A Kaggle competition: Categorizing web pages as evergreen or non-evergreen

Jeffrey De Fauw[*]
`jeffrey.defauw@ugent.be`

Abstract

This paper is the report of the author's participation in the Kaggle competition "StumbleUpon Evergreen Classification Challenge"[1] in which the goal was to build a classifier to categorize a web page as *evergreen* (long time relevance) or *non-evergreen* (short time relevance). Our "final model" consisted of a two layer ensemble approach with logistic regression models on different feature sets in the first layer, and an arithmetic mean in the second (combining the predictions). These feature sets were bag-of-words models with tf-idf weighting on different parts of the raw html content and some of their transformations by PCA (Principal Component Analysis). Even much simpler versions of this "final model" resulted in a strong performance in the competition. On the final leaderboard we were placed 8th of the more than 600 participants, despite entering the competition late.

1 Introduction

Kaggle[2] is a platform where companies can organize data prediction competitions, many of them open to the general public. The competition we competed in, the "StumbleUpon Evergreen Classification Challenge"[1], was organized by StumbleUpon, a discovery engine focussed on recommending personalized content to users, and featured a first prize of $5,000. The competition started on the 16th of August 2013 and ended on the 31st of October 2013, with more than 600 confirmed competitors.

The data consisted of 10566 URLs (or samples), 7395 of which were labelled *evergreen* or *not-evergreen*[3]. For completeness we include StumbleUpon's own description of these labels:

While some pages we recommend, such as news articles or seasonal recipes, are only relevant for a short period of time, others maintain a timeless quality and can be recommended to users long after they are discovered. In other words, pages can either be classified as "ephemeral" or "evergreen".

The remaining 3171 URLs constituted the test set for which the labels – evergreen or not-evergreen – were to be predicted and on which the final score, the AUC[4] metric, would be calculated. Of these 3171 URLs, 20% was chosen at random as a public test set (fixed for each competitor) to generate a preliminary public leaderboard during the competition.

[*]As project for the course "Machinaal Leren" taught by prof. dr. ir. B. Schrauwen at Ghent University in the academic year 2013–2014.

[1]`https://www.kaggle.com/c/stumbleupon`

[2]`https://www.kaggle.com`

[3]Also called *ephemeral*.

[4]Area Under Curve of the Receiver Operating Characteristic curve.

The raw data itself, as provided by the competition, consisted of the raw html crawled by StumbleUpon's crawler and 26 features, some of which were extracted explicitly from the page itself (e.g., average link size); others were first processed by StumbleUpon (e.g., if a URL was news or not, as determined by StumbleUpon's news classifier). We found the non-text features to offer only very small (if any) improvements and we will thus focus only on the text-features (the boilerplate and the raw html).

For most of the competition we used scikit-learn [1], a powerful and well-documented machine learning package for Python.

2 Data pre-processing

We will try to give a fairly comprehensive description of our methods for pre-processing the data since this stage – and in particular the feature extraction – was quite crucial in the competition.

Among the provided features was a boilerplate text of the URL, title and body containing words and numbers extracted from the respective parts of the page. We applied tf-idf transformations on each of these features separately with different parameters. The parameters we tried to optimize were:

- Preprocessing: whether to convert the text to lowercase first, remove parts of the text, etc.
- Tokenization: whether to use a certain stemmer on words during word tokenization.
- Analyser: whether to select words or character strings from the tokenized text.
- N-grams range: the range of n for the n-grams to be selected, e.g., a range of (1, 2) results in all words and sequences of two words to be extracted when using the word analyser. When using the character strings analyser this will select all characters and sequences of two characters.
- Minimum and maximum frequency: discard all terms in the vocabulary that have a term frequency lower (higher) than the minimum (maximum) frequency.[5]

We found that, when not using any dimensionality reduction techniques like LSA (Latent Semantical Analysis) or PCA (Principal Component Analysis), the last three parameters (the analyser, the n-grams range and the minimum/maximum frequency) and especially the minimum term frequency, can have a very significant influence on the score when using a logistic regression model. Although one can get very good results when optimizing these parameters for each part of the boilerplate separately, eventually we have chosen to select a reasonably sized (but large) fixed vocabulary of 1-grams and 2-grams of words from each part of the boilerplate and use dimensionality reduction on the resulting tf-idf matrices. We do this for several reasons:

