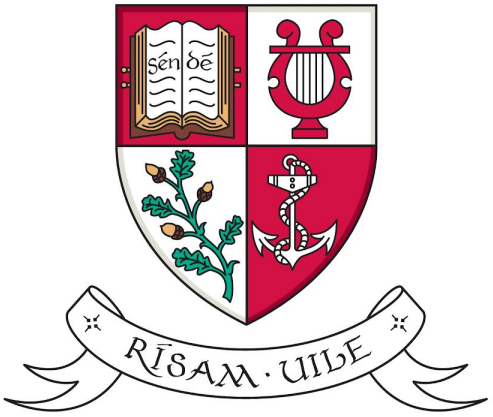
Cork Institute of Technology



A study on sentiment classification of news headlines using deep learning

by

Joao Tiago Viegas - R00157699

Supervised by Dr. Laura Climent

Higher Diploma in Science in Data Science and Analytics

December 2018

Attestation of authorship

This project report is the sole work of Joao Tiago Viegas unless otherwise indicated and is submitted in the partial fulfilment of the degree of Higher Diploma in Science in Data Science and Analytics. I understand that significant plagiarism, as determined by the examiner, may result in the award of zero marks for the entire assignment. Anything taken from or based upon the work of others has its source clearly and explicitly cited.

Date: 03 December 2018

Signature: Joao Tiago Viegas

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my supervisor, Dr. Laura Climent, for her guidance and support throughout this project. Dr. Laura has set me on the right course from day one, and in addition to the questions and suggestions, she kept me focused on delivering something valuable from the start.

I would like to thank Dr Robert Heffernan, for the critical analysis and suggestions to improve this document.

I would like to thank Larisa Soshnikova for the helping me on the classification of the dataset, and for the support throughout all this phase.

I would also like to thank Rabih Abou Fakher for the critical discussions on machine learning and on the project in general, and also Lucy Reid for proofreading this document.

As time is the most precious commodity we have, I am grateful to you all.

Thank you.

Abstract

The main motivation of this experiment was to study deep learning and assess some of its different models applied to sentiment classification of news headlines, and then to use these same models to classify unseen data. A baseline model is also used, the classic classification Naïve Bayes model, to provide a comparison with more simple machine learning models.

The dataset for this experiment is a subset of the well known reuters dataset, that we can categorize as financial or economics-related, with news headlines ranging from January, 8th 2007 through October, 2nd 2018.

Apart from the baseline Naïve Bayes model, we have used 3 feed-forward neural networks and a convolutional neural network. Most of the neural networks achieved a better average accuracy then the baseline model, Naïve Bayes had an accuracy of approximately 75.7%. The best model in the experiment was the convolutional neural network, achieving an average accuracy of 80.1%.

1. Introduction and Motivation

In the latest years we have experienced an extreme growth of data in our daily lives, from Short-Message-Service(SMS), the internet, the social networks, blogs, posts, podcasts, smartphones to the Internet-of-Things (IOT) and the extensive availability of sensors and System-On-Chip solutions, and it feels as if this phenomenon is only starting yet.

This data is out there for everyone to tap into and leverage in ways that we are still discovering nowadays. As an example, daily news, apart from a stream of factual data and, to some degree, processed data, provide insight of what is the commonly perceived status of an entity, whether a product, an event, a person, a group or an organization, among other entities. This perceived status can convey several dimensions of information, that can be leveraged to understand the past, present and, eventually, to predict the future.

News, headlines, also tweets and social posts, can provide customer feedback to companies on how successfully their products are playing out in the market, and nudge them towards a change of their strategy to tackle unexpected hurdles and unknown variables.

To a similar extent, the information conveyed by these sources may act as a proxy for financial investor sentiment, regarding a specific corporation, and be used as a variable for predicting stock market volatility, or on its own or as an augmentation variable to other predictive models. This type of sentiment analysis, also called opinion mining, refers to the use of natural language processing to systematically identify, extract, quantify, and study affective states and subjective information expressed in a piece of text, especially in order to determine whether the writer's attitude towards a particular topic, product, etc. is positive, negative, or neutral. It has been widely tested in behavioural finance, and frequently used

lately by investors as an alternative tool to generate *"alpha"*, e.g., excess return against a benchmark.

Recently, a subfield of Machine Learning (ML) called Deep Learning (DL) has been playing a prominent role in solving these kind of natural language processing (NLP) tasks. DL has been around for many years, but only in this decade has been recognized as an effective tool to solve historically difficult problems of machine learning.

The input data for the research was extracted from the reuters web site, data that we can categorize as financial or economics-related, ranging from January, 8th 2007 through October, 2nd 2018.

The main motivation of this study is then to learn deep learning and assess some of its different models applied to sentiment classification of news headlines, and then to use these same models to classify unseen data. A baseline model is also used, the classic classification Naïve Bayes model, to provide a comparison with more simple machine learning models.

In the next chapter, a concise overview of latest researches and studies on the interrelated subjects of financial sentiment analysis, opinion analysis, machine learning and deep learning, is presented.

In Chapter 3 an explanation of the study, and its theoretical and technical basis, is provided. Deep Learning is introduced and the input data is described. We then describe the different stages of the experiment.

The results of the study and all the outcomes in each of the experimental stages are presented in the 4th chapter.

Finally, in the 5th chapter, we review the main findings of the study, we try to critically assess the relevance of the outcomes and a set of ideas and recommendations are lined up as a potential basis for a future work.

2. Literature Review

Natural Language Processing (NLP) as a subfield of machine learning and Classification as a typical machine learning task, have been subject to extensive research in the latest years. With the availability and profusion of data sources, researchers and engineers of all scientific domains are finding specific contextual challenges that might benefit from the application of these tools, they are also finding new challenges and opportunities, in some cases providing more value to existing solutions and products, in other cases creating new solutions and products. One science domain that has leveraged this tool is economics and finance, we can find extensive research on the application of NLP and machine learning in finance data. Convolutional neural networks (CNN) have emerged as a relevant solution obtaining the best outcomes for these kind of tasks. In this chapter we present a review of some research being produced lately on this subject.

Rojas-Barahona[1], compiled a set of sentiment analysis (SA) evaluation records across different machine learning models, recursive(i), not recursive(ii) and the combination of both(iii) on 3 well known datasets (movie review, the Sentiment Treebank and the twitter sentiment datasets). The study shown that on the first 2 datasets, convolutional neural networks (CNN) performed better, also the study has shown the variants of long short term memory (LSTM) models are statistically indistinguishable from variants of the recursive models. On the twitter dataset though, the best models were the ones from the third group. Also remarkable was the evidence of the importance of initializing the input layers with pre trained word embeddings.

Reshma U et al. [2] applied Valence Aware Dictionary and sEntiment Reasoner (VADER) as a lexical classification approach, Support Vector Machine algorithm and Convolutional Neural Network (CNN) deep learning algorithm to a data set of news in order to identify the data with a positive sentiment and filter out the data with neutral or negative sentiment. This experiment yielded better results using CNN, with a training accuracy of 96% and a test accuracy above 85%.

Çano and Morisio [3] researched a new neural network architecture designed for sentiment analysis of long text documents called *NgramCNN*, that uses pre-trained word embeddings and a very simple single-layer classifier, and that achieved 91.2% accuracy on the popular IMDB movie reviews dataset. The experiment found *NgramCNN* to be more accurate than similar shallow convolution networks and deeper recurrent networks.

Kirange and Deshmukh [4] studied the effect of emotion classification of financial news, available in the public domain, to the prediction of stock market prices. Using Naive Bayes, KNN(k-nearest neighbors) and SVM (support vector machines) for sentiment classification of news, they were able to establish a correlation between the sentiment and the stock price trends for a set of companies during a 10 years time span.

Sohangir et al.[5] studied the impact that Deep Learning models such as long short term memory (LSTM), DocVec and CNN can have in financial sentiment analysis. It used a dataset derived from the StockTwits social network, where investors share their info and knowledge. The conclusion underlined the best accuracy obtained by using a convolutional neural network.

John and Vechtomova [6] used text vectorization models, such as N-gram, TF-IDF (term frequency–inverse document frequency) and paragraph embeddings, coupled with regression model variants to predict the sentiment scores (Simple Linear Regression, Support Vector Regression, XGBoost Regression). Amongst the methods examined, unigrams and bigrams coupled with simple linear regression obtained the best baseline accuracy, and their study showed that text augmentation with content of articles and other content did not show any improvements to the accuracy of their models.

Khuong Vo et al. [26] found that the application of deep learning in sentiment analysis could benefit of domain knowledge, which was not being factored in, in most of the cases. They also found that the loss function should be adjusted to properly define the degree of error of misclassification. They proposed to combine deep learning with domain knowledge using sentiment scores, learnt by regression, to augment training data, and also to introduce a penalty matrix for enhancing the loss function of cross entropy. In their experiment the convolutional neural network (CNN) models, in most of the cases, achieved an average precision between 90% and 95%.

Hassan and Mahmood [27], propose a model leveraging both convolutional and recurrent layers to efficiently perform sentiment analysis. They define a CNN architecture with multiple layers to capture long-term dependencies in sentences, and a recurrent layer LSTM (long short term memory) is used as a substitute for the pooling layer to reduce the loss of detailed local information and capture long term dependencies. This model, named *ConvLstm,* achieved an accuracy of 88.3% on the well known Stanford Sentiment Treebank (SSTb) dataset.

3. Methodology

In this chapter we describe the model used for defining a baseline for this study, the machine learning classification algorithm called Naïve Bayes. We then describe Deep Learning in general, and the algorithms used in more detail. We then describe the data used for the experiment, and the different stages of the experiment are then presented.

3.1 Naïve Bayes - baseline machine learning model

Naïve Bayes is one of the most simple machine learning classification algorithms. It's called naïve because its formulation makes some naïve assumptions, mainly that the probability distribution of its features is independent from the other features distribution. It can be used with nominal values, it handles multiple classes and performs well even with a small amount of data.

It is based on conditional probabilities, specifically on the Bayes rule. As an example, let us assume we have a container, *container1*, with 5 balls inside, 3 yellow and 2 green.

What is the probability that a ball is green, when we randomly retrieve it from the container? It is 2/5, as we compute the probability dividing the number of balls with that color by the total number of balls.

Let us assume now that we have 2 containers, and in the second container, *container2*, the distribution of balls is different, with 2 green balls and 1 yellow ball inside.

