Machine Learning-I (CS/DS 864): Project Report

Gayathri Garimella (IMT2013015)
Komaragiri Vasundhara (IMT2013018)
Sneha A. (IMT2013045)
Trisha Mittal (IMT2013053)

December 22, 2016

1 Problem Statement and Motivation

The dataset for our project is from the **StumbleUpon Evergreen Classification Challenge** hosted on the Kaggle website. The objective is to use the dataset given by StumbleUpon containing details of various URLs to build a classifier which can efficiently label the URLs based on the web content as "evergreen" or "ephemeral". It is a binary classification problem.

The website StumbleUpon is a web content discovery engine that recommends relevant web pages and media to its users based on their interests. The ability to determine apriori if a link is evergreen or ephemeral would greatly improve such a recommendation system.

The content present in evergreen URLs drives traffic for a longer period of time. In general, they would require very less or almost no updates. The content present in them is narrowly focused, sustainable and historical. Websites which have content related to how-to-tutorials, scientific articles in various subjects, or cooking, parenting etc. commonly come under the category of evergreen URLs. On the other hand, websites which provide content like news related updates are classified as ephemeral URLs. They drive-in a lot of traffic, may carry valuable information and content which becomes outdated.

2 Dataset

The dataset provided training and testing set of URLs. The training set has 7,395 URLs and testing has 3,171 URLs (total 10,566 URLs). There are 27 field descriptors or attributes provided for all these URLs. We used the following attributes for building the classifier:

Alchemy category: This feature classifies the content present in the URL into categories. We expect categories like food, jobs and careers, parenting, health etc. to come under the evergreen category. So based on this feature, we can predict if the URL will have evergreen content or not.

Alchemy score - A particular URL can come under two or more Alchemy Categories. For example, a website can have details related to Business and Science & Technology. But the major topic under which we classify it can be one among them. So this score gives us a sense of how much

the content is related to a particular Alchemy Category.

Boilerplate - Boilerplate is a text that can be reused in new applications or context without much change from the original. In order to standardize their work, people accommodate boilerplate languages or code into their content or document. Most of the HTML pages also have boilerplate code. In this dataset, HTML boilerplate contains the title, body and URL of the HTML page. Using boilerplate, we can do semantic analysis without actually parsing the entire web page.

Compression ratio - Most of the evergreen URLs talk about very specific topics and don't provide a broad view. Writing about something in a broad manner leads to more redundancy of keywords and makes the content available on the site very long. If keywords repeat more often in a long text, the compressed file will be very small when compared to the uncompressed file. This will imply that its compression ratio will be high. So if the compression ratio is low, then most likely it can be considered evergreen.

News front page - This attribute has value 0 or 1. The value is 1 if the StumbleUpon news classifier determines that this web-page is front-page news.

Spelling errors ratio - This attribute has value between 0 and 1. It is the ratio of words not found in Wikipedia (considered to be a spelling mistake).

3 Building the classifier

We tried various techniques and approaches for building the classifier. The approaches along with their details follow:

3.1 Approach 1

In this approach, we used the raw-dump of the URLs given as one of the features and apply the Latent Dirichlet Allocation (LDA) algorithm to be able to describe documents based on a set of topics and then use Support Vector machine (SVM) for classifying the URLs into evergreen and ephemeral links.

Summarising the LDA Model
LDA represents documents as mixtures of topics that spit out words with certain probabilities. It assumes that documents are produced in the following fashion: when writing each document, you

- Decide on the number of words N the document will have (say, according to a Poisson distribution).
- Choose a topic mixture for the document (according to a Dirichlet distribution over a fixed set of K topics).
- Generate each word w_i in the document by:
  - First picking a topic
  - Using the topic to generate the word itself

Assuming this generative model for a collection of documents, LDA then tries to backtrack from the documents to find a set of topics that are likely to have generated the collection.

Remarks about Gensim

To implement this we used the gensim package in Python. Gensim is a free Python library designed to automatically extract semantic topics from documents, as efficiently (computer-wise) and painlessly (human-wise) as possible. Gensim is designed to process raw, unstructured digital texts ("plain text").

Implementation Details

There were two types of pre-processing performed on the data before proceeding to topic modeling.

1. Removing words which occur only once.
   Removing these words seemed reasonable as given their low occurrence they will not really feature in any of the topics in the LDA algorithm.

2. Removing StopWords
   To remove the stopwords from the documents, we used the ONIX stopword list which contains about 571 words.

