blank Sequential model to further supply it with diverse layers of the neural network. The constructed LSTM has the following structure where layers are indicated in the order of their implantation:

- **Embedding layer** is the first hidden layer of the LSTM network that inquires an embedding for all words in the text reviews of the training set [12]. Next parameters were specified for the layer:[12]

  1. **input_dim** - the vocabulary scale in the text feedback.
  2. **output_dim** - the scope of the vector space where the words are interposed.
  3. **input_length** - the length of the input textual feedback sequences.

  For the LSTM model I have selected the *vocabulary* with the size of **40000** where the words are encrypted with numerical values from 0 to 39999, inclusively; a *vector space* with **150** dimensions where the words are imprinted; the *input textual reviews* with length of **500** words each. The outcome of the Embedding layer is represented by a two-dimensional vector where for every word there is a single embedding in the input words sequence [12].

- **LSTM layer** takes as a first argument the *scope of the word embeddings* which is defined by the second parameter in the first hidden layer

of the neural network, in our case it is **150**. Next, I have specified a *dropout term* that represents a regularization method, commonly applied in the neural networks to impede overfitting [1]. I have implied **20%** probability of dropout where one fifth of all weights in the LSTM layer is reset with each round. Hence, it prevents the neural network from learning the recurring patterns in the data that can lead to deriving much robust network, due the regulations of the network being more generalized.

- **Dense layer** is the most plain layer of the neural network that returns the total sum of the activations from the preceding layer [8]. I assign the *size of the attained outcome* of the Dense layer to **1**, since I obtain the output with a value either 0 or 1 where I define 0 as a negative feedback and 1 as a positive review. In addition, the applied *activation function* was chosen to be **sigmoid**.

Finally, I have compiled the described LSTM model to further be executed on TensorFlow. Since I obtain only positive (class 1) or negative (class 0) reviews, I have employed **binary cross-entropy loss** that is a valid choice of the objective loss function for the binary labels prediction. Assume that our LSTM model predicts m while the target object is r, the logarithmic loss is determined as: [8]

$$-r \times log(m) - (1 - r) \times log(1 - m) \tag{8}$$

Moreover, I have utilized **Adam optimizer** that was applied instead of traditional stochastic gradient descent to update the weights in the iterative manner in the training data set. In addition, I have indicated the desired evaluation metric - **accuracy** that estimates the ratio of the correct classification predictions.

Figure 6 demonstrates the code for training the LSTM network where I pass as parameters the processed textual feedback (after maintaining tokenization and zero-padding), designated class labels. The parameter **validation_split=0.5** divides the data set into half to obtain equal-sized training and validation sets. Thus, the neural network determines the patterns in the tokens, denoted with positive and negative labels, in the training set [15]. Onwards, the network predicts the class reference for the reviews in the validation set without knowing the real labels with an eye to make comparison among the carried out predictions and actual labels to identify how well it performed with the learned patterns [15]. Finally, I set the parameter **epochs=3**, therefore the neural network processes training data three times.

In order to speed up the LSTM network, I have added the **Convolutional layer** to the network. The LSTM with CNN layer provides better time performance on the training set, due to various filters being computed independently [13]. On the other hand, it is complicated to parallelize the

LSTM networks, since every computation step is affected by the previous steps [13].

Figure 7 displays the implementation of the CNN LSTM (**Appendix, Figure 7**). LSTM network with CNN layer operates with the fragments of sequences rather than series of words. The structure of the CNN LSTM slightly distinguishes from the plain LSTM neural network. The **Dropout layer** is appended after the **Embedding layer**. Furthermore, a **Convolutional layer** is applied that carries out a filter over textual reviews to learn certain windows [2]. Next, a **MaxPooling layer** is derived that gathers the diverse fragmented representations into a uniform fragment [2]. The training stage is similar to the LSTM network (**Appendix, Figure 6**), even though the deployed training time considerably improved.

In the end I have implemented a **Support Vector Machine** classifier (**Appendix, Figure 8**). In the LSTM neural network I have used the tokenization and zero-padding for transferring every word to a numerical number regarding its frequency whereas in the SVM I employed a *term-frequency inverse document frequency* (TF-IDF) to vectorize the data. In the vectorization procedure, the order of the words is not taken into account. Every review is depicted as the immense spare matrix where every entry maintains the certain word and its frequency in the textual feedback. I have regularized the numerical representations by the overall frequency of the word occurring in that feedback to designate sparse words with greater importance comparing to the common words.

