StumbleUpon Project

Classifying whether a site is evergreen or not

By: Amanda Karch

With assistance from: James Walters, Matt Campbell, Colin Cianciolo, and Benji Cantera

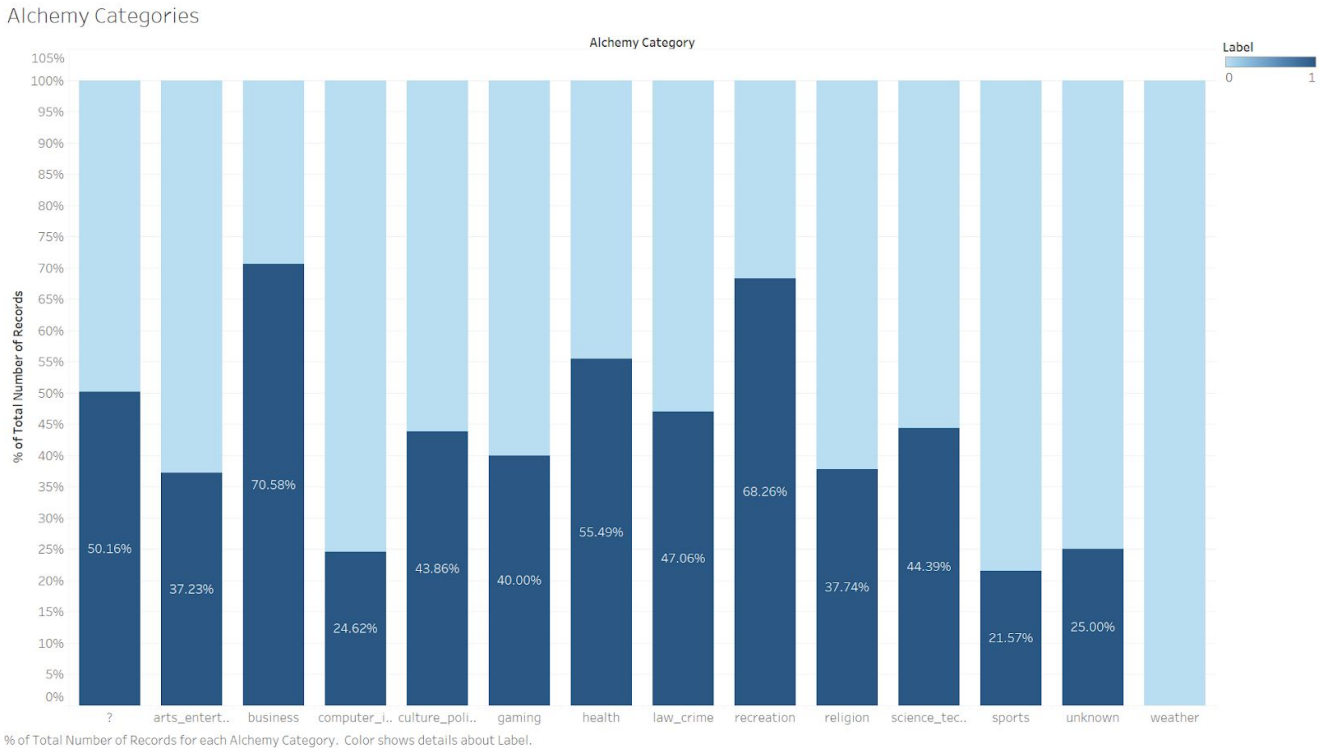
Project Phase 1

QTM2000-04

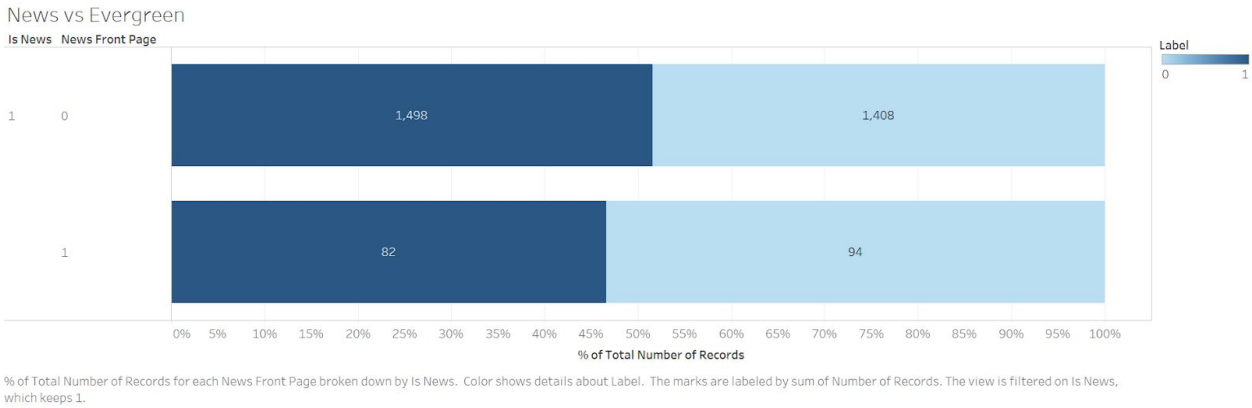
Executive Summary:

