Machine Learning Engineer Nanodegree

Capstone Proposal

Steffen Bunzel
October 14th, 2018

Proposal

Domain Background

The capstone project I propose here is in the area of automated text classification. This machine learning task seeks to classify documents into predefined categories. In particular, the area of focus is "single-lable binary classification" where each document has exactly one of two possible labels. Text classification has been an area of active research for years. The increasing amount of digital information, which is impossible to process for humans without being prefiltered, has spurred even more interest in the area.

Jindal et al. (2015) reviewed publications on automated text classification and compared existing approaches. These include methods to preprocess and represent documents in numerical form, transform them into a lower dimensional feature space, apply machine learning algorithms on these features and evaluate their performance. Among others, their review shows two things: First, a text classification pipeline involves more than just training a machine learning model, but requires careful thinking about how to preprocess and represent the text data. Second, the machine learning methods applied are manifold ranging from naive-bayes and k-nearest neighbors through tree-based models and support vector machines to neural networks. The latter class of models has just recently showed outstanding results when being combined with the paradigm of transfer learning (cp. Howard and Ruder, 2018 as well as Radford et al., 2018). In particular these approaches use unsupervised pretraining in the form of generative language models and fine-tune the resulting models for the task of text classification, among others.

Having observed the power of transfer learning in the field of computer vision myself, I am intrigued by its potential for text processing. On the other hand, I have a lot of respect for the complexities that working with natural language entails and I am curious to learn more about them through this project.

Problem Statement

I love to read blog posts on machine learning (ML) and artificial intelligence (AI) on medium.com, a popular online publishing platform. However, not all of the articles available there fit my taste. Unfortunately, I sometimes find this only after having read the article. In particular, I do not enjoy articles which are superficial and aim to benefit from the hype around ML and AI, thereby confusing beginners. On the other hand, I enjoy articles offering a deep coverage of technical concepts and are written in a way that keeps the reader engaged nevertheless. As my "blog post taste" seems to follow certain patterns, I am curious whether I can use ML to build a classfier that indicates how likely a given article will fit my taste and thus make recommendations on which I will actually enjoy. Therefore, the problem statement is the following:

> Can I train an ML-based classifier that distinguishes whether I will like a blog post on ML / AI published on medium.com only using the blog's text as an input?

Datasets and Inputs

To tackle this problem, a dataset of ML and AI blogposts published on medium.com is needed.
Luckily, a Kaggle user has already provided a collection of such data on the platform. This dataset was scraped from medium.com and made avaible under the `CC0: Public Domain license, i.e. as a work in the public domain without copyright` license. The dataset contains the article's author, its estimated reading time, a link to the article, its title, its text body as well as the number of claps (a form of like) it has received to date.

The next step to make this data usable for my project was to clean and label it. While the raw data contained 337 rows, I found that it included several duplicates, leaving me with 230 unique articles. Of these, six were not written in English, thus reducing the number to 224. As a first shot, I have labeled 100 of these articles and made the labels available through my GitHub.

This data (articles and labels) will be the input to further preprocessing and creating the text classifier. For this work, I will only use the text body as a feature as my vision is to build a classifier that works on just the text and is universally usable even without access to the metadata.

Solution Statement

The proposed solution, which I plan to evaluate in this project, involves building a pipeline for preprocessing the text body and using a machine learning model to classify it into one of two categories: "I find this article interesting" or "I do not find this article interesting". My goal here is two compare several approaches of increasing complexity (e.g. from term frequency + tree-based classifier to fine-tuned neural language model). To do so, I will make use of spaCy (https://spacy.io/), a Python library for natural language processing (NLP) and the pretrained language models that come with it.

Benchmark Model

A simple benchmark model for this task would be one that only uses the numerical metadata on each article, i.e. claps and reading time.

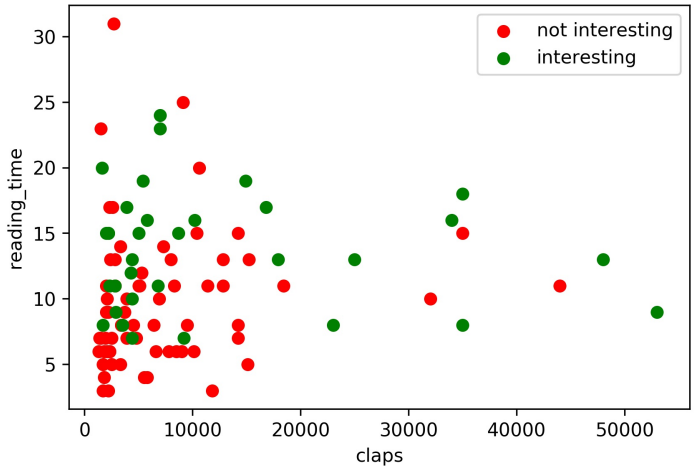

Figure 1: Classes by claps and reading time

From the scatter plot above it seems likely that my performance requirements cannot be reached using just these two features (unfortunately, the data does not contain the publication dates of the articles. Thus, it is not easy to calculate the average claps per day which would probably be a more useful feature than the absolute amount of claps. As my focus is on gathering experience with NLP, I chose not to spend time on collecting this data point for each article). For the final write up on my project, I will train a classifier based on these two inputs only and provide the performance as a benchmark for the text based classifier.

Evaluation Metrics

The metrics that will be used to assess the classifier's performance need to balance two aspects: As I have a limited capacity to read blog posts, I want to find the ones I am interested in with high precision. Thus, the first metric employed is precision:

```
precision = true positives / (true positives + false positives)
```

A **target precision of 95%** will serve as a guideline for this project, which means that I will only find one in twenty articles

recommended by the classifier not interesting.