The SVM classifier took into account not only the separate words, but also all the combinations of the two words (**ngram_range = (1,2)**). Moreover, the classifier neglected the words that appeared in less than three diverse reviews via defining **min_dif = 3**. Therewith, I have selected an SVM with a *linear* type of kernel (**classify = LinearSVC()**). Next, I have transferred textual feedbacks into vectors via computing the vocabulary throughout the whole set of reviews and further converting every review into an immense sparse vector. Finally, a cross-validated score is obtained from the SVM classifier where I defined *the amount of training times* to be **5** (**cv=5**). In the same way as with the LSTM network, I have selected 200000 samples for the training set as well as 200000 samples for the validation set. Moreover, *n_jobs* parameter was set to **1**, therefore the classifier employs all accessible central processing unit cores.

Figures 9 and **10** illustrate the codes for implementing predictions via LSTM, CNN LSTM and SVM on the **Grocery and Gourmet Food** data set that contains the reviews on the corresponding subjects from **Amazon** (**Appendix, Figures 9-10**).

6 Results

Figure 11 demonstrates the obtained results from the performed experiments (**Appendix, Table 1**). A LSTM neural network attains the best test classification accuracy (**0.8149**) whereas a CNN LSTM network provides a slightly lower accuracy (**0.8062**), even though the training time significantly improved (**17013 s**), comparing to the LSTM's training time (**44427 s**). A SVM classifier has lowest accuracy value among other classifiers (**0.7612**), but it derives the best training time (**1701 s**). LSTM outperforms SVC regarding the accuracy parameter due to the capability of learning the long-term dependencies.
The LSTM with CNN layer maintains better time performance on the training set, owing to diverse filters being calculated independently [13]. On the other hand, it is complicated to parallelize the LSTM networks, since every computation step is dependent on the outcome of previous steps [13].
Figure 12 displays the derived confusion matrix for the LSTM network (**Appendix, Figure 12**). The classifier has made in total 151254 predictions where 48105 reviews were categorized as negative and 103149 reviews - positive. According to the actual classification, 31210 reviews are defined as negative and 120044 - positive. Overall, the CNN LSTM classifier correctly categorized 25662 negative reviews and 97601 positive reviews. The obtained total accuracy metric of the classifier is 0.8149.
Figure 13 illustrates gained confusion matrix for the CNN LSTM network (**Appendix, Figure 13**). The classifier has made in total 151254 predictions where 48842 reviews were categorized as negative and 102412 reviews - positive. According to the actual classification, 31210 reviews are defined as negative and 120044 - positive. Overall, the CNN LSTM classifier correctly categorized 25372 negative reviews and 96574 positive reviews. The obtained total accuracy metric of the classifier is 0.8062.
Figure 14 represents acquired confusion matrix for the SVM (**Appendix, Figure 14**). The classifier has made in total 151254 predictions where 60631 reviews were categorized as negative and 90623 reviews - positive. According to the actual classification, 31210 reviews are defined as negative and 120044 - positive. Overall, the SVM classifier correctly categorized 27864 negative reviews and 87277 positive reviews. The obtained total accuracy metric of the classifier is 0.7612.

7 Discussion

After running the experiments I have concluded that LSTM Neural Networks have better accuracy than SVM classifier, but the SVM classifier has the lowest training time among all. In comparison with the study on Clas-

sifying News Media Sympathy, this was accomplished by creating a CNN, they performed the text analysis in four different languages, and obtained an overall accuracy of 0.725, whilst the project on Health Care Text Analysis was done using both a RNN and a LSTM Neural Network, which got a precision of 0.414 and 0.843 respectively. I ran experiments using a SVM, LSTM and CNN LSTM, which resulted in 0.7613, 0.8149, 0.8062 respectively. In all experiments, the classifiers which had a LSTM architecture had a better accuracy in comparison to classifiers which used other architectures like standard CNN or SVM.