After performing the preprocessing, we moved to the actual topic modeling step. Once we obtained the topics occurring in the documents, we input a vector of topic proportions to a Support Vector Machine (SVM) classifier (scikit-package) and train a SVM model. We obtain the best hyperparameters for the SVM classifier (gamma, kernel, penalty parameter (C)) by performing a grid search over possible hyperparameters by performing five-fold cross validation. The prediction occurs in the same manner. First, topics are inferred and using the topic probability as features, we input them to the trained SVM model which then will output a prediction.

Figure 1: A screen-shot of the top 10 different topic distributions generated (along with their probability of occurrence) by the LDA pretty early in the learning process (after mere 10 iterations)

The running time for this algorithm was pretty high. We submitted the output we got on the Kaggle website. The score was **0.42114** for this approach.

3.2 Approach 2

Using various functions from R packages, we built models and predicted the labels for the test data using each model. The various methods that we tried with different combination of attributes are as follows:

Random Forest
Random Forest is an ensemble learning based classification and regression technique. It is one of the most commonly used predictive modelling and machine learning technique. In a normal decision tree, one decision tree is built. However, during the execution of the random forest algorithm, a number of decision trees are built. After the trees are built, a vote from each of the decision trees is considered in deciding the final class of an object.

Advantages of using Random Forest are :

- It can be used when the features are not expected to be linear.
- It is non-parametric, flexible and is a bagging technique.
- It has higher model performance or accuracy

Package Used: randomForest, caret
Attributes Used: alchemy_category, alchemy_category_score, new_front_page, compression_ratio and spelling_errors_ratio

We can tune the machine learning algorithm parameters in R. For randomForest algorithm, two parameters are most likely to have the biggest effect on the final accuracy. They are:

- mtry: Number of variables randomly sampled as candidates at each split.
- ntree: Number of trees to grow.

We have set mtry as 8 and ntree as 500 in our implementation. We took the above mentioned attributes and a plot of the behaviour of the model as obtained from the function is below.
In statistics, a receiver operating characteristic (ROC), or ROC curve, is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. The true-positive rate is also known as sensitivity. The false-positive rate is also known as the fall-out. The ROC curve is sensitivity as a function of fall-out.

We ran randomForest algorithm by using all the attributes except boilerplate and predicted the labels. After submitting the output on Kaggle we got a score of **0.71175**. The scores obtained on running the classifier on some other combination of attributes have been tabulated below. Also, when we printed the model in R we got the following results:

OOB(Out-of-bag) estimate of error rate: 30.01%
Confusion matrix:

	y = 1	**y = 0**	**class.error**
y = 1	2635	1161	0.3058483
y = 0	1058	2541	0.2939705

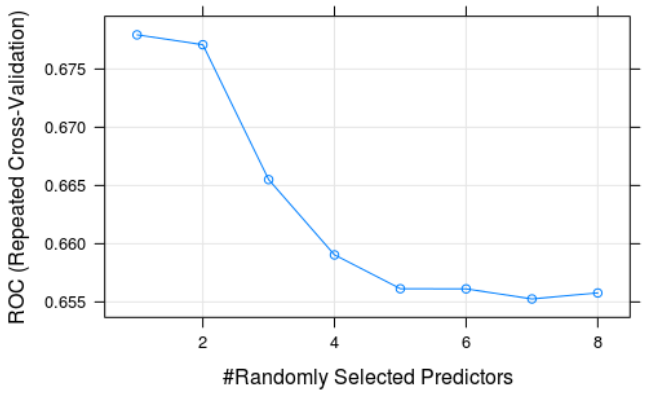

Figure 2: Variation in value of ROC with the number of randomly selected predictors

In the randomForest algorithm, the classifier is built based on two techniques - *Bagging and Random Subspace method*. Given S number of trees in the forest, first create S datasets of same size as original by random resampling of data T with-replacement. This will result in $\{T_1, T_2, ...T_S\}$ datasets. Each of these is called a bootstrap dataset. Due to sampling with-replacement technique, every dataset T_i can have duplicate data records and T_i can have missing data records from original datasets. This is called Bagging. We build a tree K_i for each T_i bootstrap dataset. For classifying a point, we pass the input through each of these trees and we get a set containing labels as output from each tree. The final prediction or label for the given input is a majority vote on this set. For the input record x in original set, select the T_i's which did not pick x. These T_i's will form the set of boostrap datasets. This set is called out-of-bag examples. OOB classifier is the aggregation of votes only over those T_i's that did not contain the particular record. OOB estimate is the error rate of the OOB classifier on the training set.