- This eliminates the need to generate many different tf-idf matrices for different minimum and maximum frequencies, which can take longer than, for example, LSA.
- The main parameter of dimensionality reduction, the number of components to reduce to, is in a way much more "stable" then specifying the minimum term frequency: the term frequency distribution is highly dependent on the type of corpus while one can safely get very good results with general dimensionality reduction by selecting a number of components of $100k$, where k is an integer (not strictly) between 1 and 10 (most of the time even between 1 and 6).
- Without dimensionality reduction the total number of features will likely be (much) bigger than 10000. This implies that many typically well performing algorithms like random forests, extremely randomized trees, gradient boosting classifiers, etc., are off-limits; after dimensionality reduction these algorithms can be used effectively.
- Most importantly: the performance was at least comparable to carefully selecting the minimum frequency.[6]

[5]We initially used a restriction on the number of features in the vocabulary but thought this to be less preferable since this implies that words could be arbitrarily discarded despite having the same term frequency as some terms still in the vocabulary. Nevertheless, arguments involving a choice of minimum and maximum term frequencies, also apply to a choice of a maximum vocabulary size.

[6]The maximum frequency was not very important and had almost no impact on the vocabulary (above a relatively low boundary).

Before exploring dimensionality reduction further, we added features we extracted ourselves from the raw html, since only a limited amount of information was provided in the boilerplate (besides being only a selection of the text on the page, it also did not contain any links). We chose to use html2text[7] which converts the html to "plain text" and also keeps the links in the format *[link text](URL)*. This turned out to provide a significant improvement over just using the boilerplate (we estimate the general improvement to be around 0.5%, which in this competition was relatively big). Because we wanted to give more weight to the body text in the boilerplate – which is also contained in the raw html – we constructed a new feature set by first adding the body to the raw html before any transformations. This gave roughly double the improvement from simply using the html and was our strongest single feature set. In later models we also removed the links from the page first in order to process them separately (but this likely did not have a big influence).

This gave us 6 "raw text features": *URL*, *title*, *body* (from the boilerplate), *html* (without links), *links* and *htmlbody* (*html* concatenated with *body*), on which we applied separate and different tf-idf transformations.

Initially we used LSA for dimensionality reduction by applying truncated SVD (Singular Value Decomposition) to the tf-idf matrices. This seemed to work well – by which we mean that we got at least comparable performance to our best models without dimensionality reduction but with optimizing the minimum term frequency and the choice of n-grams – but soon switched to kernel PCA with a linear kernel, which is equivalent with (linear) PCA. We use PCA as kernel PCA with a linear kernel because:

- First of all, we implemented kernel PCA since it allowed us to explore different kernels, making non-linear dimensionality reduction possible.
- Secondly, this is mainly a technical point: "normal" linear PCA as implemented in scikit-learn does not support the sparse tf-idf matrices but kernel PCA with a linear kernel, equivalent with PCA, does.

As for our choice of PCA instead of LSA: small tests showed that PCA did not perform worse than LSA, nor did one of them seem noticeably more computationally demanding than the other.

In the end we had the following three *feature sets*:

1. Our original optimized tf-idf matrices using a combination of word n-grams and character n-grams.

- PCA-transformed tf-idf matrices using a different number of components for each of the six raw text features and with
  2. 1-grams of words,
  3. 2-grams of words.

In each feature set the text was first converted to lowercase and the Snowball stemmer from NTLK[8] was applied. The minimum frequency of the tf-idf transformations for the last two sets was chosen such that the vocabulary was reasonably sized (but relatively large).

3 The model(s)

Before searching for the best models for each of the feature sets, we implemented a general *two-layer ensemble approach*. First different models are trained on samples of different feature sets to predict the training set labels *and* the test set labels. After a chosen number of samples (of chosen size), a new model is trained on these "sampled predictions" of the training set labels to predict the test set labels (from the sampled predictions of the test set labels). A more detailed description is provided with Figure 1.