In this case what will be the probability of getting a yellow ball from container 1? This is conditional probability. We can write:

$$P(yellow|container1) = \frac{P(yellow \ \& \ container1)}{P(container1)} \quad (1)$$

In our case this translates to

P(yellow & container1) = (3/8) / (5/8) = 15/64 = 0.2344

The Bayes rule also enables us to play with these symbols in the equation, for instance,

if we have *P(x|c)* but we need *P(c|x)* we can find it using:

$$P(c|x) = \frac{P(x \mid c) * P(c)}{P(x)} \quad (2)$$

Eventually this rule and the conditional probabilities are the basis for the classification task using Naive Bayes. For every observation the input vector is used to define a probability of a specific variable (word) when having been assigned a specific class,

$$P(word|classA) = \frac{P(classA \mid word) * P(word)}{P(classA)} \quad (3)$$

The sum of these conditional probabilities will eventually determine the most likely class of a sentence. As this is not the main model in the study, being used as baseline regarding the remaining models, the reader is referred to [8] for additional information on the Naïve Bayes classification algorithm.

3.2. Deep Learning

Deep Learning (DL) is a subfield of machine learning (ML). Its rationale is to learn through a sequence of layers of increasingly meaningful representations. The “deep” in the name is related to the depth/number of these successive layers, also called the *depth* of the model.

3.2.1. Model layers

These layers can amount to, some cases, hundreds of successive layers of representations, and in some other machine learning cases it might just aim to learn one or only two representations of the data, and are therefore sometimes called shallow learning. These layers will learn their parameters/weights automatically from exposure to training data, via neural networks, and they use processes such as gradient descent, that we will describe

later. In a simplified manner, we can think of these layers as equations, whose parameters/weights are tuned during the training process, based on the difference on any iteration between their overall value and the desired output (training classification) which is called the *loss*. The tuning is most of the time done by a gradient descent, which tries to decrease this *loss* at every step of the training phase.

As an example of these layers and the learning process in a deep learning algorithm let us look into one of the most common deep learning tasks, image recognition, specifically, and try to identify digits from images.

Taking as an example any digit representation, in this case a "4":

fig. 3.1 - example of digit image [7]

This image can be translated to a 28*28 matrix, where each position has a value, from 0 to 1, also called its *activation* value, corresponding to its *"light effect"* as if it is lighter or darker. Let us think about this image as a matrix, and as the input to our deep learning network.

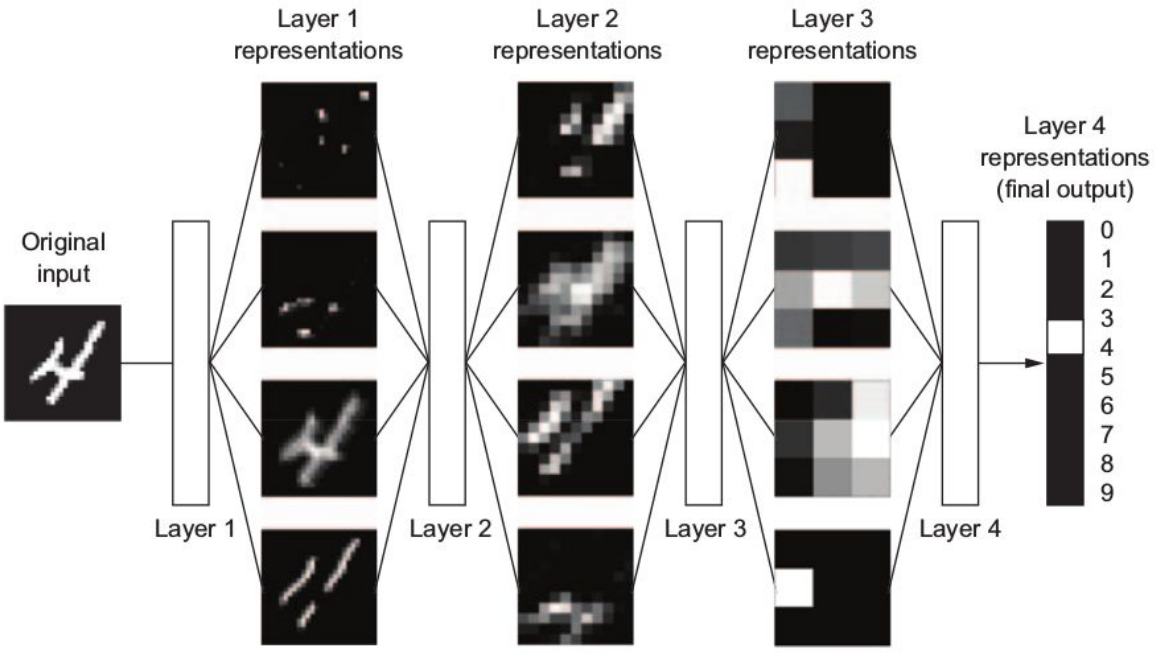


fig. 3.2 neural network [7]

We have one input layer, three hidden layers and one output layer. The hidden layers are as a set of transformation functions from the input to the output, which in this case is nothing more than a vector, or a single column matrix, with what we call the possible classes, in this case the 10 existent digits. Here we will end up also with a value telling us how much the network thinks the digit is represented by an input image. So the network works as a distillation operation, where the input data goes through a set of filters and becomes purified[7], or in this case translated to a class, according to the task in question.

So thinking of an element of a layer as an equation element, with weights as parameters, any one of these elements will turn to be an input to the next layer equation. In a simplified network with one hidden layer we would have:

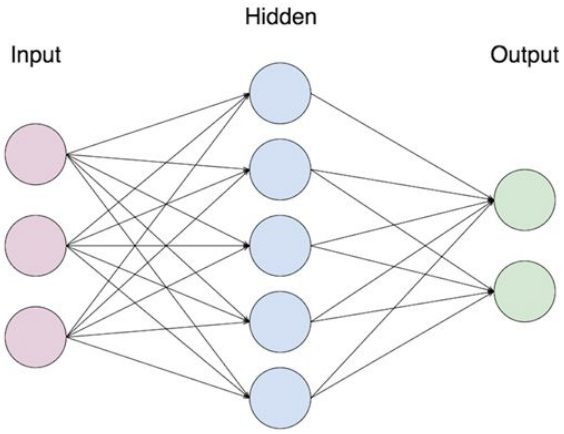


fig. 3.3 neural network layers [25]

It is in fact a simple rationale, that when applied in scale can perform amazingly well.

3.2.2. The learning functions: *loss* and *optimizer*

But now one might ask, how are these transformations defined, or if we understood correctly, how are these transformations *learned*?

These transformations, or what the layer does to its input data, end up defined in the layer weights, which is nothing more than a vector, and this process of defining these weights in all layers, so that the network maps correctly the images to the digits, is in fact the actual *learning*.

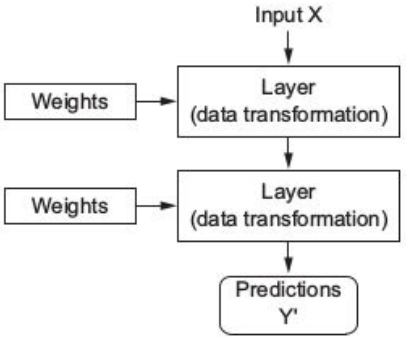


fig. 3.4 weights as layers parameters [7]

To do this efficiently, as neural networks might end up having millions of parameters, there is a *loss* function that measures the distance from predictions to the desired output, target values, in the training phase, in other words, assessing the quality of the output.

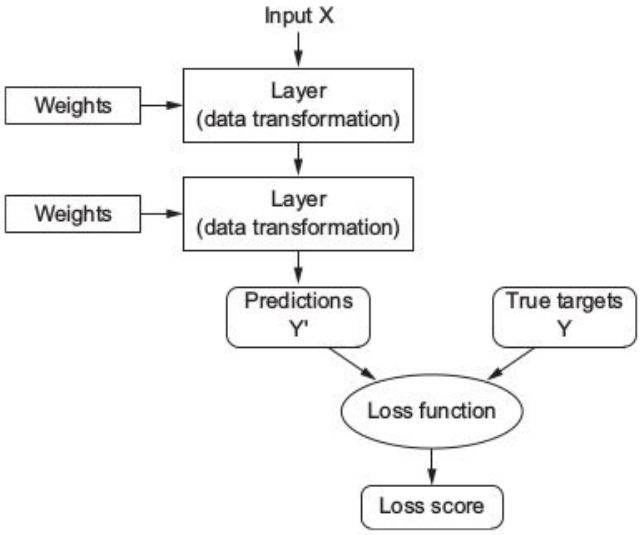


fig. 3.5 the *loss* function [7]

This *loss* function is then the feedback signal to properly adjust the layers parameters. Here the *optimizer* function is introduced, and it implements the backpropagation algorithm,

which is the most important algorithm in deep learning. After an initial weights setup made in a random manner, the *optimizer* identifies the best changes to implement in the layers' weights so that the network quickly minimizes its *loss* function outcomes.

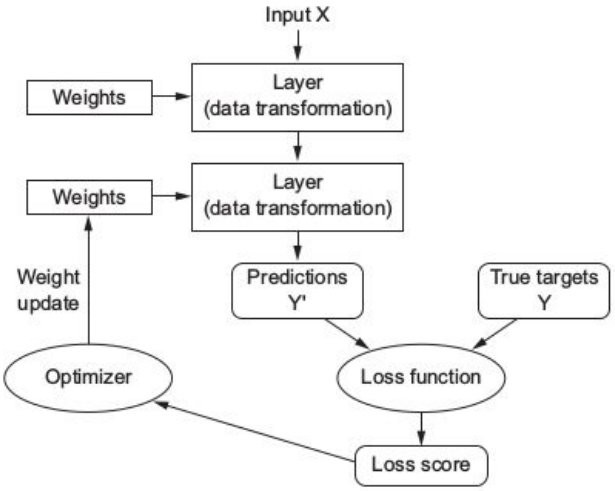


fig. 3.6 the optimizer [7]

Before anything we also need to define the metrics, that will allow us to assess our model performance during training and testing, in terms of accuracy, recall, f1 score, etc.

The *optimizer* will notify about the ongoing state of the metrics, as it will be keeping score of the *loss* function at every stage.

At this point one might ask how does this *optimizer* implement the backpropagation and set up the best adjustment for the layers weights.

If we take the network defined in fig. 3.3, let us suppose we have a representation of an image as an input layer of a 28*28 = 784 dimensional vector, in which every value is the grey value for every pixel in the image.