On the other hand, I do not want to restrict the number of article recommendations I get too much. Therefore, I simultaneously aim to achieve a **recall of 75%**, which means that only one in four articles I would have found interesting will be classified as not interesting:

```
recall = true positives / (true positives + false negatives)
```

Project Design

The proposed project involves 8 main steps:

1. Obtaining the data

   Luckily, the heavy lifting has already been done for me here. The dataset available through Kaggle contains all data points I need and has more examples than I will most likely be able to label.

2. Cleaning and labeling the data

   As outlined above, the raw data from Kaggle contains several duplicates as well as blog posts in languages other than English. To work around these shortcomings, I have already set up a preprocessing script that uses pandas' `drop_duplicates` to remove duplicates and spacy_cld's `LanguageDetector` to filter on English articles. The remaining 224 articles will be part of my evaluation. To make this a supervised learning problem, I need to manually label these articles based on how interesting they seem to me. To do so, I will skim through each article for 2-5 minutes on medium.com (to have the actual formatting preserved) and rate whether I would want to read the full article or not. I have tested this approach for the first 100 articles and found that 2-5 minutes is enough time to assess my interest in a specific article. Even more importantly, I found that for several articles whose headline I didn't like (the quickest way to assess an article), I actually liked the article based on the 2-5 minute assessment - and vice versa.

3. Preprocessing the data

   As outlined above, NLP tasks usually require preprocessing steps before a model can be trained - a characteristic which I find particulary interesting about these tasks as the decisions made in preprocessing may heavily influence the model performance. What I already found when inspecting the data is that the texts contain several special characters (mostly emojis) and one decision I will have to make is whether to remove these. What's more, there are several different strategies for stemming words, removing common words, etc. During my initial experimentation, I saw that `spaCy` implements many of these strategies already and makes them available to the user quite conveniently. Nevertheless, I will need to decide on which to use and experiment with which performance differences they bring with them, if any.

4. Choosing the document representation strategy

   In addition to the preprocessing just described, text data needs to represented in numerical form before it can be used for modeling. Again, there are different strategies to do so. One important decision here is whether to represent the text on character or word level. Arguably the most widespread approach is to treat each document as a `bag-of-words`, i.e. "a collection of words which occur in it at least once" (Jindal et al., 2015, p.6).

5. Transforming the feature space

   Especially with the bag-of-words approach, one key challenge follows from the chosen representation: The number of unique words across all documents will be very large and with it the feature space will explode. Again, there are different approaches to tackle this: Before deep learning hit, NLP tasks were commonly approached by excplitly applying feature selection or transformation techniques, possibly in combination with a weighting method like `term frequency inverse document frequency (TFIDF)`, before handing the document-term-matrix to the learning algorithm (Jindal et al., 2015). In addition, common practice was hand-designing numerical features specific to the NLP task at hand (Collobert and Weston, 2008). In contrast, deep neural networks take in the raw numerical representation of a text and in the process of optimizing their weights for the learning task, create lower dimensional representations of the input documents. In this context, the term `word embeddings` emerged to describe these represenations (Ruder, 2016). I plan to experiment with and compare different

methods for transforming the feature space in this project.

6. Training a classifier

   After all of this work, it will finally be time to train a classifier. Depending on the choices I have made leading up to his, there will be several options here: On top of TFIDF matrix, all common model classes could be used (tree-based models, linear models, distance-based models, neural network models, …). When starting from the raw text in numerical form the applicability of different models depends on whether the input is transformed to have a static input length. In addition pretrained word embeddings or language models can be employed to extract features from the raw representation and train a classifier on top of those. In summary, this is where I plan to experiment as much as possible to see which differences in performance I find for these diverse approaches. My goal is to use at least one method from each of the following categories:

   - Raw numerical input + learning algorithm (logistic regression, tree-based, support vector machine, neural network)
   - TFIDF on raw numerical input + learning algorithm
   - Pre-trained word embeddings + learning algorithm
     (- Pre-trained language model with fine-tuned classification layer) - If time allows

7. Evaluating the classifier

   When I have found a good model, I will need to evaluate it on unseen data and compare the performance metrics to the goals I set above. To do so, I plan to label an additional 25 - 50 articles after I have tuned my model on a validation set that is part of the currently labeled data. This way, there will be no temptation to use the test data for any hyperparameter tuning in the model development.

8. Gathering lessons learned and next steps

   I am sure that after this I will have learned a ton. In particular, I expect to gain a better understanding of how the diverse preprocessing and modeling approaches in NLP work on a real life task and what the complexity-performance trade off looks for each of them. And of course, I would be very interested to have such an interest classifier for myself and would love to use it to preselect from the flood of ML and AI articles being published on medium.com every day!

Sources:

- *Collobert and Weston (2008): Collobert, Ronan; Weston, Jason (2008) A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. Proceedings of the 25th International Conference in Machine Learning. Helsinki, Finland.*
- *Jindal et al. (2015): Jindal, Rajni; Malhotra, Ruchika; Jain, Abha (2018) Techniques for Text Classification: Literature Review and Current Trends. Webology, 12(2), Article 139.*
- *Howard and Ruder (2018): Howard, Jeremy; Ruder, Sebastian (2018) Universal Language Model Fine-tuning for Text Classification, In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers), Melbourne, Australia, July 15 - 20, pp. 328–339.*
- *Radford et al. (2018): Radford, Alec; Narasimhan, Karthik; Salimans, Tim; Sutskever, Ilya (2018) Improving Language Understanding by Generative Pre-Training. Preprint available at: https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf (Last accessed: 2018-09-21).*
- *Ruder (2016): Ruder, Sebastian (2016) On Word Embeddings - Part 1. Available at: http://ruder.io/word-embeddings-1/ (Last accessed: 2018-10-14).*