I have also found that although LSTM has a higher training time, adding a Convolutional layer helps decrease training time considerably without having a big impact on accuracy. LSTM seems to be the best option when analyzing text, in our case I have aimed to determine whether text was positive or negative, but research shows that even when the task is arguably more complex (analyzing whether text mentions self medication), LSTM provides good accuracy in exchange of long training time.

8 Conclusions

To begin with, the goal of the project was to teach various classifiers to differentiate positive and negative reviews of the **Yelp review data set** , based only on the feedback text, and compare gained evaluation metrics (accuracy) as well as time performance. In addition, I have classified reviews as positive in case the star rating was 4-5 and negative with star rating 1-3. I have employed 200000 reviews from **Yelp data set** for training as well as validation sets in order to maintain balanced learning for the neural networks.
Regarding the test set, I have selected the **Grocery and Gourmet Food** data set that included reviews from **Amazon** regarding the selected topic. Overall, **the test data set includes 151254 samples**. The reviews have been categorized in the test data set as **positive** and **negative** in the same manner as it has done with the Yelp **review** data set. *Eventually, I have obtained that in the test data set there are* ***120044 positive reviews*** *and* ***31210 negative reviews****.*
In the project I have implemented three types of diverse classifiers such as Long-Short-Term-Memory neural network, Long-Short-Term-Memory model with the Convolutional layer and Support Vector Machine classifier.
According to conducted experiments, the LSTM network outperformed SVM classifier, obtaining the accuracy of 0.8149 in contrast with the SVM's accuracy of 0.7612. The LSTM neural network provided the better accuracy result, due to the capability of learning long-term dependencies.
Moreover, we can make a conclusion that the worst training time was presented by LSTM network (44427 s). Adding a Convolutional layer to the

LSTM network drastically improved time performance (17013 s). The LSTM with CNN layer gains better time performance on the training set, owing to diverse filters being calculated independently [13]. In addition, it is difficult to parallelize the LSTM networks, since every computation step is dependent on the outcome of previous steps [13]. The SVM classifier provided best training time (1701 s).
In general, neural networks perform better on large data sets. In case of smaller data sets, a smaller neural network is more favorable, even though SVM might provide better classification performance, or regularization techniques are required to pretend the network from overfitting.

References

[1] Freeman D. Chio C. *Machine Learning and Security: Protecting Systems with Data and Algorithms*. O'Reilly Media Inc., February 2018. ISBN: 978-1-491-97990-7.

[2] Britz D. Understanding convolutional neural networks for nlp. November 2015.

[3] Dev D. *Deep Learning with Hadoop*. Packt Publishing Ltd., February 2017. ISBN: 978-1-78712-476-9.

[4] Keras Documentation. Text preprocessing. Technical report, November 2017. https://keras.io/preprocessing/text/.

[5] Abdallah El Ali, Tim C Stratmann, Souneil Park, Johannes Schöning, Wilko Heuten, and Susanne CJ Boll. Measuring, understanding, and classifying news media sympathy on twitter after crisis events. 2018.

[6] F. Provost G. Weiss. The effect of class distribution on classifier learning: An empirical study. Technical Report ML-TR-44, Rutgers University, 2001.

[7] Alex Graves. Generating sequences with recurrent neural networks. 2013.

[8] Pal S. Gulli A. *Deep Learning with Keras*. Packt Publishing Ltd., February 2017. ISBN: 978-1-78712-842-2.

[9] K. Kumar E. Gabrielova P. Choudary C. Lopes H. Sajnani, V. Saini. Classifying yelp reviews into relevant categories. Technical report, Donald Bren School of Information Computer Sciences, January 2018.

[10] F. Herrera, F. Charte, A.J. Rivera, and M.J. del Jesus. *Multilabel Classification*. Springer International Publishing, 2016.

[11] Yelp Incorporation. Yelp open dataset, November 2017. Available at: https://www.yelp.com/dataset.

[12] Brownlee J. How to use word embedding layers for deep learning with keras. October 2017. https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/.

[13] Zhang Y. Lei T., Artzi Y. Training rnns as fast as cnns. pages 1–8, 2018. Sixth International Conference on Learning Representations.

[14] Wang Lipo. *Support Vector Machines: Theory and Applications.* Springer International Publishing, 2005.