We implemented this for several reasons:

[7] `http://www.aaronsw.com/2002/html2text/`; We experienced some encoding issues but due to the time limit simply tried several encodings and automatically selected the one that worked (best).

[8] The Natural Language Toolkit, a collection of Python-libraries for Natural Language Processing.

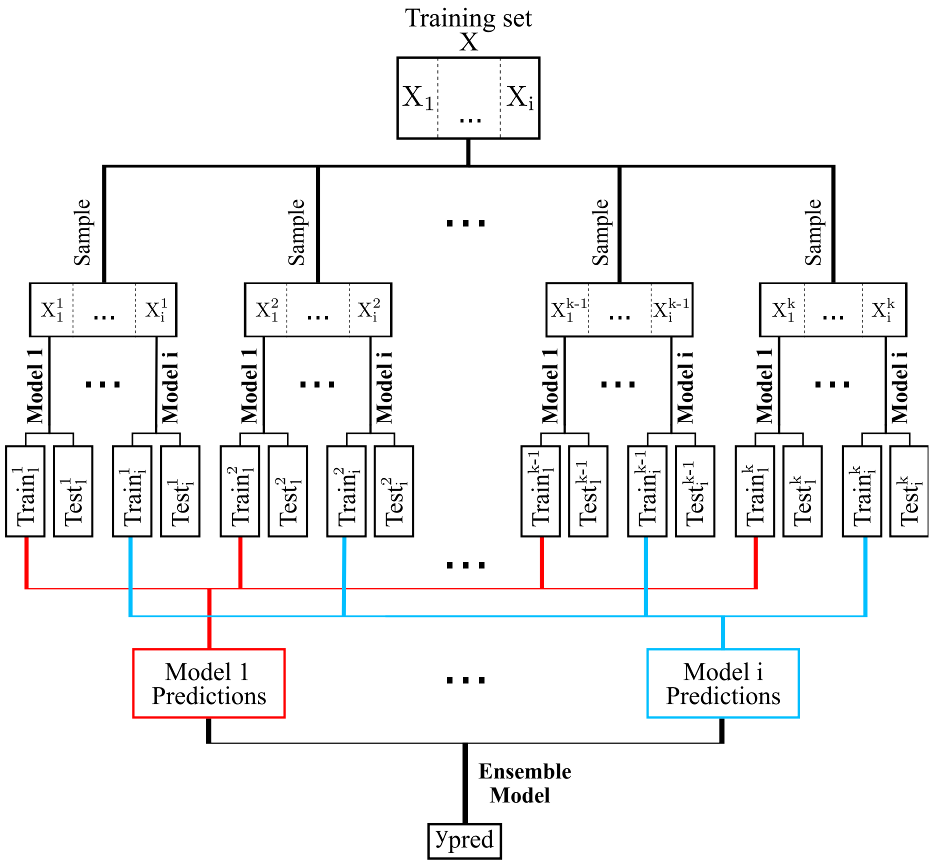

Figure 1: Our general two-layer ensemble approach for a training set X which is the collection of i "feature sets" $X_1, \ldots, X_i$, corresponding with different types of features for each example, and a target y. First the training set is sampled k times (with repetition) to produce the samples $X_j^1, \ldots, X_j^k$ for each feature set X_j. Next, suitable models are trained on the samples to predict y, resulting in k predictions $\text{Train}_j^1, \ldots, \text{Train}_j^k$ of y for each model j. The same models are then also used to generate predictions $\text{Test}_j^1, \ldots, \text{Test}_j^k$ for an unseen test set. Finally, the *ensemble model* is trained on these combined predictions Train_j^l ($1 \leqslant j \leqslant i, 1 \leqslant l \leqslant k$) to give a prediction y_{pred} of y. This final, trained model can then be used with Test_j^l to predict the test set.

- It can do no harm: we can choose the parameters as such that we simply train each model on the whole training set.
- The data was very noisy (see Section 4), with this method we can also control the overfitting somewhat.

