Machine Learning Engineer Nanodegree

Capstone Project: Sentiment Analysis for Unlocked Phones Amazon Reviews

Silvia Onofrei

22 January 2019

DEFINITION

Project Overview.

The ubiquity of the social media in the everyday life led to a great expansion of the opinionated online data, which, in its turn, propelled the research in a field known as *sentiment analysis*. A subfield of Natural Language Processing (NLP) which is dedicated to the study and the processing of human language, sentiment analysis investigates people's opinions, attitudes towards products, services, organizations, issues, events, topics (see [1] for a comprehensive treatment of this subject).

In the present day, it is quite common for a customer to research online a product or a service before making a purchase. The reviews posted by the other customers play an essential role, they are often more relevant to the customer than the opinions of professional reviewers or advertisements. On the other side, as the online commerce becomes prevalent, more businesses follow up with their customers via online channels to request product reviews. As an Amazon customer I have noticed a sharp increase in such requests during the past months. Given the amount of data that is available from these reviews, it is desirable to develop automated methods to determine whether a certain review is favorable or not, or whether the review is relevant to other customers.

Sentiment analysis is usually formulated as a binary classification problem, in which the training and the testing data are product reviews which can be regarded as favorable (or positive) and unfavorable (or negative). The labels are provided by the reviewers themselves who use a 5 star rating system. It is common to regard the 1 and 2 star reviews as negative, the 4 and 5 star reviews as positive and the 3 star reviews as neutral. The methods used to perform text classifications involve supervised learning classifiers such as Naive Bayes or support vector machines(SVM) [2].

Due to the fact that "sentiment words are often the dominating factor for sentiment classification" [1], this problem can also be approached from an unsupervised learner point of view. In the past decade, deep learning techniques have been extensively used in NLP [3]; sentiment lexicons, grammatical analysis, or syntactic patterns can be exploited. In particular, neural networks (convolutional CNN and recurrent RNN networks, specifically) give very good results in sentiment analysis problems. In [4], Kim trains a CNN with one convolutional layer on the top of an embedding layer, which achieves excellent results on multiple benchmarks.

The questions that arise in natural language processing (such as text classification, language modeling, speech recognition, document summarization) are interesting, challenging, in actuality and originate in various areas such as news, finance, business, internet.

Problem Statement.

The primary goal of this report is to perform document level sentiment analysis on a corpus of reviews. In this context, each document (review) is treated as a basic information unit, and we have to automatically determine whether the review expresses an overall positive or an overall negative opinion. Since the sentiment is assumed to take binary values (0 for negative) and (1 for positive), this is a classification problem.

I plan to employ deep learning methods, and in particular to use Convolutional Neural Networks to perform this classification. The performance of the networks will be evaluated against the sentiment analyzer utility in NLKT and also against a benchmark model, which is chosen to be a Support Vector Machine Classifier.

The corpus of reviews is preprocessed following a standard schema that eliminates upper case letters, non-letter characters from the text and extracts the morphological root of a word. The data is split into training, validation and a testing sets. The next step is to use one of the word embedding methods, in which the reviews are transformed in vectors. First, I use Td-Idf to create vectors that are then fed to the SVM classifier. Secondly, I use Word2Vec, which is in fact a shallow neural network which will constitute the first (embedding) layer in the convolutional neural networks constructed here.

Metrics.

Justification. An initial evaluation indicates that the data is unevenly distributed, with about 40% negative reviews and 60% positive reviews. Consequently, the methods used to evaluate the performance of the model have to be chosen accordingly. It is well known that in the case of unbalanced data, the classification accuracy is not an appropriate metric to choose. In this project I first employed the cross-validation `StratifiedShuffleSplit` method from `sklearn` to create stratified randomized folds that preserve the ratios of samples for each class. In order to evaluate the performance of the model, I decided to compute the widely used precision, recall and their harmonic mean, the F_1 score. The goal is to attain a classifier which is exact, this means that I am aiming to attaining a high precision model. On the other side, I would also like to maintain a good sensitivity of the classifier. A better sensitivity often has a negative effect on the precision, and the aim is to balance these two. In such cases, it is common to use the harmonic mean of these scores, the F_1 score. Thus our goal is to obtain high F_1 scores. For the benchmark models, the AUC scores will also be computed.

Description of the metrics. First recall that predicted data can be divided into four categories: (TP) are the true positives, positive reviews, predicted as positive; (FP) are the false positives, negative reviews, predicted as positive; (TN) are the true negatives, negative reviews, predicted as negative; (FN) are the false negatives, positive reviews, predicted as negative.

The *accuracy* measures how often the classifier makes the correct prediction; it is computed as: $(TP+TN)/T$, where T is the total number of reviews. The *precision* is computed as: $TP/(TP+FP)$, gives the proportion of positive reviews that are classified as positive among all the reviews that are classified positive. The *recall* tells us what proportion of reviews that were actually positive, were classified as such, among the reviews that are actually positive ; it is computed as $TP/(TP+FN)$. The *F_1 score* is the weighted harmonic average of the precision and the recall scores: $F_1 = 2 \cdot \dfrac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$. This score can range from 0 to 1, with 1 being the best possible F_1 score.

Receiver operating characteristic (ROC) curve is a performance measurement for classification problems at various thresholds settings; it is plotted with recall on the y-axis and on the x-axis the false positive rate or $FP/(TN+FP)$. ROC is a probability curve and AUC (the area under the curve) is a measure of how much the model is capable of distinguishing between classes.