[15] Hearty J. Raschka S., Julian D. *Python: Deeper Insights into Machine Learning.* Packt Publishing Ltd., August 2016. ISBN: 978-1-78712-857-6.

[16] R. Vinayakumar, Barathi Ganesh H. B., M. Anand Kumar, and K. P. Soman. Deep health care text classification. *CoRR*, abs/1710.08396, 2017.

Appendix

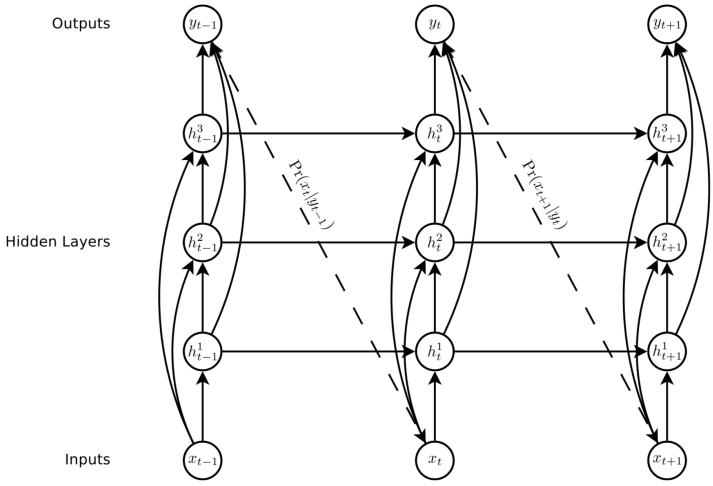

Figure 1: Deep recurrent neural network prediction architecture [7]

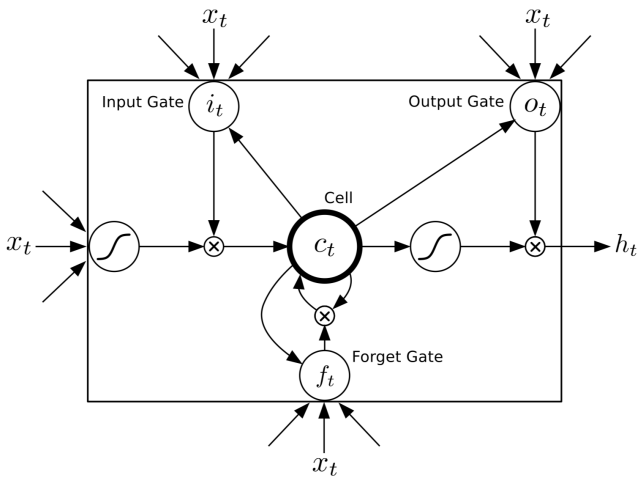

Figure 2: Long Short-term Memory Cell [7]

```
In [9]: df.describe().transpose()
Out[9]:
```

	count	mean	std	min	25%	50%	75%	max
cool	4736897.0	0.509660	1.960374	0.0	0.0	0.0	0.0	513.0
funny	4736897.0	0.402917	1.721954	0.0	0.0	0.0	0.0	631.0
stars	4736897.0	3.724048	1.421104	1.0	3.0	4.0	5.0	5.0
useful	4736897.0	0.988292	2.600021	0.0	0.0	0.0	1.0	1125.0