We had several well-performing *first layer models*:

- Logistic regression on the combined matrix of *URL*, *title*, *body*, *links* and *htmlbody* from the original feature set (feature set 1) with and without chi-squared feature selection.
- Logistic regression on the combined matrix of all the features from the 1-grams feature set (i.e., feature set 2).
- Logistic regression on the combined matrix of all the features from the 1-grams feature set together with the *body* 2-grams (from feature set 3).

We have tried many other models (random forests, gradient boosting classifiers, SVMs, AdaBoost, etc.) but found these to be quite inferior to logistic regression. First of all, they were all *much* slower. Second, they seemed to fit the noise too much and were harder to regularize than logistic regression.

Since we got such good results from the much faster logistic regression and were already near the end of the competition, we stayed with these models for the first layer.

For the second layer, or the *ensemble model*, we simply used the arithmetic mean since we did not immediately find a better performing model (we have in particular tried ridge regression and non-negative least squares).

4 Results and discussion

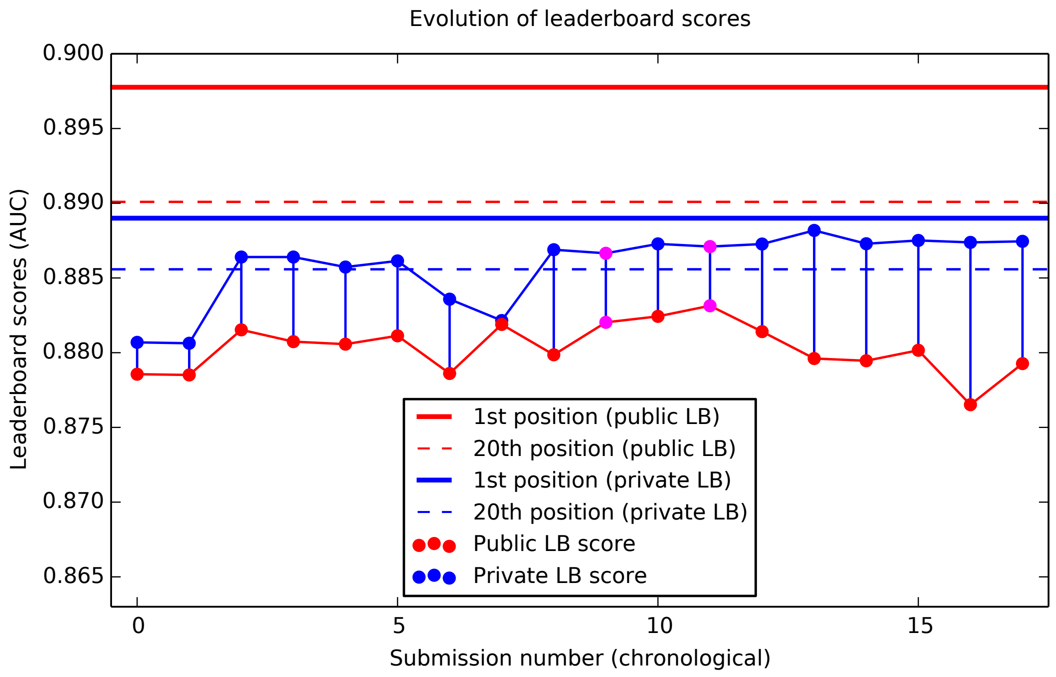

Figure 2: A scatter plot depicting our progress in the competition. Our submissions are plotted in chronological order against their respective public and private leaderboard scores (higher is better). Our two final, selected submissions are coloured in pink. The horizontal lines represent the scores of the first or twentieth position in one of the (final) leaderboards. (Five submissions were left out because they were either simple tests or they were generated with a bug in our submission function resulting in very low scores.)

Figure 2 shows the evolution of our public and private leaderboard scores during the competition.

First of all, it is important to point out that even though many Kaggle competitions ask to solve a problem that is also studied (extensively) in the literature, one should in general be very careful with generalizing successful strategies used in these competitions. In particular, for this competition we dealt with a relatively small training set that also contained many (more than 100 in a training set of almost 7400 samples) apparent "ground truth misclassifications". Many models, including ours, struggled mainly with these apparent misclassifications and even though this was reported to the competition organizers, they said they could provide no more clarification besides describing the crowdsourced gathering of the data until after the competition.[9] But even now (19 December 2013), a few months after the competition ended, no more information has been provided.