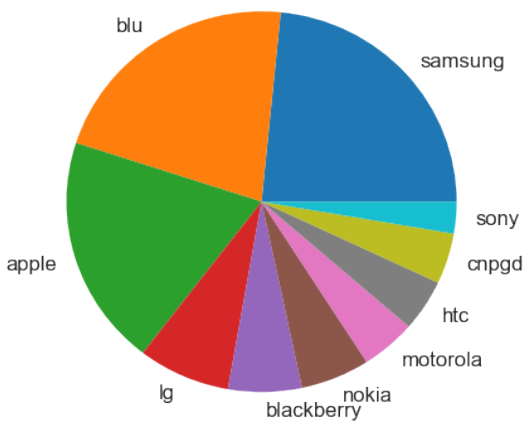
ANALYSIS

Data Exploration.

The data used in this project is from Kaggle public dataset repository: Amazon Reviews: Unlocked Mobile Phones. An initial analysis of the data can be accessed on KDnuggets.

- The dataset consists of $413,840$ reviews from Amazon's unlocked mobile phone category, extracted by PromptCloud in December 2016. It has 6 features, that are self-explanatory labeled: Product Name, Brand Name, Price, Rating, Review and Review Votes.
- There are 4410 distinct entries in 'Product Name', with no missing values. Not all the reviews are for mobile phones. Some text processing might be necessary for a deeper investigation, but not at this point.
- The 'Brand Name' has 384 distinct entries and $65,171$ missing values. A quick inspection of several entries in the brands list indicates that there are numerous inconsistencies, misspelled brands or incorrect information. Some of these can be updated and corrected using a list of available mobile phone brands, as the one from this site. Since this feature is not used in the sentiment analysis, it will be left as it is for now, but I will remove some of the brands that are clearly not cell phones makers.
- There are 1754 distinct 'Price' values and 5933 missing values in this column.
- The 'Rating' column contains the usual star rating, from 1 to 5, given in integer form.
- There are $162,491$ unique values in 'Reviews' column, which indicates that most of the reviews are single word, symbols or very short reviews. There are 62 entries with no review. I will remove these entries.
- The 'Review Votes' column has only $135,397$ non-zero entries, the remaining entries are either 0 or they are missing ($12,296$ entries). This feature will not be used in my analysis, and after changing it's type to integer, I will leave it as it is.

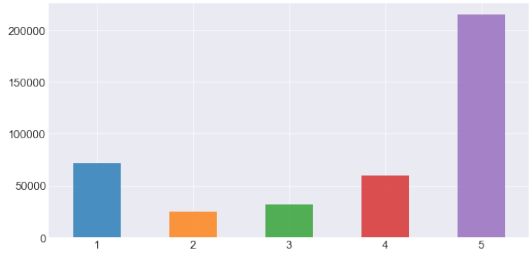
Exploratory Visualization.



Analysis by Brand. After rewriting all the brands in lower case and removing records for brands that is easy to determine they do not correspond to mobile phone makers, there are 306 brands left. The first 10 most numerous brands are given in the pie chart to the right. The list is dominated by the brands Samsung, Blu and Apple.

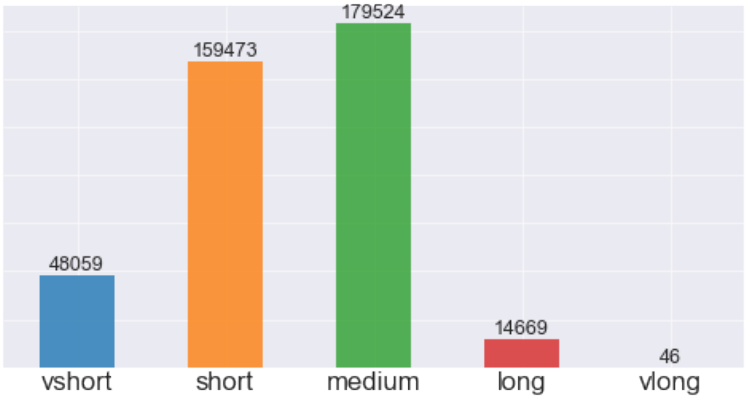
Analysis by Ratings. The data is highly unbalanced, most of the products receive 5 stars ratings. The next numerous group is given by the products that received 1 star ratings. The numbers are given below:

rating	1	2	3	4	5
count	72,350	24,728	31,765	61,392	223,606



Distribution by Ratings

Review Length Analysis. Here, the review length denotes the number of characters in a review, this includes the empty spaces. Basic descriptive statistics on the review lengths indicates that half of the reviews have length of at most 94 characters, the upper quartile consists of reviews of lengths 225 characters or more. There are some outliers in the data, very long reviews. The maximum length contains almost $30,000$ characters. It is easier to see how the length is distributed among the reviews if we create several bins on the review length as described below. It is worth mentioning that about one tenth of the reviews consist of 10 characters or less.



Distribution by Review Length

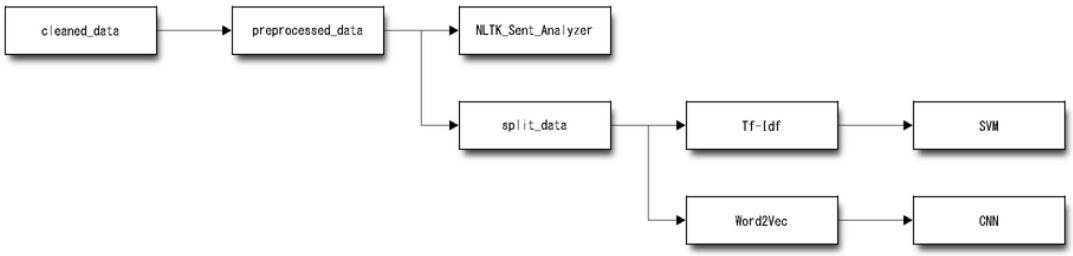
bin	length (L)
vshort	$L \leq 10$
short	$10 < L \leq 100$
medium	$100 < L \leq 1000$
long	$10^3 < L \leq 10^4$
vlong	$10^4 < L$

Here are some other facts concerning this dataset:

- Almost every single item in the set has a price, the prices range from about \$2 to \$2598. The most expensive brands are Iridium and MTM Trading LLC which are probably satellite phones.
- The average price does not vary too much between various star ratings. For example the average price for the 1 star ratings is \$203, while for 5 stars ratings is \$254.
- Most of the reviews receive 0 or 1 vote, there are only 3010 reviews that receive more than 30 votes. It seems that the number of votes is not a very good indicator for the average usefulness of of a review.
- There is no direct relation between the review length and the frequency of a brand.
- The most numerous brands have similar average review length, but this might be due to an averaging factor due to large numbers of occurrences.