If we take a single neuron in a layer, its output will be obtained from all the weights in the previous layer multiplied by the inputs plus a bias factor, and in the end the resulting value will be smoothed to a value somewhere between 0 and 1 by the activation function, in this

case it could be the rectified linear unit function (*relu*), just as the light effect on the pixel mentioned before was between 0 and 1.

We define the output vector as *y=y(x)*, a 10-dimensional vector. The end layer equation here can be defined as:

$$y = f(WX + b) \tag{4}$$

where:

f = activation function

W = weights vector

X = inputs vector

b = bias vector

So let us define a *loss* function:

$$L(w,b) = \frac{1}{2n} ||y(x) - a||^2 \tag{5}$$

with:

n = number of all training inputs

a = vector of outputs

Here it is implicit that *a* depends on *x,w* and *b*, and it is there just to simplify the notation. We will call this the *quadratic loss function*, also known by *mean squared error* or just *MSE*.

Looking closely at the function, we see that *L(w,b)* is never negative. It is also simple to see that the loss function becomes small, *L(w,b)≈0*, when *y(x)* is approximately equal to *a* for all training inputs, which is what we want, on the other hand, we do not want *L(w,b)* to be large, that would mean that *y(x)* is "far away" from *a* for a large number of inputs. So our aim is to minimize the Loss *L(w,b)* as a function of the weights and biases, and for that we use an algorithm called *gradient descent*.

3.2.3. Gradient descent

Gradient descent is a common technique used in minimization problems and we will provide a generic introduction to it in this section.

Supposing we want to minimize some function, *L(v)*, with variables v= $v_1, v_2, \ldots$, let us imagine *L* as function of just two variables, v_1 and v_2.

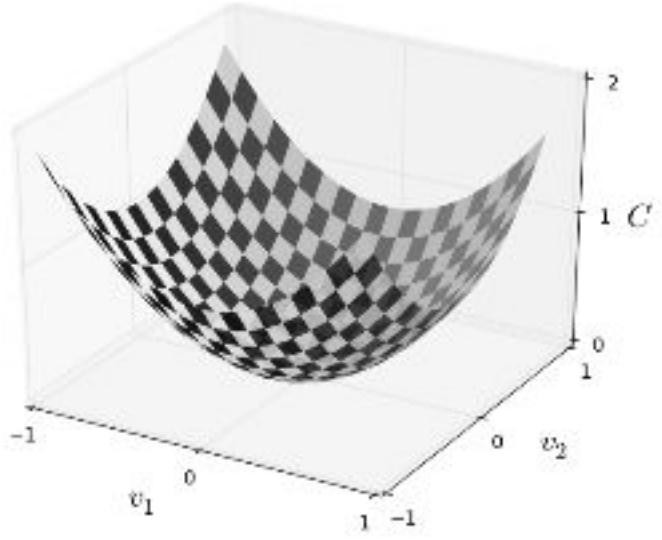


fig. 3.7 gradient descent surface on variables v_1 and v_2 [16]

In this case we want to find the minimum value of *L*, depicted as the surface above the variable space defined by v_1 and v_2. Of course the case of two variables is a simple one, but in cases where the number of variables is considerably higher, as in most cases of neural networks, this task is non trivial, even calculus will not help. The gradient descent works by randomly choosing a starting point, and then based on the gradient vector lets us, step by step, approximate to the minimum.

$$\nabla L \equiv \left(\frac{\partial L}{\partial v_1}, \frac{\partial L}{\partial v_2} \right)^T \quad (6)$$

This way with the gradient vector we can always figure out what movement allows us to minimize our position, finding a route to slide down in the function surface.

There are further details on the gradient descent that are not covered in this document, the reader is referred to [16] and [7] for additional information on the gradient descent.

3.2.4. Convolutional Neural Networks

In this section we introduce convolutional neural networks, using a simple example.

Let us imagine that there were 4 ways of expressing emotion with what we are now going to define as images:

X / \ O

these 4 images can be described by a 3*3 matrix, and therefore we have

$$\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix} \quad \backslash \quad \begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix} \quad \mathbf{X}$$

$$\begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix} \quad / \quad \begin{bmatrix} -1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \end{bmatrix} \quad \mathbf{O}$$

fig. 3.8 - simplified example of training data for a convolutional neural network

Our problem is we want to identify these images. And for a start let us assume there is a previous knowledge that the model has already developed through training, in the form of sub-patterns with lower dimension, in this case 2*2 matrices:

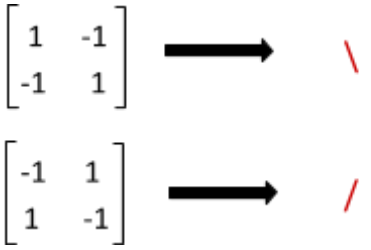


fig. 3.9 - simplified example for set of filters of a convolutional neural network

with this knowledge we can identify these patterns in the original messages and later map those findings to the correct emotion.

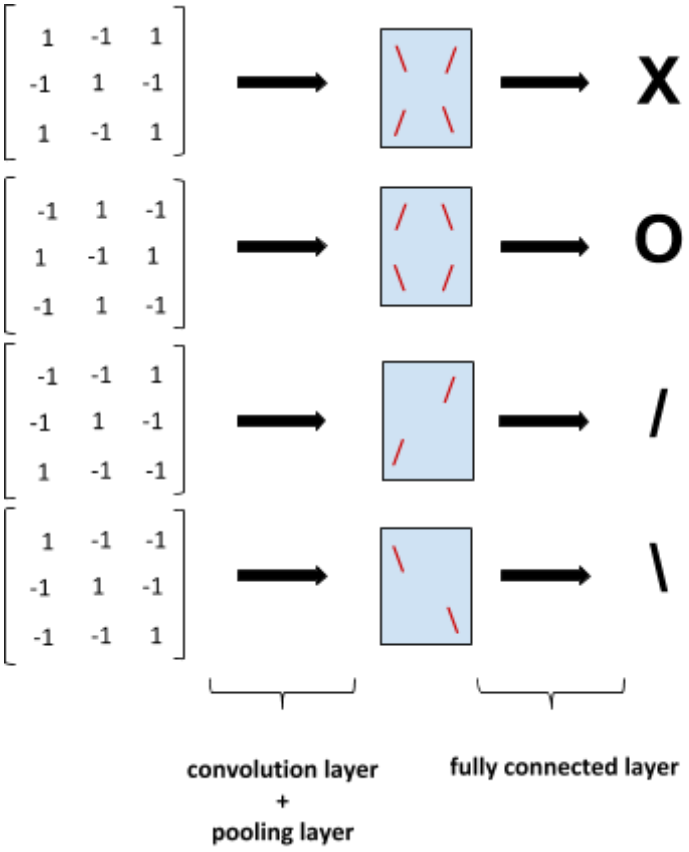


fig. 3.10 - simplified example of a convolutional neural network model

In the convolutional neural network, in a first stage the convolutional layer and the pooling layer, split the image in pieces and try to match it against known patterns, called filters (fig. 3.9). It will then output a kind of map, made of that matching information. The fully connected layer then uses logic to relate that map with the images and the classification (fig. 3.10), as if the input map it receives is a transformed set of variables, a transformed input, of the initial image data.

Convolutional neural networks are therefore combinations of pairs of *convolution+pooling* layers and fully, dense connected layers, but its inner rationale is still the same as explained before for deep learning networks, the convolutional and the dense layers do have an *activation* function, and the network learning is still driven by the *loss* and the *optimizer* functions.

3.3. Dataset

The data we used in this study are the news headlines retrieved from the US Reuters site using an available github project to retrieve and compile the data [21]. This data was then filtered out in order to use a subset that contained news related to a specific company, in this case International Business Machines (IBM), and the dates range from January, 8th 2007 through October, 2nd 2018. The dataset has 9385 observations, with the variables *id*, *timestamp*, *headline* and *classification*. The reason for this subset definition is that the author of this document wants in a future development of this study to relate these sentiment analysis outcomes with the history of this specific company stock market price.

	A	B	C	D
98	97874	20070418 05:46 AM EDT	IBM profit up on software and services	1
99	97911	20070418 10:34 AM EDT	US STOCKS-Tech shares fall on IBM, Yahoo results	−1
100	98161	20070418 12:48 AM EDT	IBM 1st-qtr profit up; software, services advance	
101	98164	20070423 11:44 AM EDT	UPDATE 1-RESEARCH ALERT-Lehman upgrades IBM to overweight	1
102	99242	20070424 01:05 PM EDT	US STOCKS-Blue chips jump on IBM; Dow again eyes 13,000	
103	99306	20070424 01:21 PM EDT	IBM raises dividend, stock buyback	1
104	99442	20070424 02:25 PM EDT	US STOCKS-Dow nears 13,000 as IBM boosts blue chips	1
105	103811	20070424 03:20 PM EDT	Fitch may downgrade IBM ratings on stock buyback	−1
106	104127	20070424 04:07 PM EDT	US STOCKS-Dow rises as IBM boosts blue chips	1
107	110371	20070424 04:35 PM EDT	US STOCKS-Dow ends up as IBM lifts blue-chip index	1
108	110377	20070424 05:29 PM EDT	US STOCKS-Dow ends higher on IBM boost; Sun Micro off late	1
109	110593	20070424 05:30 PM EDT	US STOCKS-Dow ends higher on IBM boost; Sun Micro off late	1
110	115068	20070424 07:01 PM EDT	Dow ends higher on IBM boost	1
111	115464	20070424 09:11 AM EDT	IBM authorizes $15 bln for stock repurchase	1
112	115953	20070424 09:51 AM EDT	US STOCKS-Indexes rise on earnings, IBM dividend rise	1
113	119478	20070424 10:00 AM EDT	IBM's board boosts dividend, stock buyback fund	1

fig. 3.11 - original dataset sample

The data was then classified manually according to its perceived sentiment, using three classes: positive(1), neutral(0) and negative(-1). This task has undoubtedly a level of subjectiveness, as in certain cases it is hard to grasp the exact *polarity* of the text. By *polarity* we mean in what extent does the text convey positive information regarding the financial health of the company or in terms of its attractiveness as possible investment product in the long term. We also included in this class any event regarding new technologies rollout/research or new business deals/agreements. In this light, we would classify as positive the following news headlines:

- "*Dow ends higher on IBM boost*";
- *"Red Hat, IBM form mainframe partnership";*
- *"IBM, Chartered, Samsung unite on chip development";*
- *"IBM profit up on software and services";*

We would accordingly classify the following headlines as negative:

- *"US STOCKS-Tech shares fall on IBM, Yahoo results";*
- *"IBM to cut 1,315 jobs in U.S.";*
- *"SEC says IBM misled investors on expenses";*
- *"US STOCKS-Dow turns negative as United Tech, IBM drag";*

When in doubt we presume the *polarity* is neither positive nor negative and therefore all the remaining headlines were classified as neutral.