Given the noisy data, the very small test set of almost 3200 samples and the small portion of the test set used for the public leaderboard (20% or about 600 samples), it should be evident that the (differences between the) private and public scores will have a high variance. During the competition we experienced significant drops on the public leaderboard even though, when the private scores were revealed, our private scores seemed to have been fairly stable. This is very noticeable in our

[9] https://www.kaggle.com/c/stumbleupon/forums/t/5542/ground-truth-validation

own scoring evolution shown in Figure 2. As such we were also only placed 75th on the public leaderboard when the competition ended. For all these reasons we deem a more extensive study of the performance of these models not very useful.

To conclude the discussion of our results, we take a look at the performance of other people's selected models. One of the things that immediately became clear when the final scores were published was that the leaderboard had completely and significantly changed. Figure 3 shows the difference in the leaderboard positions for the whole leaderboard. For example, the number one competitor on the public leaderboard dropped 300 places (of the almost 650). We have also depicted the number of submissions made during the competition (for completeness) although it does not seem that this variable is correlated that much with the others.

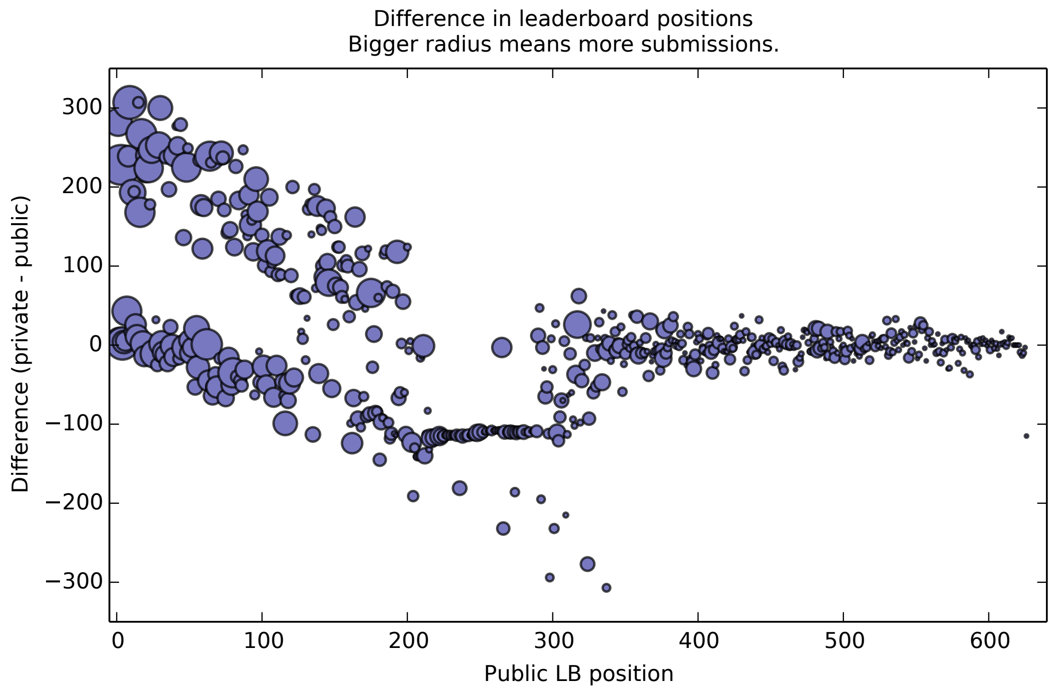

Figure 3: The difference in leaderboard positions at the end of the competition. A bigger radius of a circle represents a higher number of submissions made during the competition.