The goal of this dataset is to build an algorithm to predict whether or not websites through StumbleUpon are going to be relevant forever or only for a short period of time - evergreen versus ephemeral. Using Tableau, we have discovered multiple relationships among different variables, including which categories are more likely to be evergreen than others, an unexpected lack of connection between news links and evergreen sites, and article sizes based on category. We also have discovered a variety of quality issues that we will need to address in the future, such as missing values, logical variables that are in a different current form, and highly correlated variables.

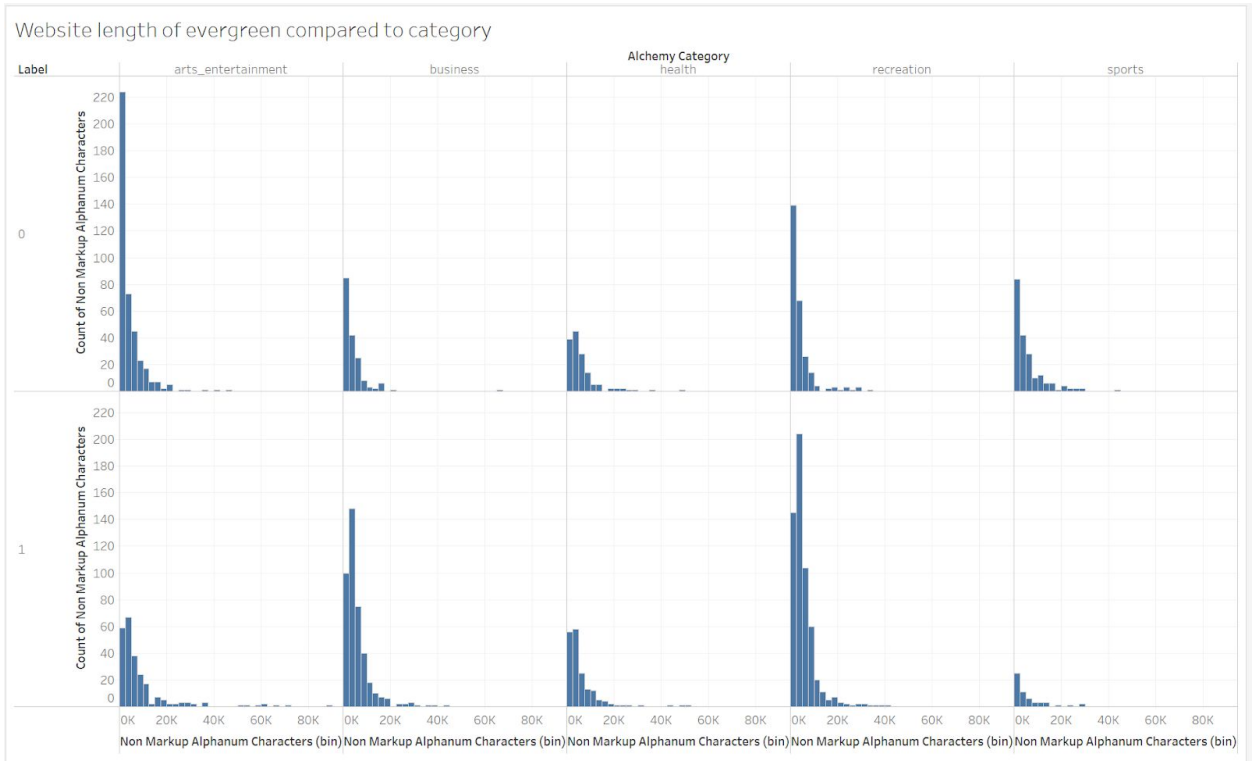
Visualizations:



This visualization shows the different categories of websites and URLs that StumbleUpon was sorting through to determine whether or not they were relevant forever. The colors represent the relevancy lifespan of each website: dark blue as “evergreen” (relevant forever) and light blue as “ephemeral” (relevant for a short period of time). Rather than show each category as a function of its number of records out of the total 5000 observations, percentages are used to compare categories of different sizes. Using this format allows one to see that there are some categories where websites were much more likely to be relevant forever versus others, such as business, recreation, and health.