Algorithms and Techniques.

The schema of the project is given in the following diagram:



The dataset is statistically and visually analyzed, a preliminary cleaning of the data is performed. The reviews corpus is then prepared according to standard techniques in text processing. The NLTK Sentiment Analyzer utility is used to perform a preliminary analysis. The data is then split in a pre-training set (of 80% of the data) and a testing set (20% of the data). The pre-training set is further split into a training set and a validation set (10% of the pre-training data). First, the preprocessed data is vectorized using Tf-Idf and then a supervised classifier, a support vector machine (SVM) is fit on the training set vectors and evaluated on the test set vectors. Secondly, the preprocessed training data is embedded using Word2Vec which is used as an embedding layer for a Convolutional Neural Network (CNN) which is evaluated on the test set.

Tf-Idf Vectorizer Method

Th-Idf is a frequency based embedding of the text into a real vector space.

We first process the reviews as described below. Next, we initialize the `TfidfVectorizer` from scikit-learn. We call `fit_transform` on the training set which first creates a vocabulary of words based on the text corpus given to it. Then it calculates the Tf-Idf for each term found in a review.

This results in a matrix, where the rows correspond to the individual reviews and the columns correspond to the words in the dictionary. Thus, every entry represents the Tf-Idf score of a word in a review.

Finally, we call `transform` on the test set, to calculate the Td-Idf scores for the test reviews, and using the same dictionary as the one created from the training data. Thus we obtain the corresponding matrix of Tf-Idf scores for the test reviews.

The Tf-Idf scores are computed as $\text{Tf} \cdot \text{Idf}$. Here Tf is the term frequency, i.e. the ratio of the count of a word present in the review to the length of the review. Then $\text{Idf} = \log(N/n)$ where N is the total number of reviews and n is the number of reviews where the word is present. The larger Idf value, the more rare is the word. Therefore, the Tf-Idf score penalizes very common words and gives higher weight to less common words, as they are more useful in determining the sentiment of a review.

Word2Vec

Word2Vec is another method of obtaining an embedding of the reviews texts into a real vector space, by mapping the words into prediction based vectors. Word2Vec was created by a team of researchers led by T. Mikolov at Google [6]. Word2Vec consists of a combination of two algorithms: CBOW (continuous bag of words) and SG (skip-gram). These models are shallow, neural networks with two layers. Both methods learn weights that act as word vector representations. Skip-Gram gives more accurate results while CBOW is faster. Intuitive descriptions of the both methods can be found in this blog.

The CBOW neural network predicts the probability of a word given a context. For one review text of length L, the input layer and the target layer are one-hot encoded arrays of size $1 \times L$ for each word. The input hidden weights are stored in a matrix of size $L \times N$, where N denotes the embedding dimension. Hence there are also N neurons in the hidden layer. The output weights are stored in a matrix of size $N \times L$. There is a no activation function between layers. The input vector is multiplied by the hidden weights matrix and it is called hidden activation. The hidden input weights are multiplied by the hidden output weights and the output is calculated. Error between output and target is calculated and propagated back to readjust the weights. The weights between the hidden layer and the output layer are giving the vector representation of the word.

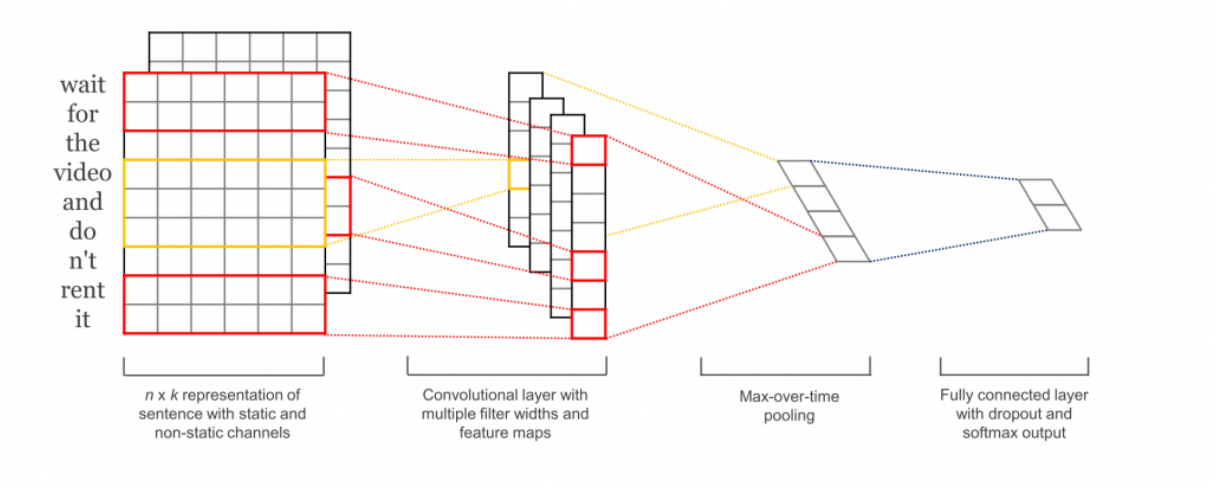
The Skip-Gram model has the same topology as CBOW model, but the architecture is flipped. The aim now is to predict the context given a word. There are two input words for each training example consisting of one input target word an input context word. Therefore there will also be two output vectors. Two separate errors are calculated with respect to the two target variables and the two error vectors obtained are added to obtain a final error vector which is propagated back to update the weights.

