Machine Learning Engineer Nanodegree

Capstone Proposal

Giacomo Sarchioni
29th December 2017

Proposal

Domain Background

The domain background of the project is that of *sentiment analysis*, i.e. the field that "refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information"[1].
In particular, this project aims at classifying the **sentiment** of more than 500,000 Amazon Fine Food Reviews, by only looking at the raw text.

In most cases, sentiment is very difficult to measure. While more and more companies and business are employing proxy metrics (e.g. the *Net Promoter Score*[2]) to grasp sentiments, we, as human beings, still rely on language as the main way to express our feelings, emotions and, overall, sentiment. That is why there is so much interest in studying language and text.
It seems quite difficult to trace back a history of sentiment analysis[3], but prominent figures like Chris Manning already started to publish papers on the topic twenty years ago. In 2001, a paper called *Text Classification in a Hierarchical Mixture Model for Small Training Sets*[4] by K. Toutanova, F. Chen, K. Popat and T. Hoffman from Xerox PARC and Browne University showed one of the first attempts to classify text using count-based vectors and simple Bayesian models.

In general, traditional approaches to sentiment analysis rely on a count-based (absolute or relative) representation of text data. In this case, the degree of presence (or absence) of certain words - irrespectively of their ordering in the text - is the solely determinant factor in explaining the sentiment. Recent developments in the field of deep learning, however, allowed for two significant changes:

- different text data representation by using word vectorisation (e.g. Tomas Mikolov's Word2vec[5]);
- new model architectures by using approaches that take into account the ordering of words (e.g. Recurrent Neural Networks or RNNs).

Such changes aim at providing a more comprehensive representation of both words and their ordered sequence, by potentially unlocking a semantic-like understanding of text. In other words, text data does not become a simple count-based representation of keywords, but an actual - yet numerical - representation of meanings and contexts.

---

[1] More details on sentiment analysis: https://en.wikipedia.org/wiki/Sentiment_analysis

[2] More details on the Net Promoter Score: https://en.wikipedia.org/wiki/Net_Promoter

[3] More details on the history of sentiment analysis: https://www.quora.com/What-is-the-history-of-Sentiment-Analysis

[4] Original paper by Toutanova et al.: https://nlp.stanford.edu/pubs/toutanova2001text.pdf

[5] Original paper on Word2vec: https://arxiv.org/pdf/1301.3781.pdf

Problem Statement

The goal of this project is to evaluate the **performance of a deep learning-based classifier** in predicting the sentiment (measured indirectly as product's rating) of a given review in the Amazon Fine Food Reviews' dataset available in Kaggle[6]. The goal is to verify whether the semantic dimensions captured by deep learning enhancements (both in terms of data representation and model architecture) do allow for performance improvement - even if marginal.

Performance is measured through a standard classification metric called Area Under the Curve (AUC).

[6] More details on the Kaggle competition on Amazon Fine Food Reviews: https://www.kaggle.com/snap/amazon-fine-food-reviews and http://i.stanford.edu/~julian/pdfs/www13.pdf

Datasets and Inputs

The data is the Amazon Fine Food Reviews' dataset available on Kaggle. It consists of 568,454 fine food reviews. For every review, I am interested in the following two variables:

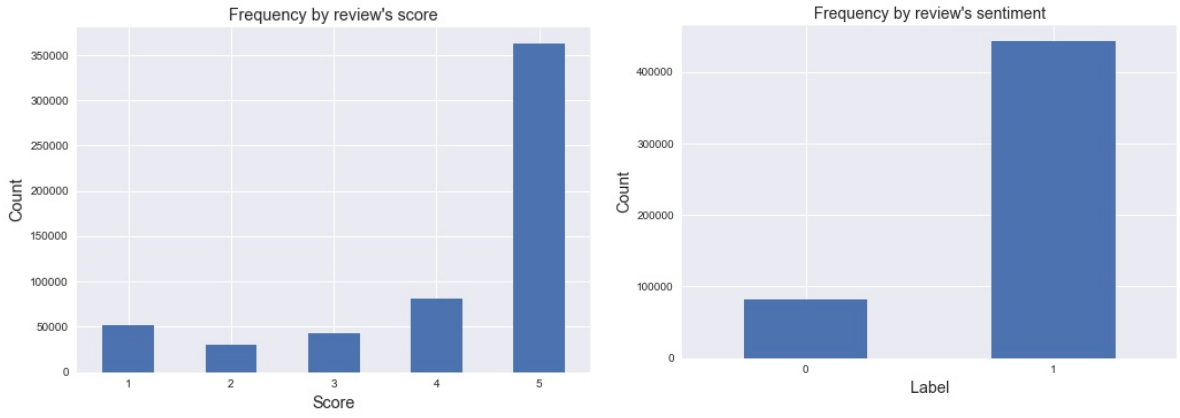
- *text*, i.e. the actual wording of the review;
- *score*, i.e. the rating given to the product (1-to-5 scale, where 1 is the lowest score and 5 is the highest).