3.4. Experimental Stages

3.4.1. Data preparation

For the purpose of the experient we have removed the *id* and *timestamp* columns, so we ended up with two columns only, the *headline* text and its classification.

	A	B
585	Microsoft, Oracle databases gain share vs	−1
586	Chordiant and Help Global Brands DAK and ING Strengthen Customer Relationships	1
587	signs 10-year multi-billion cloud deal with ABN Amro	1

fig. 3.12 - final input dataset sample

3.4.1.1. Class balance

Overall the distribution of the dataset classification was imbalanced. We had then to oversample the training data. We have thought of using SMOTE [14] as an oversampler that creates synthetic data, but due to the nature of the experiment, text classification, we found that it would be more precise to use a straight oversampler, as synthetic generated data would not make cognitive sense in terms of text, and therefore we have replicated the classes so that all had the same number of samples.

This oversampling took place only in the training data, so in our experiment while using 10-fold cross validation, in every fold 10% of the data is taken for testing purposes, so from the 9385 initial dataset length we end up with a training dataset with 8447 observations, and the oversampling takes the initial number of observations of the “neutral” (3044) and “negative“ (632) to the same number of the “positive” (4771) observations.

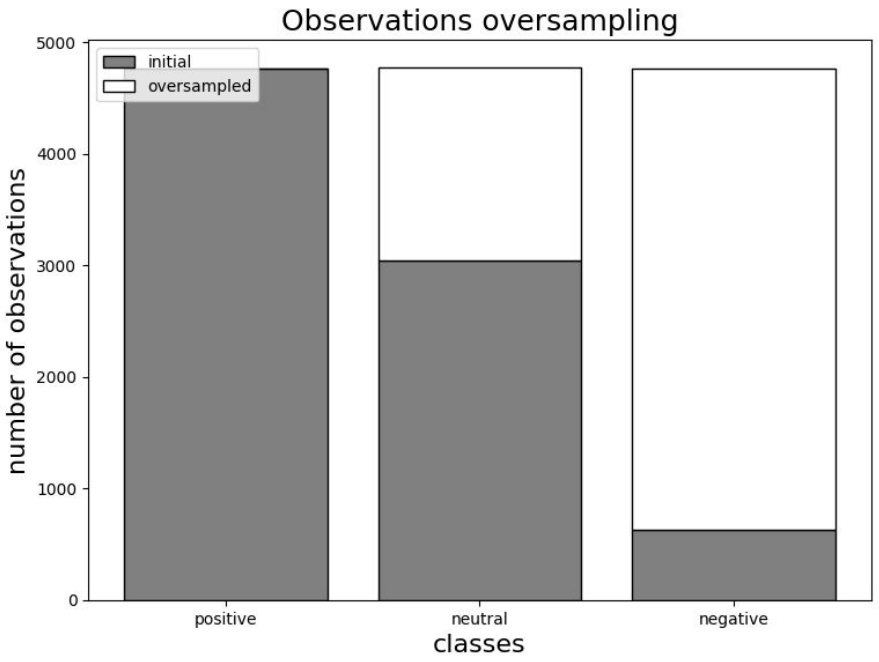


fig. 3.13 - oversampling of classes

3.4.1.2. Text cleaning

Before training our models, we removed the word “IBM” from all the sample texts, so that it would not be taken into account, and that eventually we could test the model in other datasets.

Before translating the sentences to a matrix format we applied a sequence of transformations on the text. We identified and removed punctuation, removed dangling apostrophes with suffixes, then *lowercased* it to properly remove stop words and to lemmatize the tokens. We then shuffled the data at every experiment run.

3.4.2. Baseline model - Naïve Bayes

Before studying the deep learning models, as mentioned before, we implemented a Naïve Bayes classifier, to serve as the baseline for this study.

This implementation involved the transformation of the sentences into vectors using the one-hot encoding rationale. The resulting word vectors were then used to compute the features and classes probability vectors in the training stage. With these vectors we were able then to evaluate the model and classify the test subset.

3.4.3. The Keras deep learning framework

In this study code we have used Python version 2.7 as the programming language and the Keras library [22], version 2.2.4, which is a deep learning framework for Python. We will now introduce Keras and later we describe the different deep learning models used in this study.

Keras provides an easy way of setting up almost any kind of deep learning model, with a user friendly API. Keras runs seamlessly on CPU or GPU, has built-in support for convolutional networks, recurrent networks and any combination of both, and is appropriate to build any deep learning model, from a generative adversarial network to a neural Turing machine. It has also large adoption in the academic community and more than 200,000 users, it is also very popular in Kaggle, where recently almost every deep learning competition has been won using Keras models.

We can think of Keras as a high-level level library to develop deep learning models, it relies on other libraries to handle low-level tensor (multi-dimensional arrays) operations, and allows different of these backend libraries to be plugged seamlessly into Keras.

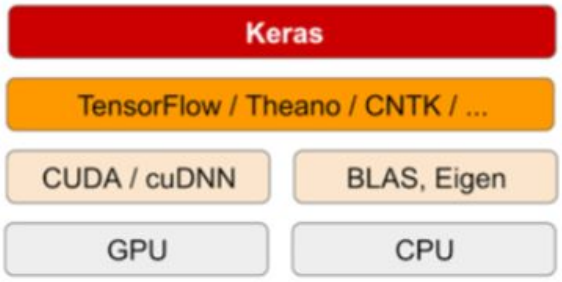


fig. 3.14 - the keras framework stack [7]

The reader is referred to [22] for additional information on the Keras framework.

To develop a deep learning model with Keras framework, and in line with the methodology presented before, there is a common workflow encompassing 5 steps:

1. define training data: input and output tensors;
2. define the layers network that will connect the input to the desired output;
3. choose the metrics, loss function and the optimizer function in order to learn from the data;
4. run your data through the model, using the model *fit()* function;
5. evaluate the model against test data

In a simple example with three layers:

x_train, y_train, x_test, y_test = load_data()	step 1
model = Sequential() model.add(Dense(32, activation='relu', input_shape=(input_size,))) model.add(Dense(32, activation='relu')) model.add(Dense(3, activation='softmax'))	step 2
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',metrics=['acc', recall, precision, f1_score])	step 3
model.fit(x_train,y_train,epochs,batch_size)	step 4
model.evaluate(x_test,y_test)	step 5

table 3.1 - Keras network setup example

Based on this workflow, we have assessed a set of different models, testing different configurations, on the news headlines data set, which we will detail in the following section.

3.4.4. Deep learning models

In this study we have used a small set of networks, we can call them shallow networks, at maximum with two hidden layers (the ones between the input and output layers), except for the convolutional network case.

Mostly in our networks configuration we have tried simple Dense layers with the rectified linear unit (*'relu'*) activation function, which in a simple manner can be described as an identity function that transforms negative values to 0, for it is the most popular activation function in deep learning [7]. In the output layers though, we have applied the '*softmax' activation* function that generates a probability vector matching the output classes and with values summing to 1, which is the most adequate for this case, a multiclass problem with single-label classification, where the classes output probabilities are not independent[7].

Regarding the *loss* function, as we are dealing with a multiclass problem (positive, neutral and negative), we are using the *categorical_crossentropy* function, which is a common selection in this case.

It is relevant that in some models we are using a word embeddings layer, which is a optimized way of translating text to a vector, conveying info about the distance between word vectors and leading to lighter, more dense vectors, as opposed to sparse vectors created with other techniques as, for instance, one-hot encoding technique.

In most of the cases we are using the Keras library Embedding layer that does the transformation from a sentences matrix and its word index to an embeddings dense matrix. In one case we have tested with a pretrained embeddings layer using a words vector from the GloVe site ("*1 hidden layer with Glove pre-trained embeddings*"), called the *Global Vectors for Word Representation* [23] which was developed by Stanford researchers.

We use the same set of metrics across all the models, *accuracy, recall, precision and f1-score*.

Remembering the confusion matrix in a binary classification problem, for the sake of simplicity:

	predicted value		
actual value	values	yes	no
	yes	true positive (TP)	false negative (FN)
	no	false positive (FP)	true negative (TN)

table 3.2 - confusion matrix

the definition of the metrics are then:

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (4)$$

$$precision = \frac{TP}{TP+FP} \quad (5)$$

$$recall = \frac{TP}{TP+FN} \quad (6)$$

$$f1\ score = \frac{2(recall*precision)}{recall+precision} \quad (7)$$

In this experiment, regardless of the insight we can get from *precision* and *recall* or the *f1 score*, we will carefully monitor the *accuracy* metric which will identify the accuracy of the model on correctly classifying all the observations, regardless of its class. As a multi-class, single label problem, the notion of false positive and false negative, used in the binary classification problems, is less precise in this case. Therefore we take the notion of true negative and true positive and the related accuracy formula, dividing its sum by the total number of cases, as the most adequate to assess the models performance.

3.4.4.1. 1 hidden layer model

layers

```
Dense(32, activation='relu')
Dense(32, activation='relu')
Dense(3, activation='softmax')
```

learning

```
optimizer: rmsprop
loss: categorical_crossentropy
metrics: accuracy, recall, precision, f1_score
```

3.4.4.2. 2 hidden layers with word embeddings model

layers

```
Embedding(...)
Flatten()
Dense(32, activation='relu')
Dense(32, activation='relu')
Dense(3, activation='softmax')
```

learning

```
optimizer: rmsprop
loss: categorical_crossentropy
metrics: accuracy, recall, precision, f1_score
```

3.4.4.3. 1 hidden layer with Glove pre-trained embeddings model

layers

```
Embedding(...)
Flatten()
Dense(1024, activation='PReLU')
Dense(3, activation='softmax')
```

learning

```
optimizer: rmsprop
loss: categorical_crossentropy
metrics: accuracy, recall, precision, f1_score
```

3.4.4.4. One dimensional Convolutional Neural Network model

layers	*learning*

`Embedding(...)`
`Conv1D(8, 24, activation='relu')`
`GlobalMaxPooling1D()`
`Dense(3, activation='softmax')`
	`optimizer: rmsprop`
	`loss: categorical_crossentropy`
	`metrics: accuracy, recall, precision, f1_score`

4. Research Findings

In this chapter we disclose the results obtained in the different models. All the code implementation for these experiments is publicly available on Github[24].