Logistic Regression (Generalized Linear Models)
Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function.
The logistic function is useful because it can take any real input t ($t \in \mathbb{R}$), whereas the output always takes values between zero and one, interpreted as a probability. The logistic function $\sigma(t)$ ($\sigma(.)$ is the sigmoid function) is defined as follows:

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

We take θ ($\theta \in \mathbb{R}$) as the model parameter, y ($y_i \in \{-1, +1\}$) as the response variables and X ($X^{(i)} \in \mathbb{R}^N$) as the independent variables. Logistic regression models the probability distribution of the class label y given a feature vector X as follows:

$$p(y = 1|X, \theta) = \sigma(\theta^T X) = \frac{1}{1 + e^{-\theta^T X}}$$

Logistic regression can be used as long as the features are roughly linearly separable. The output in terms of probability is the major advantage of logistic regression.

Package Used: stats
Function Name: glm
Attributes Used: alchemy_category, alchemy_category_score, new_front_page, compression_ratio and spelling_errors_ratio
The scores obtained by running the classifier with different combination of attributes as inputs have been tabulated below.

Naive Bayes
Naive Bayes is a simple classification technique for constructing probabilistic classifiers based on applying Bayes' theorem and it assumes strong independence between the features. Naive Bayes model is easy to build and is particularly useful for very large data sets. Naive Bayes is known to outperform even highly sophisticated classification methods. Bayes theorem provides a way of calculating posterior probability $P(y|x)$ from $P(y)$ (prior probability of class), $P(x)$ (prior probability of predictor) and $P(x|y)$ (likelihood) where y is the class variable and x are the attributes.

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

Package: e1071
Function Name: naiveBayes
Attributes Used: alchemy_category, alchemy_category_score, new_front_page, compression_ratio and spelling_errors_ratio
The scores obtained by running the classifier with different combination of attributes as inputs have been tabulated below.

We submitted the output labels for the test data we got on the kaggle site. The following table shows some of the scores obtained:

Function	**Attributes**	**Score**
Random Forest	alchemy_category, alchemy_category_score, new_front_page	0.59241
Random Forest	alchemy_category, alchemy_category_score, new_front_page, compression_ratio, spelling_errors_ratio	0.63113
Logistic Regression	alchemy_category, alchemy_category_score, new_front_page	0.53940
Logistic Regression	alchemy_category, alchemy_category_score, new_front_page, compression_ratio, spelling_errors_ratio	0.55295
Naive Bayes	alchemy_category, alchemy_category_score, new_front_page	0.54472
Naive Bayes	compression_ratio, spelling_errors_ratio	0.53048
Naive Bayes	alchemy_category, alchemy_category_score, new_front_page, compression_ratio, spelling_errors_ratio	0.53900

4 Conclusion and Benchmark Performance Comparisons

While going through the forum posts of the StumbleUpon Classification Challenge on Kaggle, we looked at the sort of attributes and algorithm used by other people. Following that, we started with text analysis on the boilerplate attribute and tried running LDA and SVM for building the classifier. Then, we tried out other approaches like logistic regression, naive bayes and random forest with different combinations of attributes.
From the result table, we can see that random forest algorithm outperformed the other approaches. RandomForest is a non-parametric algorithm so we deciced to experiment with it. Our target was to use randomForest algorithm, select appropriate attributes and reach the benchmark performance. We were able to get a score of 0.71. On Kaggle, there is a benchmark solution which used random forest and got an AUC of 0.75109 in the leaderboard.
Through this project we got hands-on experience on how to apply machine learning algorithms on datasets. Also, we got to experiment with various methods (parametrized and non-parametrized) that helped us to improve our score.

5 References

1. [1]2016. [Online]. Available: http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/ (for LDA description). [Accessed: 22- Dec- 2016].
2. [2]"gensim: topic modelling for humans", Radimrehurek.com, 2016. [Online].
Available: https://radimrehurek.com/gensim/intro.html. [Accessed: 22- Dec- 2016]. (Gensim link)
3. [3]"Beating the benchmark and getting AUC = 0.75109 - StumbleUpon Evergreen Classification Challenge — Kaggle", Kaggle.com, 2016. [Online].
Available: https://www.kaggle.com/c/stumbleupon/forums/t/5450/beating-the-benchmark-and-getting-auc-0-75109. [Accessed: 22- Dec- 2016].