I will implement the Gensim Word2Vec utility. The text corpus is prepared so that each review is a pre-processed list of words. The following parameters are specified:

- *sentences* this is the list of lists of words;
- *window* is the maximum distance between the current and predicted word within a sentence;
- *workers* gives worker threads to train the model;
- *size* is the embedding dimension;
- *min_count* gives the minimum frequency of the words used.

Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a deep feed-forward artificial network. A CNN has filters of determined size s, or convolutions, that apply to certain neighborhoods of input values. This means that $s \times s$ input squares are mapped to a single output point. In a convolution operation, this $s \times s$ moving filter would shuffle across each possible combination of the input array. It follows that not every input node is connected to an output node (the connections are sparse). The output of the convolutional step is passed through a non-linear activation function. The simplest example of a CNN for NLP consists of an input layer (word embedding), a convolutional layer (a feature extraction layer from the embedding) and a fully connected layer (to interpret the extracted features in terms of a predictive output).



Model architecture with two channels for a sentence [4]

The CNN architectures used in this project follow the one given and detailed by Kim [4]. We are using only one channel in this project. The neural network consists of the following components.

The *embedding layer* stores a matrix to map the words represented by numeric indexes to their dense vector representations. The vectors have dimensionality given by the hyperparameter called embedding dimension. The embedding layer is obtained using Word2Vec representation.

The *deep network* consists mainly of several convolutional and pooling layers. It takes the set of embedding vectors to a compressed representation that captures the information contained in the text corpus. The convolutional layers have increasing number of kernel sizes, the number of filters also depend on the choice of the model. Each convolutional layer has ReLu activation map. Max pooling layers are added to reduce the dimensionality of the convolutional layers. Dropout layers are added to reduce over fitting.

Fully connected layers take the representation provided by the deep network and transforms it into the output form. This part contains dense layers, combined with batch normalization and dropout layers.

The *output layer* is also a fully connected (dense) layer which has a sigmoid activation function (for the text classification discussed here) and which gives the final classification output.

The CNN models constructed here use TensorFlow and Keras utilities.

Benchmark.

The results obtained with the deep learning models constructed in the project are compared against a couple of benchmark models. One is the sentiment analyzer utility in NLTK, the second is a supervised algorithm (specifically a support vector machine) trained on a Tf-Idf embedding of the text corpus.

NLTK Sentiment Analyzer

The Sentiment_Analyzer module provides tools to implement and facilitate sentiment analysis tasks using NLTK features and classifiers. I particular, I implemented VADER sentiment analysis tools [5]. This analyzer returns a float score between -1 and 1. The negative VADER scores are treated as negative reviews (0) and the nonnegative VADER scores are treated as positive reviews (1).

Support Vector Machine (SVM)

SVM is a supervised machine learning algorithm that is used in both classification and regression problems. The input data is a set of vectors in an n-dimensional real vector space (here n is the number of words in the reviews corpus). Each vector (corresponding to a review) is represented as a point in $\mathbb{R}^n$. A hyperplane that separates the vectors in two classes (positive and negative reviews) is found. SVM chooses the decision boundary that maximizes the distance from the nearest data points of all the classes, obtaining the most optimal decision boundary.

SVM are widely used for text and hypertext categorization. They are effective in high dimensional spaces, they are memory efficient and versatile. However, the model requires fine tuning to achieve the best performance and the choice of the best kernel (the formula for the classification surface) might be cumbersome.

In this project I implemented the SVM classifier with linear kernel from scikit-learn.

METHODOLOGY

Data Preprocessing.

Each review text is preprocessed according to the following scheme:

- convert all text to lowercase;
- remove special characters, punctuation signs and numerical characters; this is achieved using the **re** package designed to process regular expressions;
- remove stop words, low information common words in the English language; this step is achieved by removing all the words that do not appear in a list of 153 stop words available in **nltk.corpus** module;
- use **nltk.tokenize** module to tokenize, split each review as a Python list of words;
- use stemming, this is the process of reducing a word to its word stem by removing suffixes and prefixes; this is performed using SnowballStemmer from the NLTK library;
- the clean text is now saved as a Python list (for processing with Word2Vec) or a string (for processing with Td-Idf vectorizer).

The above steps are collected in a Python function denoted `clean_data`. For example, the first review in the corpus is in its initial form:

> *"I feel so LUCKY to have found this used (phone to us & not used hard at all), phone on line from someone who upgraded and sold this one. My Son liked his old one that finally fell apart after 2.5+ years and didn't want an upgrade!! Thank you Seller, we really appreciate it & your honesty re: said used phone.I recommend this seller very highly & would but from them again!!"*

After processing with the `clean_data` function, the review becomes a Python list:

> *['feel', 'lucki', 'found', 'use', 'phone', 'us', 'use', 'hard', 'phone', 'line', 'someon', 'upgrad', 'sold', 'one', 'son', 'like', 'old', 'one', 'final', 'fell', 'apart', 'year', 'want', 'upgrad', 'thank', 'seller', 'realli', 'appreci', 'honesti', 'said', 'use', 'phone', 'recommend', 'seller', 'high', 'would']*

Implementation.