Figure 3: Descriptive statistics of the Yelp *review* data set

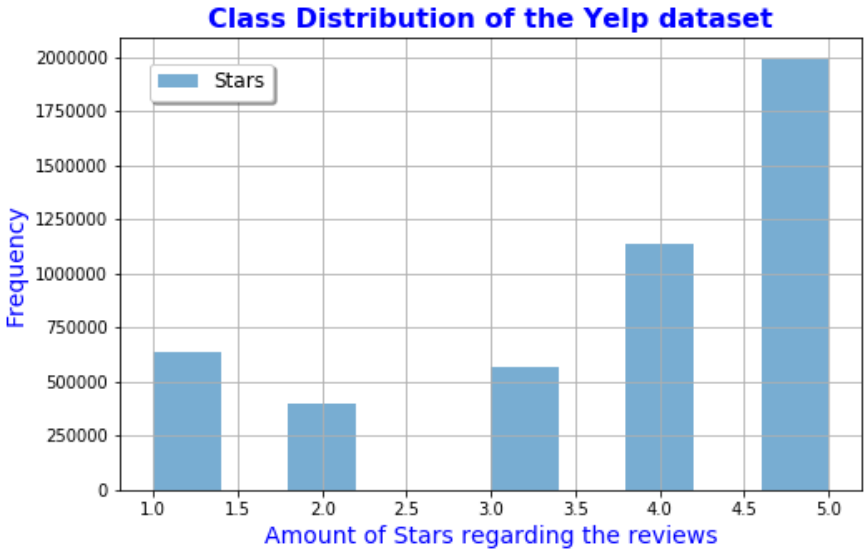

Figure 4: Histogram of the Yelp *review* data set

```
print("Carrying out LSTM model")
current = datetime.now()
model = Sequential()
model.add(Embedding(40000, 150, input_length=500))
model.add(LSTM(150, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', \
              metrics=['accuracy'])
print("Finished establishing LSTM", datetime.now() - current)
```

Figure 5: Developing LSTM neural network for the Yelp *review* data set

```
print("Training model")
current = datetime.now()
model.fit(data, np.array(norm_data), validation_split=0.5, epochs=3)
print(datetime.now() - current)
```

Figure 6: Training LSTM neural network for the Yelp *review* data set

```
print("Designing a model")
current = datetime.now()
model = Sequential()
model.add(Embedding(40000, 150, input_length=500))
model.add(Dropout(0.2))
model.add(Conv1D(75, 5, activation='relu'))
model.add(MaxPooling1D(pool_size=4))
model.add(LSTM(150))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', \
              metrics=['accuracy'])
print(datetime.now() - current)
```

Figure 7: Designing LSTM neural network with Convolutional layer for the

```
print("Vectorizer")
current = datetime.now()
vectorizer = TfidfVectorizer(ngram_range=(1,2), min_df=3)
classify = LinearSVC()
vect = vectorizer.fit_transform(norm_reviews)
print(datetime.now() - current)

print("Shape", vect.shape)

current=datetime.now()
res = cross_val_score(classify, vect, norm_data, cv=5, n_jobs=-1)
print(datetime.now() - current)
print("Score> ")
print(res)
```

Figure 8: Deriving SVM classifier for the Yelp *review* data set

```
with open('Grocery_and_Gourmet_Food.json', 'rb') as f:
    grocery = f.readlines()

gg = list(map(lambda x: x.rstrip(), grocery))
g = [json.loads(review) for review in gg]
gtexts = [tmp['reviewText'] for tmp in g]

# Convert Grocery_and_Gourmet_Food ratings into (negative or positive)
binary_ratings = [0 if tmp['overall'] <= 3 else 1 for tmp in g]
review_texts = []
true_labels = []
limit = len(gg)
count = [0, 0]
for i in range(len(gtexts)):
    polarity = binary_ratings[i]
    if count[polarity] < limit:
        review_texts.append(gtexts[i])
        true_labels.append(binary_ratings[i])
        count[polarity] += 1

test_data = gtexts
sequences = tokenizer.texts_to_sequences(test_data)
data = pad_sequences(sequences, maxlen=500)
predictions = model.predict(data)
'''
if prediction score <= 0.5 => assign sample to class 0
if prediction score > 0.5 => assign sample to class 1
'''

predictions[predictions>0.5]=1
predictions[predictions<=0.5]=0
print("Obtained prediction score", \
      sklearn.metrics.accuracy_score(true_labels, predictions))
```

Figure 9: Performing predictions on **Grocery and Gourmet Food** data set via **LSTM/CNN LSTM**, trained on the Yelp *review* data set

```
model = classifier.fit(out, balanced_labels)

with open('Grocery_and_Gourmet_Food.json', 'rb') as f:
    grocery = f.readlines()

gg = list(map(lambda x: x.rstrip(), grocery))
g = [json.loads(review) for review in gg]
gtexts = [tmp['reviewText'] for tmp in g]

# Convert Grocery_and_Gourmet_Food ratings into (negative or positive)
binary_ratings = [0 if tmp['overall'] <= 3 else 1 for tmp in g]
review_texts = []
true_labels = []
limit = len(gg)
count = [0, 0]
for i in range(len(gtexts)):
    polarity = binary_ratings[i]
    if count[polarity] < limit:
        review_texts.append(gtexts[i])
        true_labels.append(binary_ratings[i])
        count[polarity] += 1

test_data = gtexts

outt = vectorizer.transform(test_data)

predictions = model.predict(outt)
print("Obtained prediction score", \
      sklearn.metrics.accuracy_score(true_labels, predictions))
```