4.1. Baseline model - Naïve Bayes

In this process we have used the *10-fold cross validation* method, and we have also *oversampled* the training data. The baseline model achieved an *accuracy* of **75.70% (+/- 1.58%)**, using a dictionary size of 25000 words and a maximum sentence size of 100 words.

4.2. Deep learning models

The models were all trained using 10-fold cross validation.

In every model experiment some parameters were adjusted in order to get the best possible outcome.

dictionary size	size of words dictionary, which holds the most frequent words. default: 25000
sentence size	limit of words to be considered in a sentence. default: 100
embedding size	size of the word embedding vector. default: 64
epochs	number of iterations over the same data in *Keras* model fit operation. default: 16
batch size	number of samples to be trained at every epoch in *Keras* model fit operation. default: 256

table 4.1 - model training parameters

The Keras *fit* method, in the training operation, allows us to keep track of the metrics at every iteration/epoch, this is suitable for us to identify if our model is not overfitting, that can be found when the metrics score and the loss value reach a plateau. In such situations we should retrain the model with less epochs.

For that reason we will present the training scores outcome along the training iterations and the final scores evaluated on test subset of data.

4.2.1. 1 hidden layer model

4.2.1.1. parameters selection

	base	**diff**						
layer dim	32	64		64				16
n layers	3		4	4	tanh	prelu	elu	
act.func.	relu							
avg accuracy %	43.9	43.5	43.4	41.9	42.1	42.2	41.8	40.1

table 4.2 - model training parameters - 1 hidden layer

4.2.1.2. Training scores sample

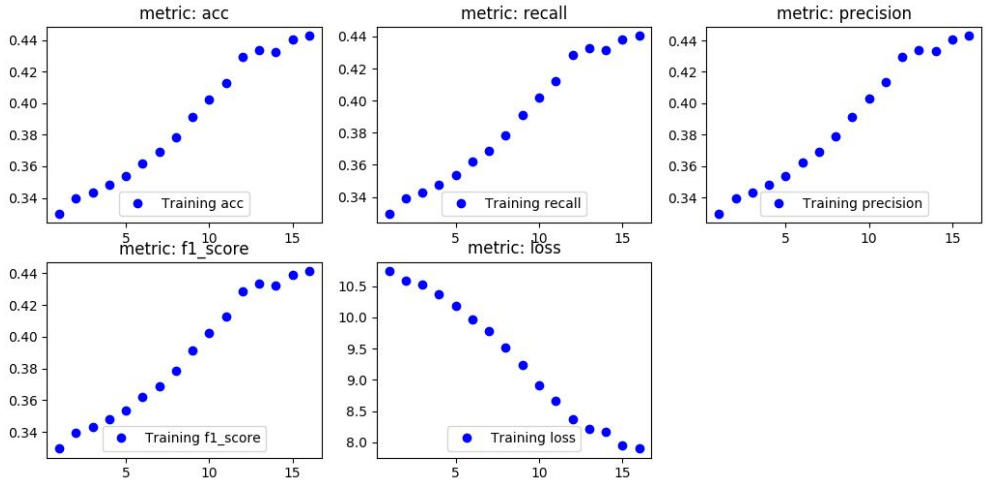


fig 4.1 - training scores sample: 1 hidden layer model

4.2.1.3. Evaluation scores

acc: 43.93% (+/- 5.18%)

recall: 43.88% (+/- 5.24%)

precision: 43.94% (+/- 5.20%)

f1_score: 43.91% (+/- 5.22%)

4.2.1.4. Analysis

This model was the most simple of all, as it takes an input layer without embeddings, with the sparse matrix derived from the sentences and word index, and not taking advantage of any geometric relationship between word vectors, and this probably contributed to the fact

of this model scoring the lowest on all metrics. Somehow the network does not learn the best way of matching the text with the classification, even in the training phase where it never did go over the 50% in all metrics. There is no sign of overfitting as well, and we could not extract any improvement from the model training when we changed the number of epochs and the batch size.

4.2.2. 2 hidden layers with word embeddings model

4.2.2.1. parameters selection

	base	diff									
layer dim	64	32	32	64	64	128	128				512
n layers	4		5	5	8						
act.func.	relu							tanh	prel u	elu	
epochs	10						16				
avg accuracy %	79.1	79.7	79.9	79.2	78.8	79.0	77.9	79.0	79.3	78.4	79.0

table 4.3 - model training parameters - 2 hidden layers with word embeddings

4.2.2.2. Training scores sample

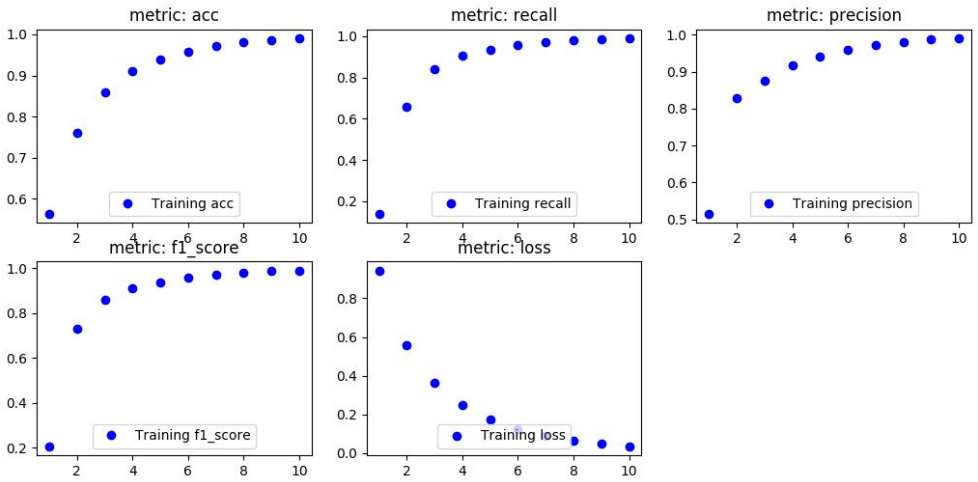


fig 4.2 - training scores sample: 2 hidden layers with word embeddings model

4.2.2.3. Evaluation scores

acc: 79.88% (+/- 1.30%)

recall: 79.82% (+/- 1.27%)

precision: 79.98% (+/- 1.28%)

f1_score: 79.90% (+/- 1.28%)

4.2.2.4. Analysis

In this model we used a Keras Embedding layer for the word vector input. We have noticed before some overfitting when using 16 iterations, so here we have used 10 iterations, and from the training sample graphs the overfitting does not occur anymore.

4.2.3. 1 hidden layer with Glove pre-trained embeddings model

4.2.3.1. parameters selection

	base	diff									
layer dim	64	32		128	256	512	256	1024	1024	768	1024
n layers	4		5				5				
act.func.	relu								tanh		prelu
avg accuracy %	75.9	74.2	73.9	76.2	76.8	76.9	76.5	77.3	72.2	77.2	77.8

table 4.4 - model training parameters - 1 hidden layer with Glove embeddings

4.2.3.1. Training scores sample

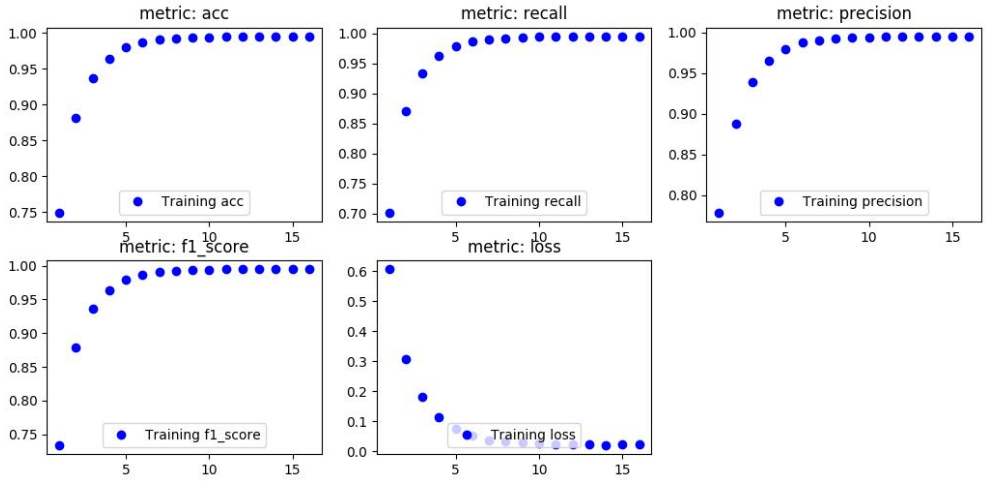


fig 4.3 - training scores sample: 1 hidden layer with Glove embeddings

4.2.3.2. Evaluation scores

acc: 77.78% (+/- 1.90%)

recall: 77.68% (+/- 1.89%)

precision: 77.91% (+/- 1.95%)

f1_score: 77.79% (+/- 1.92%)

4.2.3.3. Analysis

The model including the GloVe word vectors in the Embedding layer performed worse than the one where we have built the Embedding with the Keras library from the actual training dataset, and that might be due to the specific context of this data set, and how the word vector might not generalize well on the financial news headlines from Reuters. Still it is noticeable the overfitting after the 7th iteration. We then trained the model with 7 epochs, but the accuracy did not improve.