Since I am only interested in evaluating reviews' sentiment (positive or negative), I have edited the dataset such as:

- 1- and 2-rated reviews are classified as negative (label 0);
- 4- and 5-rated reviews are classified as negative (label 1);
- 3-rated reviews are removed from the dataset.

The edited dataset then consists of 525,814 reviews (this includes 92% of the original reviews).

The charts below show the frequency distribution of all reviews by review score (on the left) and the frequency distribution of non-neutral review by sentiment (see above for details on how reviews are classified). It is quite evident how the dataset is imbalanced. Such issue is addressed by choosing appropriate performance metrics (see *Evaluation Metrics* paragraph) and appropriate weighting schemes in the models (see *Project Design* for more details)[7].



With regards to *text*, I use two possible representations:

- representation based on a count-based approach (i.e. a bag-of-words model[8]) for the non-semantic classifier;
- semantic representation based on a vector-based approach (i.e. word and paragraph embeddings[9]) for the semantic classifier.

---

[7] Ways to manage imbalanced datasets: https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

[8] More details on bag-of-words models: https://en.wikipedia.org/wiki/Bag-of-words_model

[9] More details on word embeddings: https://en.wikipedia.org/wiki/Word_embedding

Solution Statement

Text data will be fed to a classification model whose objective is that of correctly identifying the review sentiment, i.e. positive or negative.

The solution is easily quantifiable by comparing the predicted values on the test dataset versus the true labels. In addition, replication of the experiment is quite trivial, as long as new input text is formatted using the vocabulary and/or vectorisation model defined during training.

Benchmark Model

Since the objective of the project is to identify whether semantic representation and/or semantic processing of text data enhances the performance of a sentiment classifier, my benchmark model would, naturally, be a non-semantic one. In particular, I would define two benchmark models:

- a naive model, where the classifier always predicts a positive sentiment (given the fact that most of the reviews are positive);
- a bag-of-words classifier with tf-idf vectorisation[10].

For the second benchmark model, I would choose the algorithm with the highest performance (see *Evaluation Metrics* below) among the following:

- Naive Bayes;
- Decision Tree;
- Logistic Regression;
- Random Forest;
- AdaBoost;
- XGBoost;
- Multi-Layer Perceptron.

In the Kaggle's webpage for the Amazon Find Food Reviews' dataset (link), users have achieved different results in terms of performance. It is very difficult to identify one single benchmark value, since it all depends on how text has been pre-processed, how train and test data has been split and which random state (if any) has been set.

Therefore, in this context, I will compare the performance of my semantic-based classifier against the AUC score of the naive classifier (whose $AUC = 0.5$ since such model is equivalent to informed random guessing) and that of the bag-of-words classifier with tf-idf vectorisation (this model will be defined and evaluated in the project itself).

Pleas bear in mind that I am going to use the same pre-processed text throughout the entire project.

[10] Tf-idf vectorisation gives more importance to words that appear often in a review but are quite rare in the entire set of reviews. Those words should be more peculiar to a specific review and define it better. More details on tf-idf: https://en.wikipedia.org/wiki/Tf–idf.

Evaluation Metrics

Given the nature of the dataset, I have to choose a metric that can deal with the fact that classes are very imbalanced. A naive classifier that predicts all reviews to be positive would actually have a pretty high accuracy - which is, if not adjusted, a very bad metric for this project. A metric like AUC, on the contrary, is insensitive to the imbalanced dataset, so I am going to choose it as the main evaluation metric for this project.

AUC or Area Under the Curve represents the actual area under the Receiver Operating Characteristic curve (ROC curve). In statistics, such curve "illustrates the diagnostic ability of a binary classifier system as discrimination threshold is varied"[11]. The curve plots the True Positive Rate (TPR) of a classifier, against its False Positive Rate (FPR), for multiple discrimination thresholds. Such TPR-FPR combinations are used to plot a line (or a curve); the AUC metric is the area under such curve.

True Positive Rate is defined as:

$$TPR = \frac{TP}{P}$$

where TP is the number of true positives, and P is the number of positive samples (TPR actually corresponds to *recall*).
False Positive Rate is defined as:

$$FPR = \frac{FP}{N}$$

where FP is the number of false positives, and N is the number of negative samples.

It follows, quite intuitively, that TPR should be as high as possible (maximum value equal to 1), while FPR should be as low as possible (minimum value equal to 0).