Data Cleaning

• After an initial evaluation of the dataset, I decided to perform the following cleaning steps, to better prepare the data for text pre-processing:

◇ rename the columns, avoid blank spaces and upper case names:
'Product Name" is replaced with 'name', for example;

◇ change some of the column types;

◇ perform some basic cleaning of the product names, which resulted in deleting a number of records that do not correspond to mobile phones;

◇ remove all those entries that do not have reviews.

NLTK Sentiment Analyzer

• The Vader Sentiment Analyzer utility is fit on the entire pre-processed text corpus. The utility associates a score between -1 and 1 to every review. These results were categorized as positive (1) and negative(0) to conform with the rest of our analysis.

Data Split

• The text corpus consists of $n_w = 370,720$ reviews. These are split into a training set `X_train` with $n_r = 266,918$ reviews (70% of n_w), a validation set with $n_v = 29,658$ reviews and a testing set `X_test` with $n_t = 74,144$ reviews. Since the data is highly unbalanced we used the `StratifiedShuffleSplit()` utility in `sklearn`.

Text Preprocessing

• Each set is pre-processed according to the above steps. We obtain two lists of words.

Tf-Idf Vectorizer

• The `TfIdfVectorizer` is initialized. On the train data, the method `fit_transform` creates an embedding of shape $(n_r, 39,118)$ where the second number corresponds to the number of different words in the vocabulary. The `transform` method applied to the test data creates an $(n_t, 39,118)$ output.

SVM Classifier

• The `svm.SVC` classifier in sklearn is fit on the training data. A linear kernel is used. The input data for the trained classifier consists of the Tf-Idf embedded vectors. Various metrics are computed for this classifier.

Word2Vec Embedding

• Choose hyperparameter: `embedding_dim` to be 200.

• Each list is processed using `Word2Vec` in `Gensim` with the following parameters:

```
gensim.models.Word2Vec(sentences, window=4, workers=4,
          size = embedding_dim, min_count = 10)
```

The vocabulary size on the training set contains 8712 words. The embedding is then saved in a file.

• The `Word2Vec` embedding is processed to be used as input for the Neural Network. This is done first using the `Tokenizer()` object in NLTK and then replacing each sequence of words by a sequence on integers, which each corresponding to the word index in the vocabulary.

• The sequences of integer are padded to have the same length, equal to the embedding dimension. As a result the shape of the train matrix is $(n_r, 200)$, the validation matrix is $(n_v, 200)$ dimensional and the test matrix is $(n_t, 200)$ dimensional.

• Next we map the embeddings from the saved `Word2Vec` vocabulary to create a matrix of weights, which in our case has shape $(num\_words, embedding\_dim) = (35,463,200)$.

The First Neural Network Architecture

• The first standard step in Keras is to initiate a Sequential model.

• The embedding layer is added. It loads the embedding matrix which was previously built on the training data set. Since we set `trainable = False`, the weights will not be uploaded during training.

• There are three groups of layers, each consisting of a convolutional layer, a batch normalization layer and a pooling layer. The last two groups also contain droput layers.

• I add three 1D convolutional layers, with the goal of increasing the depth of the final layer. This is attained by choosing an increasing geometric sequence of filters $[32, 54, 128]$. These convolutional layers have same kernels of sizes $[3,4,5]$, the stride is left to the default value of 1. To avoid loss of information, the data is padded with zero entries (choose the padding hyperparameter as 'same').

• Reduce the covariance shift with a batch normalization layer between the convolutional layer and the pooling layer.

• To reduce the spatial dimensions of each layer (the width and the height) I pair each convolutional layer with a pooling layer. The choice of hyperparameters for each MaxPooling layer downsizes each spatial dimension by a factor of 2. This helps with over fitting and decreases the computational cost by reducing the number of parameters to learn.

• For extra regularization I employ Dropout, with a rate of 0.2, on the second and third convolutional layers.

• After flattening, the last layer is a Dense layer, with *sigmoid* activation function and L_2 regularization.

• The model is compiled, it computes the binary cross entropy as the loss function, uses the Adam optimizer and the metric is accuracy.

• The first model is fit over the training set from which 10% of the data is used for validation. The model is run over 4 epochs, using a batch size of 64. The accuracy and the test loss are evaluated as standard metrics. The precision, the recall and the F_1 score are defined as custom metrics and computed after each epoch.

COMMENTS. We focus on document sentiment analysis for a rather small dataset file, thus the review corpus was easy to preprocess. The `sklearn` classifiers and metrics are easy to implement and straightforward to use. The training of the SVM model was rather slow on my personal computer. The `TensorFlow` package is easy to employ, the deep learning models can take 3 minutes or more per epoch, and training the model on the full data set can be time consuming. However, the biggest challenge I encountered in creating and

analyzing the model was related to the choice and to the implementation of the performance metrics for the CNN. It took me some time to find and adapt the code which would compute the precision, the recall and the F_1 score after each epoch of training and evaluating the model.

Refinement.

In order to attain a better performance of the CNN model, I changed some of the parameters such as the number of filters per convolutional layer, the kernel sizes and the number of convolutional layers. I also experimented with various pool sizes, dropout rates and batch sizes. I also varied the number of epochs. These changes resulted in a second CNN model, as described below (see also the corresponding diagram).

(1). Given that I am using the same embedding layer, I decided to increase the depth of the deep network from three 1D convolutional layers to five 1D convolutional layers. All these layers have 100 filters, the kernels sizes varies from 2 to 6.

(2). I kept the activation functions as ReLu. I also removed the BatchNormalization layers, as I noticed that they do not have a significant effect on the performance.

(3). I changed the dropout rate from 0.2 to 0.5 for both Dropout layers. I removed the L_2 regularization from the output Dense layer.