Figure 10: Performing predictions on **Grocery and Gourmet Food** data set via **SVM**, trained on the Yelp *review* data set

Classifier Type	Hyperparameters	Epochs/ Cross Validation	Training Loss/Accuracy	Validation Loss/Accuracy	Processing Time, s	Cross Validation Score	Test Accuracy
LSTM	Embedding layer - vocabulary size: 40000, vector space: 150 dimensions, input textual reviews: 500 words; LSTM layer: output dimensions: 150, dropout: 20%; Dense layer - hidden units: 1, activation: 'sigmoid'; Loss - binary cross entropy, Optimizer - adam, Evaluation metrics - accuracy	1/3	0.3099/0.8702	0.3437/0.8572	13563	-	**0.8149**
		2/3	0.2075/0.9169	0.2681/0.8865	17487	-	
		3/3	0.1662/0.9344	0.2872/0.8896	13377	-	
CNN LSTM	Embedding layer - vocabulary size: 40000, vector space: 150 dimensions, input textual reviews: 500 words; Dropout level - 20\% probability of dropout; 1D Convolutional layer - filters: 75, kernel size: 5, activation: 'relu'; 1D Max pooling layer - pool size: 4; LSTM layer - output dimensions:150; Dense layer - hidden units: 1, activation: 'sigmoid'; Loss - binary cross entropy, Optimizer - adam, Evaluation metrics - accuracy	1/3	0.2607/0.8918	0.3282/0.8648	5784	-	**0.8062**
		2/3	0.1841/0.9267	0.2728/0.8889	5708	-	
		3/3	0.1366/0.9470	0.3341/0.8864	5521	-	
SVM	classifier with a linear kernel (employing TFIDF vectorizor with ngram_range=(1,2), min_df=3)	5	-	-	1701	[0.8944, 0.8877, 0.8923, 0.9002, 0.8993]	**0.7612**

Figure 11: Training, validation and test outcomes on **Grocery and Gourmet Food** data set obtained by **LSTM, CNN LSTM** and **SVM** that were trained on the Yelp *review* data set

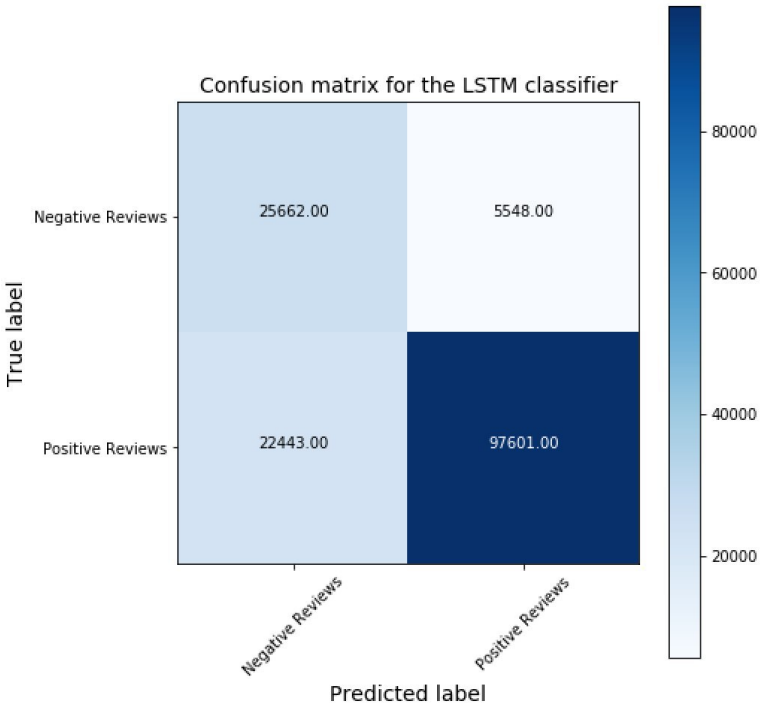

Figure 12: Confusion matrix obtained by **LSTM**, trained on the Yelp *review* data set, on **Grocery and Gourmet Food** data set