The ROC curve is built by varying the discrimination threshold. Let me give a quick example with the Fine Food Review's dataset, using the naive classifier (i.e. the algorithm that always predicts a review to be positive). In the table below, I present the labels for two discrimination threshold values λ. Please notice that the label is assigned as follows:

$$label = \begin{cases} 1 & \text{if } \hat{y} \geq \lambda \\ 0 & \text{otherwise} \end{cases}$$

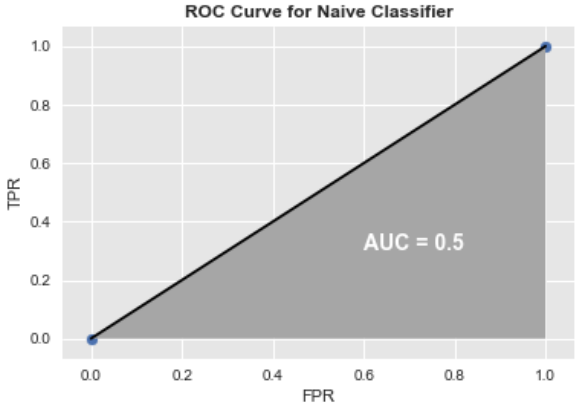
where $\hat{y}$ is the label predicted by the model.

Review	Predicted sentiment	Label for $\lambda = 1$	Label for $\lambda = 2$
1	1	1	0
2	1	1	0
3	1	1	0
…	1	1	0
n	1	1	0

[11] More details on ROC: https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve

In the case of $\lambda = 1$ the label attributed to every review would be 1, while in the case of $\lambda = 2$ the label attributed to every review is 0 (i.e. negative). When $\lambda = 1$, the TPR is equal to 1, since all actual positive reviews are classified as such by the model. In addition, the FPR would also be 1, since none of the negative reviews are classified as such, but they are all falsely classified as positive. It is easy to verify how, in the case of $\lambda = 2$, where all predicted labels are 0, both TPR and FPR are 0.

In the ROC plot, this means I have two points (1,1) for $\lambda = 1$ and (0,0) for $\lambda = 2$. The chart below shows how the AUC for such the naive classifier is 0.5.



Contrarily to accuracy, AUC is quite robust to the presence of an imbalanced dataset and even if I use a naive classifier that always predicts the sentiment of the most frequent sentiment label, it still outputs an AUC value of 0.5 (while accuracy would indeed be approximately 84%). An AUC score of 0.5 is equivalent to random guessing. In our case, that would be equivalent to tossing a biased coin, where the probability of 1 is equal to 84% (i.e. the frequency of the most represented class)[12].

Throughout the project, I will also look at class-weighted (i.e. taking into account performance on both positive and negative labels) metrics such as precision, recall and f-beta score (in *scikit-learn* this class-based weighting scheme is applied by setting the parameter *average* equal to *macro*[13] in any of the metric functions). However, AUC will remain the reference performance metric.

[12] More details on AUC score: https://ashokharnal.wordpress.com/2014/03/14/a-very-simple-explanation-for-auc-or-area-under-the-roc-curve/

[13] See example of *macro* averaging for *f1_score* in *scikit-learn* at http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

Project Design

The project follows a very structured design. In particular, I have identified three macro sections:

1. Data pre-processing;
2. Train-validation-test split;
3. Modelling.

1. Data pre-processing

In this step I clean the text input by performing some common pre-processing operations. In particular, I do the following:

- remove HTML tags and other text (if any). I use the module *Beautiful Soup*[14]*;*
- transform all text in lower case;
- remove numbers - in general numbers should not convey any sentiment-related information[15];
- remove punctuation;
- remove English stop words (e.g. “the”, “of”, “and”, etc.);
- remove leading and trailing whitespaces.

I may also use a technique called *stemming* which “removes morphological affixes from words, leaving only the word stem”[16]. The usage of such technique exclusively depends on the eventual recording of an improved performance in the bag-of-words benchmark model.

The approach followed in data pre-processing may be a bit too restrictive. Punctuation may in fact include possible information related to sentiment (e.g. multiple exclamation marks in a review may indicate a strong sentiment, possibly positive or negative). For **simplicity**, however, I have followed the approach of removing punctuations and concentrate on words only (this is in line with a well-known tutorial on NLP available in Kaggle - link here).

2. Train-validation-test split

This is a crucial point. It is in fact essential to have:

- a training set on which the model can learn;
- a validation set to fine-tune the model (i.e. adjust parameters value);
- a test set to evaluate.