(4). I kept the Adam optimizer, and I changed the batch size from 64 to 50. I increased the number of epochs from 4 to 10.

RESULTS

Model Evaluation and Validation.

Three different models and two distinct word embedding methods were used on a data set whose corpus consists of roughly $400,000$ rated reviews. To compensate the fact the data is unbalanced, with most of reviews having a 5 star rating, the StratifiedShuffleSplit method in sklearn was used to split the data.

The simplest approach used was the **NLTK sentiment analyzer** utility combined with the Tf-Idf vectorizer method, on the entire data set. The model was used for a preliminary evaluation only and it scored the following numbers:

Accuracy score: 0.81 | Precision score: 0.82 | AUC score: 0.67
Recall score: 0.97
F_1 score: 0.89

The F_1 score and the accuracy have rather high values, although the AUC score is rather small. The high accuracy might be due to the unbalanced data, while the lower AUC score indicates that the model does not perform very well in distinguishing between positive and negative reviews.

The **SVM classifier** is rather slow on this data, on my personal computer. It required 4916 sec. for training and 497 sec. for testing. However this model outperforms all the other approaches, and its performance is hard to beat. The performance metrics are given below:

Accuracy score: 0.94 Precision score: 0.96 Recall score: 0.97 F_1 score: 0.96 AUC score: 0.92

The **Word2Vec** embedding was trained from scratch. As a result the vocabulary size has 8712 words only. These following results emphasize the shortcomings of the embedding. If we take a look at the words which the Word2Vec finds similar to the word `good` we find that the embedding identifies the following words:
`(great, 0.77224159240722), (decent, 0.75202798843383), (nice, 0.65960770845413), (excel, 0.63933765888214), (awesom, 0.5950999259948), (bad, 0.58407104015350), (amaz, 0.57887196540832), (superb, 0.54662299156188), (fantast, 0.53933668136596), (okay, 0.510024368762).`

Now, let's take a look at the **deep learning models**.

• The first CNN model, trained on 4 epochs in about 23 minutes. A screenshot of the last two epochs:

```
 Epoch 3/4
 - 341s - loss:  0.5248 - acc:  0.7381 - val_loss:  0.5231 - val_acc:  0.7363
Precision - 0.74, Recall - 1.0, F_1 score - 0.85
Epoch 4/4
 - 342s - loss:  0.5222 - acc:  0.7394 - val_loss:  0.5253 - val_acc:  0.7368
Precision - 0.74, Recall - 1.0, F_1 score - 0.85
The first model trained in 1418.37 sec.
```

On the test set the first model scores: Test loss: = 0.51 Test accuracy = 0.74
Notice that the validation accuracy and the test accuracy have very similar values.

• The second CNN model, trained on 10 epochs, for about one hour. The output for the last two epochs is given below:

```
 Epoch 9/10
 - 344s - loss:  0.5743 - acc:  0.7388 - val_loss:  0.5767 - val_acc:  0.7365
Precision - 0.74, Recall - 1.0, F_1 score - 0.85
Epoch 10/10
 - 345s - loss:  0.5743 - acc:  0.7388 - val_loss:  0.5767 - val_acc:  0.7365
Precision - 0.74, Recall - 1.0, F_1 score - 0.85
The model trained in 3579.66 sec.
```

On the test set the model scores: Test loss: = 0.57 Test accuracy = 0.74.
Again the validation accuracy and the test accuracy have very similar values.

COMMENTS:

• The two CNN were trained on 70% of the data, validated on 10% of the data and tested on the remaining 20% of the data (which is disjoint from the other two sets). The validation and the test accuracies are almost equal, therefore the models are robust, in the sense that they are not very much affected by the changes in the data set.

• The performances of the two models are quite similar, we do not have a significant improvement from one model to the other. Both models have lower performance scores than the benchmark model, the SVM classifier. The fact that the performance scores remain practically unchanged with the changes in the model architecture is an indication that these metrics are mostly determined by the embedding layer and the size of the vocabulary used.

• *Given that the two neural networks have quite different parameters, and that the test accuracy is practically unchanged we can conclude that the best results were achieved.*

• SVM performs quite well on smaller data sets, and in this case I suspect that the text corpus was not large enough to create a strong embedding. Using a pre-trained embedding would probably make a more significant difference in the test accuracy of the neural network than the optimization of the various parameters.

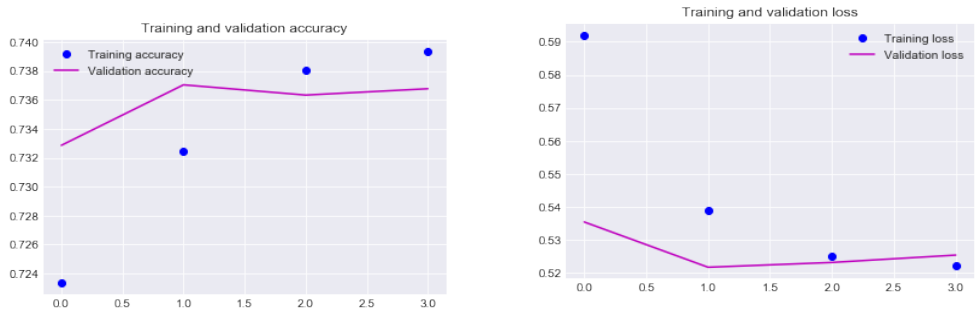
Justification.

The convolutional neural networks chosen here follow most of the suggested practices for achieving the best results in sentiment analysis and text classification , see for example J. Brownlee's blog for an informal discussion or [7] for a more formal approach. Both models follow the schema developed by Kim [4], which was applied with excellent results on numerous data sets. Due to computational limitations, a pre-trained embedding was not used in this case. However, I consider that the self-contained results are interesting on their own, especially for those cases when the vocabulary obtained from the text corpus contains numerous unusual words, or consists of words in other languages where such pre-trained embeddings are easily available.