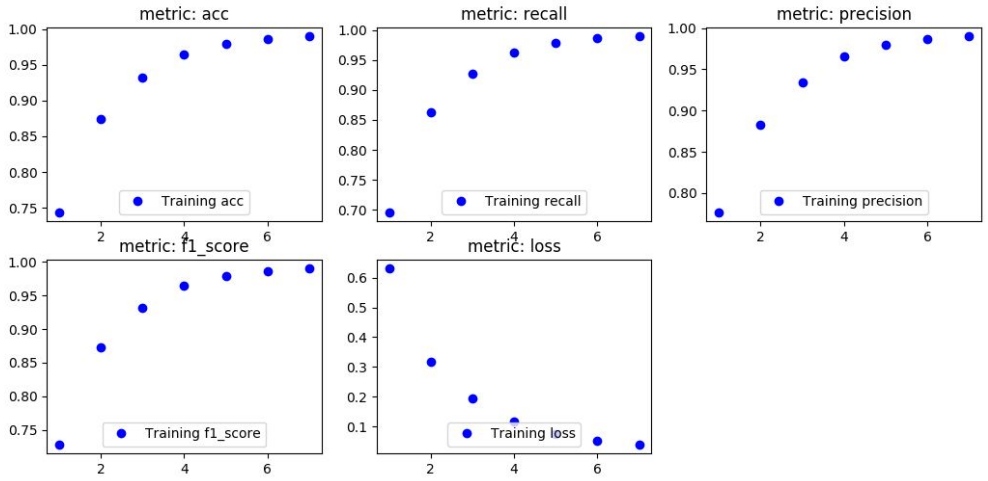


fig 4.4 - training scores sample: 1 hidden layer with Glove embeddings - 7 epochs

acc: 77.61% (+/- 1.11%)

recall: 77.52% (+/- 1.09%)

precision: 77.81% (+/- 1.14%)

f1_score: 77.66% (+/- 1.11%)

4.2.4. One dimensional Convolutional Neural Network model

4.2.4.1. parameters selection

	base	diff									
n filters	64	32	16	8	4		8	8	128	128	8
kernel size	12					6	6	24			24
n layers	4									6	
act.func.	relu										elu
dropout layer	false									yes	
epochs	12										
avg accuracy %	79.0	79.2	79.1	79.8	79.4	79.7	79.9	80.1	79.1	78.8	79.2

table 4.5 - model training parameters - 1D CNN

4.2.4.2. Training scores sample

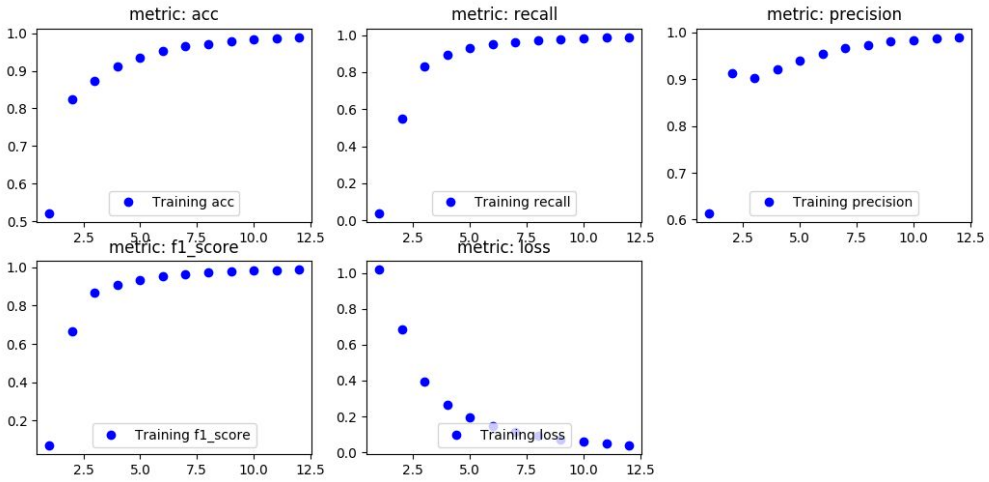


fig 4.5 - training scores sample: 1D CNN

4.2.4.3. Evaluation scores

acc: 80.12% (+/- 0.85%)

recall: 80.01% (+/- 0.85%)

precision: 80.29% (+/- 0.89%)

f1_score: 80.15% (+/- 0.87%)

4.2.4.4. Analysis

This is a really simple Convolutional Neural Network (CNN), we have included a convolutional and a pooling layer and a fully connected layer as output, the simplest case for a CNN. Still it achieves an accuracy above 80%.

5. Main Findings, Recommendations & Conclusion

Apart from the model called "1 hidden layer", all the other neural network models achieved similar accuracy outcomes in the range between 77%-80%, superior to the baseline model Naïve Bayes that achieved an average accuracy of 75.7%.

Overall we could say the CNN model has a slight edge over the “2 hidden layers with word embeddings” model as its average values are roughly similar but with a smaller variation.

While better than a random guess, this definitely might be improved, for it is known that deep learning can achieve far better results in similar sentiment analysis tasks [2][3].

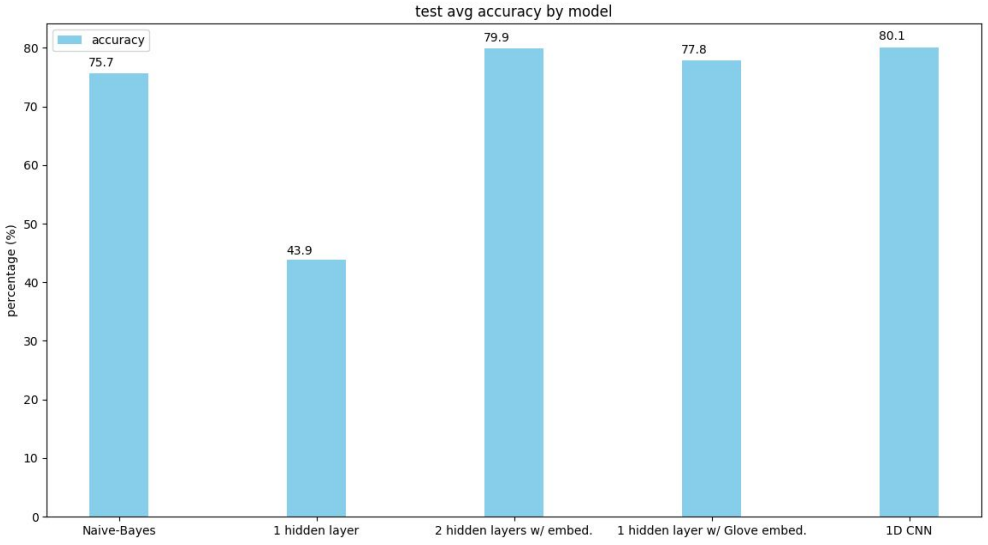


fig. 5.1 - test avg accuracy evaluation by model

model	accuracy
Naïve Bayes	75.70% (+/- 1.58%)
1 hidden layer	43.93% (+/- 5.18%)
2 hidden layers with word embeddings	79.88% (+/- 1.30%)
1 hidden layer with Glove pre-trained embeddings	77.78% (+/- 1.90%)
one dimensional Convolutional Neural Network	80.12% (+/- 0.85%)

table 5.1 - models evaluation

In hindsight, one main factor in this study is the dataset. News headlines can sometimes be subjective and difficult to classify, and its classification always suffers from some kind of personal bias. Although there are studies that found it not to be relevant [6], we could eventually assess the relevance of including the analysis of the complete news article content into the observation classification.

In the training of the deep learning models some parameters could still be further tested, based on the possible combinations we have available to setup the Keras model, as for instance the number of hidden layers and its dimension, and even using a different set of learning and activation functions, with additional experimenting we can eventually improve the performance of the models.

The Keras library has experienced strong growth in adoption by the academic researchers and companies alike, its momentum is reflected on the amount of available layers and functions in the framework, its usage presumes a sound knowledge of the theoretical basis

of deep learning but also the value that one can extract from it derives from the experience one develops in leveraging it in different tasks and settings.

This is obviously a learning process for the author of this study, and one main goal in future studies would be to develop a stronger understanding of the building blocks of neural networks, and in the scope of the present study, namely to explore further the design of convolutional neural networks, for instance, how including additional dimensions in the input data, as the chronological sequence of text, might affect this specific task.

Once we have developed an optimal model for the current study, it can become a complement for a broader study scope, as an example, to study the available correlation of the news headlines sentiment polarity and stock market trends, and use it as an augmentation feature for the analysis of time-series, eventually doing deep learning analysis of sequences.

References and Bibliography

[1] Lina M. Rojas-Barahona, "Deep Learning for Sentiment Analysis", December 2016 [Online]. Available: https://www.researchgate.net/publication/312188963_Deep_Learning_for_Sentiment_Analysis

[2] Reshma U, Barathi Ganesh H B, Mandar Kale, Prachi Mankame and Gouri Kulkarni, "Deep Learning for Digital Text Analytics : Sentiment Analysis", Apr 2018 [Online]. Available: https://www.researchgate.net/publication/325413224_Deep_Learning_for_Digital_Text_Analytics_Sentiment_Analysis

[3] Erion Çano, Maurizio Morisio, "A Deep Learning Architecture for Sentiment Analysis", April 2018 [Online]. Available: https://www.researchgate.net/publication/323243005_A_deep_learning_architecture_for_sentiment_analysis

[4] D. K. Kirange1, Ratnadeep R. Deshmukh, "Sentiment Analysis of News Headlines for Stock Price Prediction", Marcg 2016 [Online]. Available: https://www.researchgate.net/publication/299536363_Sentiment_Analysis_of_News_Headlines_for_Stock_Price_Prediction

[5] Sahar Sohangir, Dingding Wang, Anna Pomeranets and Taghi M. Khoshgoftaar, "Big Data: Deep Learning for financial sentiment analysis", Journal of Big Data, December 2018 [Online]. Available: https://www.researchgate.net/publication/322712095_Big_Data_Deep_Learning_for_financial_sentiment_analysis

[6] Vineet John, Olga Vechtomova, "UW-FinSent at SemEval-2017 Task 5:Sentiment Analysis on Financial News Headlinesusing Training Dataset Augmentation", July 2017, [Online].

Available: https://www.researchgate.net/publication/318813994_Sentiment_Analysis_on_Financial_News_Headlines_using_Training_Dataset_Augmentation

[7] Francois Chollet, “Deep Learning with Python”, Manning Publications, 2018, New York, US, ISBN 9781617294433

[8] Peter Harrington, “Machine Learning in Action”, Manning Publications, 2012, New York, US, ISBN 9781617290183

[9] Gavin Hackeling, “Mastering Machine Learning with scikit-learn”, Packt Publishing, 2012, Birmingham, UK, ISBN 9781783988365

[10] Ian H. Witten, Eibe Frank, “Data Mining, Practical Machine Learning Tools and Techniques”, 2nd Edition, Elsevier Inc., 2005, San Francisco, CA, US, ISBN 0120884070

[11] “Neural networks”, video series [Online]. Available: https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi [Accessed: 3-Dec-2018]

[12] “A friendly introduction to Convolutional Neural Networks and Image Recognition”, video [Online]. Available: https://www.youtube.com/watch?v=2-Ol7ZB0MmU [Accessed: 3-Dec-2018]

[13] “Evaluate the Performance Of Deep Learning Models in Keras” [Online]. Available: https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/ [Accessed: 3-Dec-2018]

[14] “The Right Way to Oversample in Predictive Modeling” [Online]. Available: https://beckernick.github.io/oversampling-modeling/ [Accessed: 3-Dec-2018]

[15] “The art and science of dealing with imbalanced datasets” [Online]. Available: https://medium.com/@humansforai/the-art-and-science-of-dealing-with-imbalanced-datasets-209b448a11c5 [Accessed: 3-Dec-2018]