In Figure 4 we have plotted the same leaderboard differences but now only for the first 100 competitors and here the radius of a circle represents the Kaggle rank of the competitor. Although we think this could be interpreted as an indication that top competitors on Kaggle are less likely to experience significant drops (i.e., overfit), there is significant bias in the data: "good competitors" are automatically more likely to have a higher rank since they have just earned a lot of points with doing well in this competition. Once can, however, more rigorously approximate these influences and most likely agree with the presence of some correlation between rank and the amount of overfitting.[10]

5 Conclusion

It is hard to conclude with some remarks regarding the specific problem since, as noted in Section 4, although Kaggle problems might regularly feature some known problem, conclusions drawn from the results are often not generalizable to other, a priori similar problems. In particular for this competition we had to deal with a small and relatively noisy dataset and have consequently mostly tried to make our model as stable as possible by implementing the general two layer ensemble

[10]We included this small discussion because a lot of the people that experienced these significant drops, claimed it was simply because of the noisy data. Although we certainly agree with the claim that the data is noisy, we do not agree with the claim that these big drops were hardly preventable.

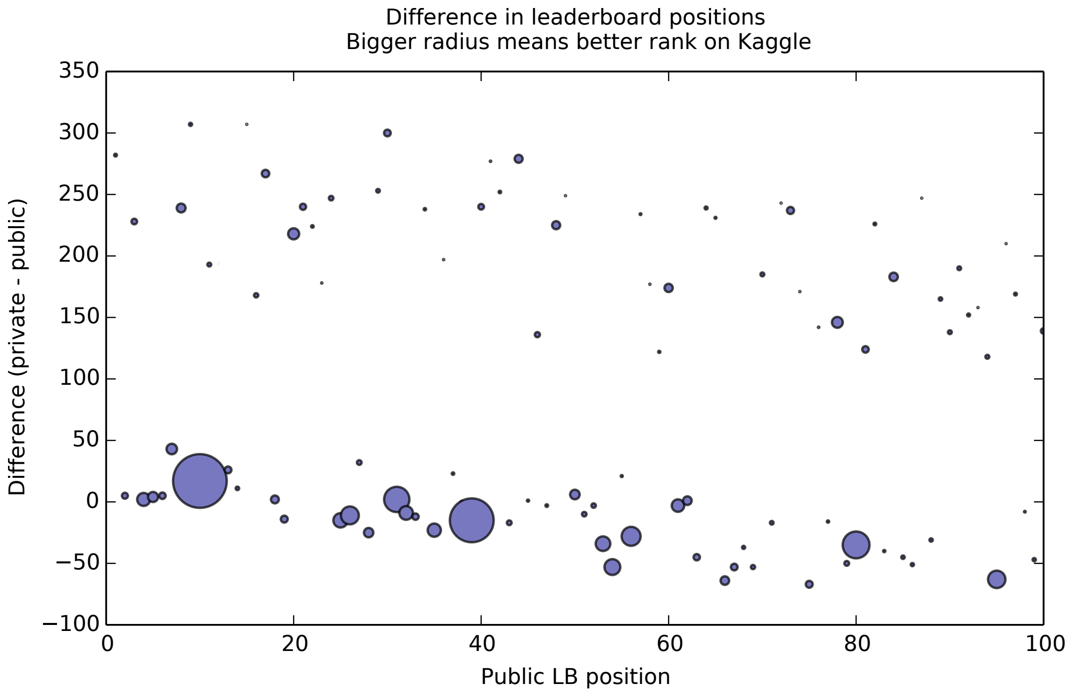

Figure 4: The difference in leaderboard positions for the first 100 competitors on the public leaderboard. A bigger radius of a circle represents a better (lower) rank on the Kaggle website. As noted, there is significant bias in this data.

approach described in Figure 1. Eventually, due to it becoming clearer that the data was very noisy, the problem lost a significant part of its meaning and thus we feel it would misguide the reader when concluding with some of our insights regarding the competition in which we "tried to categorize a web page as evergreen".

Personally, partly since I participated in this competition at the beginning of the course (as opposed to the last three weeks reserved for projects), I only had a limited knowledge of machine learning and Python, and no experience working with scikit-learn. Nevertheless, I have learned a significant deal about many different techniques in machine learning, the many applications they have and how they can be applied. In conclusion, I would say that this project has been an interesting and (mostly) enjoyable experience.

Acknowledgements

I would like to thank prof. Schrauwen for allowing me to enter this highly stimulating competition for my project.

References

[1] Pedregosa, F., et al., *Scikit-learn: Machine Learning in Python*, JMLR 12, pp. 2825-2830, 2011.