CONCLUSION

Free-Form Visualization.

We will take a look at several pots that depict how the two CNN models perform epoch by epoch.



The First CNN Model: model_first

From the above two plots we can see that the training accuracy continually increases from epoch 1 to epoch 4, while the validation accuracy remains almost unchanged after the second epoch. The validation accuracy still has a positive slope after the third epoch, but this slope is smaller than the one for the training accuracy curve. On the other side, we see that the training loss decreases while the validation loss has a more stationary behavior. This might indicate some degree of overfitting.

The Second CNN Model: model

For the second model, which was trained on 10 epochs, the training accuracy sharply increases in the firs epoch after which is mostly constant. The validation accuracy on the other side remains stationary after the first epoch. The difference between the values of the two accuracies might be related to using higher dropout rates in this model than in the previous one. The losses have similar values, which again does show that the model is just right.

Reflection.

I did have some interest in natural language processing before I started this project, but after the work done here I have a greater interest in the questions that arise in this field. I also have a keen interest in neural networks, the results that can be obtained with simple neural networks can be quite impressive. During this project I came upon a number of results and sources I plan to investigate further.

For this project, I choose a reasonably large data set for my computer. This data set is easy to manipulate and allowed me to follow the initial plan I outlined in the proposal. For the future, I plan to delve deeper in this area and try more challenging data sets, combined with pre-trained word embeddings. I also intend to investigate classifications on more that two labels, and to research how other sentiments (such as sarcasm) that are harder to handle could be studied and classified.

Improvement.

The best way to obtain a better performance for the CNN model is to use a pre-trained embedding with a larger vocabulary size, such as Google's trained Word2Vec model. I think, this change will significantly improved the performance without major changes in the network architecture.

Several options to experiment with the model include: remove certain words from the vocabulary, such as very rare words or one letter words; increase the length of the deep network by adding convolutional layers; try 2D convolutional layers; experiment with other optimizers. However, I am not entirely convinced that any of these changes would bring major improvements to the performance as long as the embedding layer is maintained as it is; this also follows from [4].

The model can also be expanded, by taking into account other features which were not considered in the present analysis, such as connections with the 'brand' or the 'price' of the reviewed item. This analysis would require a more thorough investigation and cleaning of the data.

REFERENCES

[1] B. Liu, *Sentiment Analysis and Opinion Mining*, Morgan and Claypool Publishers, ebook ISBN 9781608458851, May 2012.

[2] T. Joachims, *Making large scale SVM learning practical*, in Advances in Kernel Methods - Support Vector Learning, B. Schölkopf, C. Burges, A. Smola, Editors, 1999, MIT Press.

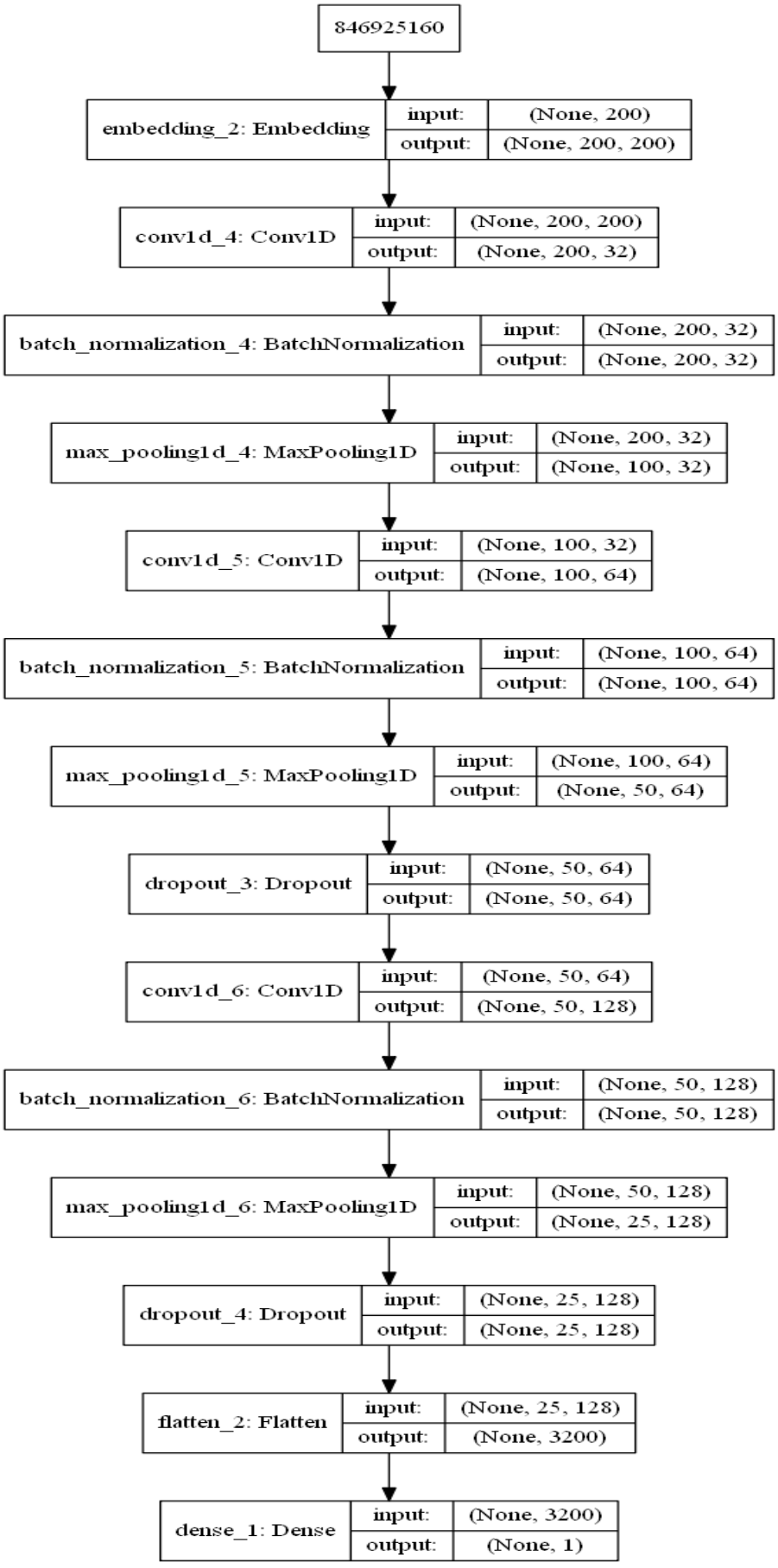
[3] L. Zhang, S. Wang, B. Liu, *Deep learning for sentiment analysis: A survey*, arXiv: 1801.07883.