[16] “Neural Networks and Deep Learning” [Online]. Available: http://neuralnetworksanddeeplearning.com/ [Accessed: 3-Dec-2018]

[17] “An Intuitive Explanation of Convolutional Neural Networks” [Online]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/ [Accessed: 3-Dec-2018]

[18] “Introduction to Convolutional Neural Networks” [Online]. Available: https://rubikscode.net/2018/02/26/introduction-to-convolutional-neural-networks/ [Accessed: 3-Dec-2018]

[19] “A Gentle Introduction to Convolutional Neural Networks” [Online]. Available: https://sefiks.com/2017/11/03/a-gentle-introduction-to-convolutional-neural-networks/ [Accessed: 3-Dec-2018]

[20] “A gentle introduction to Image Recognition by Convolutional Neural Network” [Online]. Available: https://soprasteriaanalytics.se/2018/04/13/a-gentle-introduction-to-image-recognition-by-convolutional-neural-network/ [Accessed: 3-Dec-2018]

[21] Philippe Rémy's github Reuters-full-data-set repository page [Online]. Available: https://github.com/philipperemy/Reuters-full-data-set [Accessed: 3-Dec-2018]

[22] Keras framework page [Online]. Available: http://keras.io [Accessed: 3-Dec-2018]

[23] Global Vectors for Word Representation page [Online]. Available: https://nlp.stanford.edu/projects/glove [Accessed: 3-Dec-2018]

[24] This study code on github [Online]. Available: https://github.com/jtviegas/studies/tree/master/cit/deep_learning/code/analysis [Accessed: 3-Dec-2018]

[25] Artificial Neural Network : Beginning of the AI revolution [Online]. Available: https://hackernoon.com/artificial-neural-network-a843ff870338 [Accessed: 3-Dec-2018]

[26] Khuong An Vo, Tho T. Quan, Mao Nguyen, "Combination of Domain Knowledge and Deep Learning for Sentiment Analysis", November 2017, [Online]. Available:

https://www.researchgate.net/publication/320477455_Combination_of_Domain_Knowledge_and_Deep_Learning_for_Sentiment_Analysis

[27] Abdalraouf Hassan, Ausif Mahmood, "Deep Learning Approach for Sentiment Analysis of Short Texts", April 2017, [Online]. Available: https://www.researchgate.net/publication/317701706_Deep_Learning_approach_for_sentiment_analysis_of_short_texts

Appendix A - code

sentiment.py

```
from keras import layers
from keras.layers import Flatten, Dense, LeakyReLU, PReLU
from keras.layers import Embedding, Dropout
from keras import utils
from keras.models import Sequential
from sklearn.model_selection import StratifiedKFold
import numpy as np

import opsio
import opsdata
import opsplot


def train_and_evaluate_model(x, y, model_function, model_parameters, epochs=8, batch_size=256,
plot=False):
    metrics = ['acc', 'recall', 'precision', 'f1_score']

    kfold = StratifiedKFold(n_splits=10, shuffle=True)
    fold_scores = []

    for train, test in kfold.split(x, y):
        x_oversampled, y_oversampled = opsdata.oversample(x[train], y[train])
        model = model_function(model_parameters)
        model.summary()
        model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
                      metrics=['acc', opsdata.recall, opsdata.precision, opsdata.f1_score])
        history = model.fit(x_oversampled, utils.to_categorical(y_oversampled, num_classes=3),
                            epochs=epochs,
                            batch_size=batch_size)
             scores  =  model.evaluate(x[test],  utils.to_categorical(y[test],  num_classes=3),
verbose=0)
        score = []
        for i in range(0, len(metrics)):
            print("%s: %.2f%%" % (model.metrics[i], scores[i + 1] * 100))
            print("_____________________________________________________________")
            score.append(scores[i + 1] * 100)
        fold_scores.append(score)

    final_scores = np.asarray(fold_scores)
    print("_____________________________final scores_______________________________")
    for i in range(0, len(metrics)):
        print("%s: %.2f%% (+/- %.2f%%)" % (
            metrics[i], np.mean(final_scores[:, i]), np.std(final_scores[:, i])))

    # plot the latest trained model as a sample
    if plot:
        opsplot.plot_model_history(history, metrics, validation=False)

    return model, history


def model_cnn_1d(params):
    model = Sequential()
                        model.add(Embedding(params['dict_size'],      params['embedding_size'],
input_length=params['sentence_size']))
```

```
    model.add(layers.Conv1D(8, 24, activation='relu'))
    #model.add(layers.Conv1D(8, 24, activation='relu'))
    #model.add(layers.MaxPooling1D())
    #model.add(layers.Conv1D(128, 12, activation='relu'))
    #model.add(layers.Conv1D(128, 12, activation='relu'))
    model.add(layers.GlobalMaxPooling1D())
    #model.add(Dropout(0.5))
    model.add(Dense(3, activation='softmax'))
    return model


def model_one_hidden_with_glove_embeddings(params):
                embedding_matrix    =    opsio.get_glove_world_embeddings(params['glove_dir'],
params['dict_size'], params['word_index'])
    model = Sequential()
                        model.add(Embedding(params['dict_size'],      embedding_matrix.shape[1],
input_length=params['sentence_size']))
    model.add(Flatten())
     model.add(Dense(1024, activation=PReLU(alpha_initializer='zeros', alpha_regularizer=None,
alpha_constraint=None, shared_axes=None)))
    model.add(Dense(3, activation='softmax'))
    model.summary()
    model.layers[0].set_weights([embedding_matrix])
    model.layers[0].trainable = False
    return model


def model_two_hidden_with_embeddings(params):
    model = Sequential()
                        model.add(Embedding(params['dict_size'],      params['embedding_size'],
input_length=params['sentence_size']))
    model.add(Flatten())
    model.add(Dense(32, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    model.summary()
    return model


def model_one_hidden(params):
    model = Sequential()
    model.add(Dense(32, activation='relu', input_shape=(params['sentence_size'],)))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    model.summary()
    return model


def  run_experiment(data,  option=4,  dict_size=25000,  sentence_size=100,  embedding_size=128,
glove_dir=None):

    x, y, dummy, dummy, dummy, dummy, word_index = \
                        opsdata.load_dataset(data,  dict_size,  sentence_size,  test_ratio=0.0,
val_ratio=0.0, oversampling=False)

       params  =  {"sentence_size":  sentence_size,  "dict_size":  dict_size,  "embedding_size":
embedding_size,
              "glove_dir": glove_dir, "word_index": word_index}

    if option == 1:
        func = lambda parameters: model_one_hidden(parameters)
    elif option == 2:
        func = lambda parameters: model_one_hidden_with_embeddings(parameters)
    elif option == 3:
        func = lambda parameters: model_one_hidden_with_glove_embeddings(parameters)
    elif option == 4:
        func = lambda parameters: model_cnn_1d(parameters)
```

```
        train_and_evaluate_model(x,  y,  func,  params,  epochs=epochs,  batch_size=batch_size,
plot=True)


if __name__ == "__main__":
    try:

        GLOVE_DIR = '/home/jtviegas/Documents/github/studies/cit/deep_learning/data/glove.6B'
        DATA = 'data.csv'

        DICT_SIZE = 25000
        SENTENCE_SIZE = 100
        EMBEDDING_SIZE = 64
        epochs = 16
        batch_size = 256

        run_experiment(DATA, 1, DICT_SIZE, SENTENCE_SIZE, EMBEDDING_SIZE, GLOVE_DIR)

        print('done')

    except ValueError as e:
        print("sorry there was a processing exception:", e)
```

opsdata.py

```
import numpy as np

import pandas as pd

from imblearn.over_sampling import RandomOverSampler

from keras import backend as kback

import opsio

import opstext



def shuffle_and_split_data(x, y, testing_fraction, validation_fraction=0.0):

   d = np.concatenate((x, y), axis=1)

   np.random.shuffle(d)

   variables = d.shape[1]

   y = np.asarray([[o] for o in d[:, -1]])

   x = d[:, :variables-1]



   num_samples = d.shape[0]
```

```
    testing_samples = int(num_samples * testing_fraction)
    total_training_samples = num_samples - testing_samples
    validation_samples = int(total_training_samples * validation_fraction)
    training_samples = total_training_samples - validation_samples
    x_train = x[:training_samples]
    y_train = y[:training_samples]
    if testing_samples > 0:
        x_test = x[training_samples + validation_samples:]
        y_test = y[training_samples + validation_samples:]
        print('Shape of x_test tensor:', x_test.shape)
        print('Shape of y_test tensor:', y_test.shape)
    else:
        x_test = None
        y_test = None
    if validation_samples > 0:
        x_validation = x[training_samples: training_samples + validation_samples]
        y_validation = y[training_samples: training_samples + validation_samples]
        print('Shape of x_validation tensor:', x_validation.shape)
        print('Shape of y_validation tensor:', y_validation.shape)
    else:
        x_validation = None
        y_validation = None
    print('Shape of x_train tensor:', x_train.shape)
    print('Shape of y_train tensor:', y_train.shape)
    return x_train, y_train, x_validation, y_validation, x_test, y_test


def oversample(x, y):


    ys = pd.DataFrame(y)
```

```
    print(ys[0].value_counts())
    sm = RandomOverSampler()
    _x, _y = sm.fit_sample(x, y[:, 0])
    ys = pd.DataFrame(_y)
    print(ys[0].value_counts())
    return _x, _y


def load_dataset(data, dict_size, sentence_size, test_ratio=0.3, val_ratio=0.0,
oversampling=False, use_sequences=True):
    x, y = opsio.get_x_and_y(data)
    x = opstext.lemmatize_it(opstext.remove_stopwords(
        opstext.lowercase(opstext.remove_apostrophes(
            opstext.remove_punctuation(x)))))


     result_sequences, word_index, result_one_hot_encoding = opstext.vectorize(x, dict_size,
sentence_size)
    if use_sequences:
        x = result_sequences
    else:
        x = result_one_hot_encoding
    x_train, y_train, x_validation, y_validation, x_test, y_test = shuffle_and_split_data(x, y,
test_ratio, val_ratio)
    if oversampling:
        x_train, y_train = oversample(x_train, y_train)
    return x_train, y_train, x_validation, y_validation, x_test, y_test, word_index


def load_data(data, dict_size, sentence_size):
    x, y = opsio.get_x_and_y(data)
    x = opstext.lemmatize_it(opstext.remove_stopwords(
        opstext.lowercase(opstext.remove_apostrophes(
```

```
            opstext.remove_punctuation(x)))))

    word_sequence_matrix, word_index, sentence_word_matrix = opstext.vectorize(x, dict_size,
sentence_size)


   return y, sentence_word_matrix, word_sequence_matrix, word_index


"""
Keras 1.0 metrics.
This file contains the precision, recall, and f1_score metrics which were
removed from Keras by commit: a56b1a55182acf061b1eb2e2c86b48193a0e88f7
"""
def precision(y_true, y_pred):
   """Precision metric.
   Only computes a batch-wise average of precision. Computes the precision, a
   metric for multi-label classification of how many selected items are
   relevant.
   """
   true_positives = kback.sum(kback.round(kback.clip(y_true * y_pred, 0, 1)))
   predicted_positives = kback.sum(kback.round(kback.clip(y_pred, 0, 1)))
   _precision = true_positives / (predicted_positives + kback.epsilon())
   return _precision


def confusion(y_true, y_pred):
   y_pred_pos = kback.round(kback.clip(y_pred, 0, 1))
   y_pred_neg = 1 - y_pred_pos
   y_pos = kback.round(kback.clip(y_true, 0, 1))
   y_neg = 1 - y_pos
   tp = kback.sum(y_pos * y_pred_pos) / kback.sum(y_pos)
   tn = kback.sum(y_neg * y_pred_neg) / kback.sum(y_neg)
```

```
    return {'true_pos': tp, 'true_neg': tn}


def specificity(y_true, y_pred):
    y_pred_pos = kback.round(kback.clip(y_pred, 0, 1))
    y_pred_neg = 1 - y_pred_pos
    y_pos = kback.round(kback.clip(y_true, 0, 1))
    y_neg = 1 - y_pos
    _specificity = kback.sum(y_neg * y_pred_neg) / (kback.sum(y_neg) + kback.epsilon())
    return _specificity



def recall(y_true, y_pred):
    """Recall metric.
    Only computes a batch-wise average of recall. Computes the recall, a metric
    for multi-label classification of how many relevant items are selected.
    """
    true_positives = kback.sum(kback.round(kback.clip(y_true * y_pred, 0, 1)))
    possible_positives = kback.sum(kback.round(kback.clip(y_true, 0, 1)))
    _recall = true_positives / (possible_positives + kback.epsilon())
    return _recall


def f1_score(y_true, y_pred):
    """Computes the F1 Score
    Only computes a batch-wise average of recall. Computes the recall, a metric
    for multi-label classification of how many relevant items are selected.
    """
    p = precision(y_true, y_pred)
    r = recall(y_true, y_pred)
    return (2 * p * r) / (p + r + kback.epsilon())
```