There are two main elements to take into account here:

1) Preservation of imbalanced data across the three datasets;
2) Identification of which dataset/s to use in order to define the word vocabulary;

Since our dataset is **imbalanced** (i.e. more positive reviews than negative reviews), it is important to - at least - have a validation set that maintains a similar class proportion. In the fine tuning process of every model, having a similarly imbalanced dataset would allow for a more complete

---

[14] More details on Beautiful Soup: https://www.crummy.com/software/BeautifulSoup/

[15] Actually, an expression like “number 1” may imply a positive sentiment. By removing numbers, such information will be lost in the input data. Please note that, however, the expression “number one” will be kept.

[16] More details on stemming: http://www.nltk.org/howto/stem.html

learning process (i.e. I am sure that when I am fine-tuning each model on the validation set, I let it "see" kind of the same data, i.e. an imbalanced set of food reviews).
In theory, it is **not necessary** to have a similarly imbalanced test set. Suppose, in fact, we have a test set entirely made of positive reviews. A good classification model is one that has the lowest possible number of false negatives (i.e. the model should not predict any negative review). To make everything consistent and let me audit the performance of the model/s, however, I will include negative reviews in the test dataset as well (again with a similar class proportion).
In order to perform such dataset splitting, I use the *scikit-learn* module called *StratifiedShuffleSplit*[17]; this would in fact me allow to preserve the percentage of samples for each class.

In order to work with text data, I need to **define a word vocabulary**. This corresponds to the words our model "sees". Therefore it is important that such vocabulary is built **excluding** any text in the test dataset. This means that if a review in the test dataset contains a word/s not included in the vocabulary, the model would simply ignore such input. I am going to only use **the train and validation sets** to build the word vocabulary, and format the test set using such vocabulary only at a later stage.

3. Modelling
First of all, I need to identify a bag-of-words classifier to use as my second benchmark (in addition to the naive model).
I will use the module *RandomizedSearchCV*[18] from *scikit-learn* to fine-tune each model tested (see above for a list of algorithms).
For all models trained using *scikit-learn*, in order to counterbalance the effects due to an imbalanced dataset, I will set the parameter *class_weight* to *balanced* so that similar importance is given to all classes.

Secondly, I will concentrate on the core objective of the process, i.e. semantic-based implementations. In particular I am going to use three different models:

1. **Word vectorisation:** I am going to transform input data using Word2vec vectorisation technique. This implies training a Word2vec model and transform all words in every review into their vector equivalents[19]. After that, following the example provided in the Kaggle's NLP tutorial (link to part 3 here), I will train a classifier using the vector averages in every review.
2. **Document vectorisation:** soon after releasing his paper on word vectorisation, Mikolov et al.[20] published a paper that inspired the so-called *Doc2vec* model. Similarly to *Word2vec*, such model provides a vector-based representation of an entire sentence. According to a very well-written document by Gidi Shperber[21], such approach should improve the performance of

---

[17] More details on *StratifiedShuffleSplit*: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html
Please notice that in order to split the data only one time I need to set *n_splits* equal to 1.

[18] More details on RandomizedSearchCV: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

[19] Depending on time required to train the model, I may use the built-in module called *Embeddings* available in Keras. Such module lets me create a vector-based representation of input data.

[20] Original paper on Doc2vec: https://arxiv.org/pdf/1405.4053.pdf

[21] Link to article by Gidi Shperber on Medium: https://towardsdatascience.com/a-gentle-introduction-to-doc2vec-db3e8c0cce5e

my classifier versus the strategy presented above (i.e. vector averaging). Therefore I will train a *Doc2vec* model to represent my input data (i.e. my reviews) as vectors and will feed them to one of the classifiers defined previously;

3. **Recurrent Neural Networks (RNNs):** using the Word2vec input data defined above, I am going to train a Recurrent Neural Network classifier. A RNN is capable of learning sequential data. The RNN "reads" each word after the other and is able to retain information of the words appearing in the sequence of text - it almost builds a memory-based representation of the context. It is not by chance that one of the most commons implementation of RNNs is an architecture called Long Short-Term Memory, or LSTM[22]. In short, RNNs allow to keep some of the information learned from one word and maintain it throughout the process of learning from all the following words. In this project, such RNN architecture is the only one capable of **making sense of words' sequencing**. I plan to implement a multi-layered LSTM architecture to define a RNN classifier.

Pleas bear in mind that RNNs input data must have the same dimension (i.e. reviews must have the same length). Let me call *k* the value representing the length (in terms of number of words) each review should have. If a review has more than *k* words, I will truncate the review at the k^{th} word; if a review has less than k words, I will pad the missing words with zeros. Such value *k* would be an important parameter when it comes to fine-tuning the model.

I will then compare the performance of each of the models defined above versus the benchmarks, using the predictions generated on the test set.
For training the vectorisation algorithms as well as the RNN I plan to use a GPU-enabled instance available on AWS.

[22] More details on LSTM cells: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Project diagram

The diagram below illustrates a high-level representation of the steps expected to be completed throughout the process.