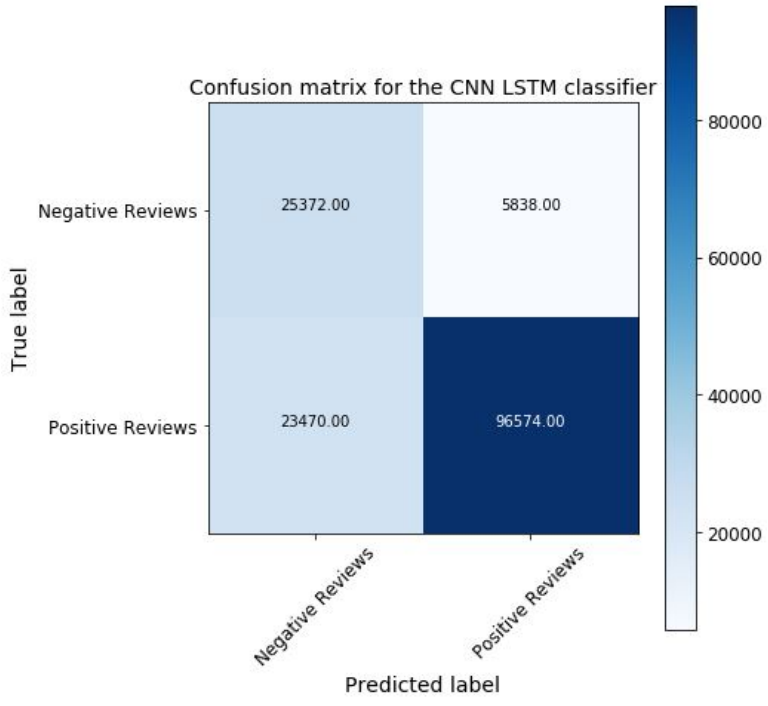

Figure 13: Confusion matrix obtained by **CNN LSTM**, trained on the Yelp *review* data set, on **Grocery and Gourmet Food** data set

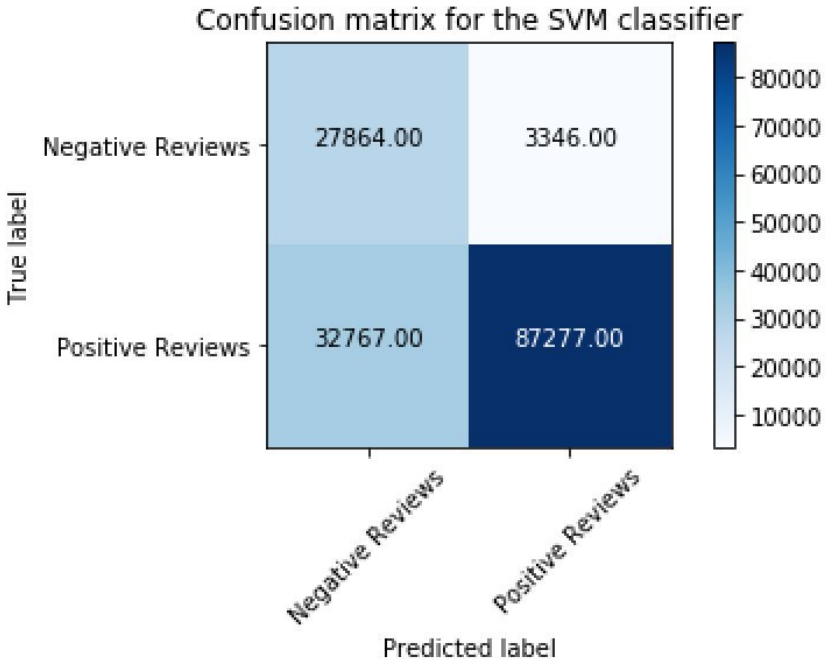

Figure 14: Confusion matrix obtained by **SVM**, trained on the Yelp *review* data set, on **Grocery and Gourmet Food** data set