[4] Y. Kim, *Convolutional neural networks for sentence classification*, arXiv: 1408.5882v2.

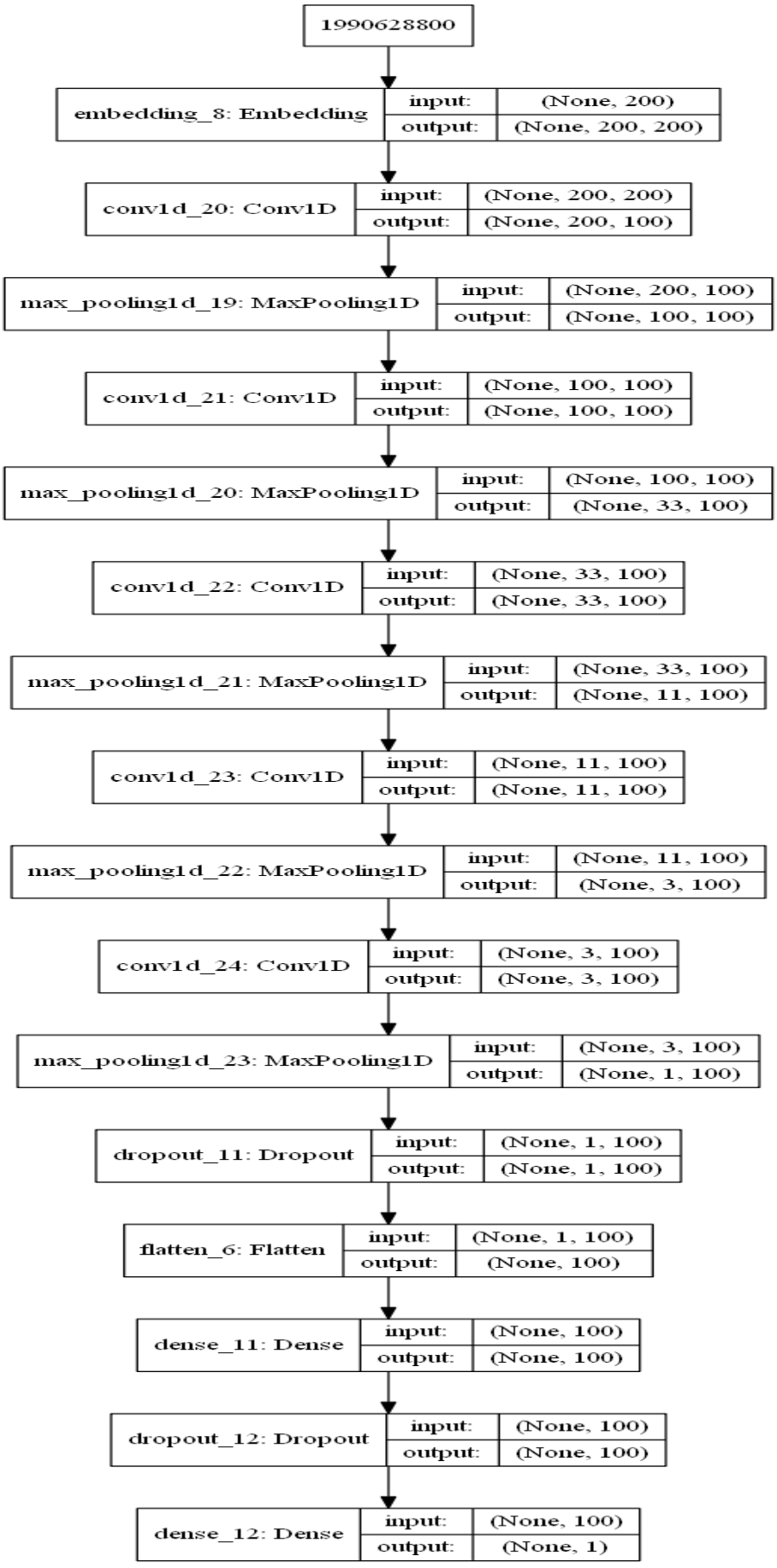
[5] C.J. Hutto, E.E. Gilbert, *VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text*. Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014.

[6] T. Mikolov et al., *Efficient Estimation of Word Representations in Vector Space*, arXiv:1301.3781.

[7] Y. Zhang, B. Wallace, *A sensitivity analysis of (and practitioner's guide to) convolutional neural networks for sentence classification*, arXiv: 1510.03820.



The First CNN Model: model_first



The Second CNN Model: model