opsio.py

```
import csv

import unidecode

import numpy as np

import os



def get_x_and_y(data_csv):

   x = []

   y = []

   line = 0

   with open(data_csv, 'r') as file:

       reader = csv.reader(file, quotechar='"')

       for row in reader:

           if '' == row[1]:

               y.append([float(0)])

           else:

               y.append([float(row[1])])

           _as_string = unidecode.unidecode(row[0].decode('ascii', 'ignore'))

           x.append([_as_string])


           line = line + 1


   print('processed {} lines'.format(line))

   return np.asarray(x), np.asarray(y)
```

```
def get_glove_world_embeddings(glove_dir, max_dict_words, word_index):
   embeddings_index = {}
   glove_embedding_dim = 100
   glove_embedding_file = 'glove.6B.100d.txt'
   f = open(os.path.join(glove_dir, glove_embedding_file))
   """generate index from embeddings file"""
   for line in f:
       values = line.split()
       word = values[0]
       embeddings_index[word] = np.asarray(values[1:], dtype='float32')
   f.close()
   print('Found %s word vectors.' % len(embeddings_index))

   embedding_matrix = np.zeros((max_dict_words, glove_embedding_dim))
   for word, i in word_index.items():
       if i < max_dict_words:
           embedding_vector = embeddings_index.get(word)  # get its embedding vector
           if embedding_vector is not None:  # create the matrix with it
               embedding_matrix[i] = embedding_vector
   return embedding_matrix
```

opstext.py

```
import numpy as np
from nltk.corpus import stopwords
import re
import string
from keras.preprocessing.text import Tokenizer
from keras import preprocessing
```

```
from nltk import word_tokenize

from nltk.stem.wordnet import WordNetLemmatizer

from nltk import pos_tag


def remove_stopwords_from_str(s):

   r = []

   for x in s.split():

       if x.strip() not in stopwords.words('english'):

           r.append(x.strip())

   return [' '.join(r)]


def remove_apostrophes_from_str(s):

   r = []

   for x in s.split():

       _x = x.strip()

       if re.match('\w+\'.', _x) is not None:

           _x = _x[:-2]

       if re.match('\'.$', _x) is None:

           r.append(_x)

   return [' '.join(r)]


def remove_punctuation_from_str(s):

   r = []

   for x in s.split():

       if x.strip() not in string.punctuation:

           r.append(x.strip())

   return [' '.join(r)]


def lemmatize(token, tag):
```

```
    lemmatizer = WordNetLemmatizer()
    if tag[0].lower() in ['n', 'v']:
        r = lemmatizer.lemmatize(token, tag[0].lower())
        return r
    return token


def lemmatize_it(x):
    tagged_corpus = [pos_tag(word_tokenize(document[0])) for document in x]
    return [[lemmatize(token, tag) for token, tag in document] for document in tagged_corpus]


def remove_stopwords(x):
    return [remove_stopwords_from_str(k) for k in x[:, 0]]


def remove_apostrophes(x):
    return [remove_apostrophes_from_str(k) for k in x[:, 0]]


def remove_punctuation(x):
    x = [remove_punctuation_from_str(k) for k in x[:, 0]]
    return np.asarray(x)


def lowercase(x):
    _x = [[o[0].lower()] for o in x]
    return np.asarray(_x)


def vectorize(x, word_dict_len, sentence_words_max):
    sentences_list = x
    tokenizer = Tokenizer(num_words=word_dict_len)
    tokenizer.fit_on_texts(sentences_list)
    sequences = tokenizer.texts_to_sequences(sentences_list)
```

```
    word_index = tokenizer.word_index
    print('Found %s unique tokens.' % len(word_index))
    data = preprocessing.sequence.pad_sequences(sequences, maxlen=sentence_words_max)
    return data, word_index
```

opsplot.py

```
import matplotlib.pyplot as plt
import numpy as np
def plot_model_history(history, metrics, validation=False):
    metrics.append('loss')
    fig = plt.figure()

    for i, name in enumerate(metrics):
        metric = history.history[name];
        epochs = range(1, len(metric) + 1)
        p = fig.add_subplot(2, 3, i + 1)
        p.plot(epochs, metric, 'bo', label='Training ' + name)

        if validation:
            metric_validation = 'val_' + name
            validation = history.history[metric_validation]
            p.plot(epochs, validation, 'b', label='Validation ' + name)

        p.set_title('metric: ' + name)
        p.legend(loc=8)
    plt.show()
```

naive.py

```
import numpy as np

from math import log

from sklearn.model_selection import StratifiedKFold

import opsdata


def classify(x, word_class_probabilities, class_probabilities, classes):

   classification = None

   result=0

   for i in range(len(word_class_probabilities)):

       val = sum(x * word_class_probabilities[i]) + log(class_probabilities[i])

       if (classification is None) or (val > classification):

           classification = val

           result = i

   return classes[result]


def training(trainMatrix,trainCategory):

   n_docs = len(trainMatrix)

   n_words = len(trainMatrix[0])

   n_positives=0

   n_neutrals=0

   n_negatives=0

   word_count_positives = np.ones(n_words)

   word_count_neutrals = np.ones(n_words)

   word_count_negatives = np.ones(n_words)

   total_word_count_positives = 2.0

   total_word_count_neutrals = 2.0
```

```
    total_word_count_negatives = 2.0
    for i in range(n_docs):
        if trainCategory[i] == 1:
            n_positives+=1
            word_count_positives += trainMatrix[i]
            total_word_count_positives += sum(trainMatrix[i])
        else:
            if trainCategory[i] == -1:
                n_negatives += 1
                word_count_negatives += trainMatrix[i]
                total_word_count_negatives += sum(trainMatrix[i])
            else:
                n_neutrals += 1
                word_count_neutrals += trainMatrix[i]
                total_word_count_neutrals += sum(trainMatrix[i])
    positive_probability = n_positives/float(n_docs)
    neutral_probability = n_neutrals / float(n_docs)
    negative_probability = n_negatives / float(n_docs)
    positive_word_probability_vector = np.log(word_count_positives/total_word_count_positives)
    neutral_word_probability_vector = np.log(word_count_neutrals/total_word_count_neutrals)
    negative_word_probability_vector = np.log(word_count_negatives/total_word_count_negatives)
            return [positive_probability, neutral_probability, negative_probability],
[positive_word_probability_vector, neutral_word_probability_vector,
negative_word_probability_vector]


def testing(x, y, word_class_probabilities, class_probabilities, classes):
    total = len(x)
    t = 0.0
    for i in range(len(x)):
```

```
        classified = y[i]
        predicted = classify(x[i], word_class_probabilities, class_probabilities, classes)
        if classified == predicted:
            t += 1.0
    accuracy = t/total
    return accuracy


def train_and_evaluate_model(x, y):
    classes = [1, 0, -1]
    kfold = StratifiedKFold(n_splits=10, shuffle=True)
    accuracy_scores = []
    fold = 0
    for train, test in kfold.split(x, y):
        x_oversampled, y_oversampled = opsdata.oversample(x[train], y[train])
        class_probabilities, word_class_probabilities = training(x_oversampled, y_oversampled)
          accuracy = testing(x[test], y[test], word_class_probabilities, class_probabilities,
classes)
        accuracy_scores.append(accuracy*100)
        print("accuracy at fold %d: %.2f%%" % (fold, accuracy*100))
        fold += 1
    final_scores = np.asarray(accuracy_scores)
    print("_____________________________final accuracy score_______________________________")
    print("%.2f%% (+/- %.2f%%)" % (np.mean(final_scores), np.std(final_scores)))
if __name__ == "__main__":
    try:
        DICT_SIZE = 25000
        SENTENCE_SIZE = 100
        DATA = 'data.csv'
        y, sentence_word_matrix, word_sequence_matrix, word_index = \
```

```
            opsdata.load_data(DATA,DICT_SIZE,SENTENCE_SIZE)
        train_and_evaluate_model(sentence_word_matrix,y)
        print('done')
    except ValueError as e:
        print("sorry there was a processing exception:", e)
```