This visualization shows the proportion of news links that are evergreen versus ephemeral, divided by whether or not they are deemed worthy of "front page news." Clearly, one can see that there are slightly more evergreen links when the news link is not front page news, and the opposite is true of front page news links. This graphic is interesting because, typically, news articles would be seen as ephemeral, as they are usually only relevant for a short period of time, due to constant updates and new stories every day. However, there is a close-to-even split for both front page and non-front page news in terms of length of relevance, which would not be expected.



This visualization shows the different article lengths across alchemy categories and if the site is evergreen or not. What this helps us understand is how to create an evergreen website

based on what alchemy category the content falls into. For example, if someone wanted to create an evergreen website in the recreation category, they would want to try and keep the character count around the four thousand to six thousand mark. This is interesting because it shows us that there are different strategies for ensuring evergreen content based on what the website covers. For example, evergreen sports content is much shorter on average than non-evergreen sports content, helping us predict whether or not the website will be evergreen.

Data Quality Issues:

Upon looking at the data set, there are many variables and values that could cause issues later in the data analysis process. There are many variables that need to be changed to logical variables, as their values are inputted as 0 or 1, for false or true (based on whatever the variable means). Some are stored currently as numerical (*framebased, hasDomainLink, lengthyLinkDomain*, and *label*) and some are stored as characters (*is_news* and *news_front_page*).

There are also a lot of missing values that show up as either question marks or null in both R and Tableau. Some variables also have weird values and total counts. *Is_news* has only 1s (true) or ?s as values - no 0s (false), which is strange because not all of the URLs are news articles. There must be some false/0 values in that data set, but they do not appear when performing an analysis.

There are some highly correlated variables as well. *Numwords_in_url* and *avglinksize* are very closely related, as they represent the number of words in a URL versus the average number of words in a URL. Also, the common link ratios have the possibility to be skewed, as they could double-count links (ie. if the link shares a word with 3 other links, it also shares a word with 1 and 2 link(s)).

Conclusion:

From the relationships we have gathered between different variables, we can start to see some patterns that could help us predict whether a site will be evergreen or ephemeral. Differences between Alchemy categories and non-markup alphanumeric characters (website length) already seem to have patterns that would be able to anticipate which of the two categories a website will be found in. However, due to the number of data quality issues we have also found, there is quite a lot of data management and cleaning that we have to do in order to find out which variables really are useful in predicting whether a site is evergreen.

Project Phase 2

QTM2000-04

Executive Summary:

The overall goal of this project is to predict whether a website is classified as an evergreen or non-evergreen site. In case you are unfamiliar with the term, being classified as evergreen means that your site has sustainable and lasting content that stays relevant past its initial publication. In order to predict whether a site is classified as evergreen or non-evergreen, we have constructed two models. These two models use various inputs from our data set in order to predict the classification of a site. The first model that we constructed is the K-Nearest Neighbors Model or kNN. This model uses numerical inputs from the data set to produce a categorical output, such as whether the site is classified as evergreen or not. The second model we constructed is a Classification Tree. This model uses both numerical and categorical inputs in order to produce a categorical output, such as whether the site is classified as evergreen or not. Ultimately, we ended up choosing kNN as our best model, with evidence supporting it in the body of this work.

Model 1 - kNN:

This paragraph will discuss the kNN model. The process for which kNN works is that the model predicts a categorical outcome, in this case by determining whether a website is classified as evergreen or not, by placing weights on predictor variables based on the distance they are from the target outcome. Predictor variables are able to "vote" on what the final outcome should be. When creating the model, the user sets how "far" the model should go from the target to predict the outcome.

When evaluating the effectiveness of this model, we want to look at the error rate, which is computed by the total number of predictions that were incorrect and dividing it by the total observations, compared to the benchmark error rate, which is computed by finding the mode of the observations. We want to compare these two error rates because it tells us two things: the first being how well the model performed at predicting the outcome compared to choosing the most common outcome, and the second being how often the model makes predictions that are incorrect. For the StumbleUpon data set, the benchmark error rate was 49.6%, while the error rate for the model was 38.55%. We can see that this model outperforms its benchmark by

11.05%. This is a major improvement from the benchmark guess but a 38.55% error rate is still relatively high to be confident in the predictions of the model.

When deciding which model to choose, we need to look at more than just the error rate. One of the many advantages of kNN is that it is very easy to implement and improve. This allows for easy and regular updates to the model at a low cost. One of the drawbacks to kNN is that it is only able to use numerical inputs. This can lead to more inaccuracies in predictions because the model is not utilizing categorical variables which could actually be valuable in predicting whether a site is evergreen or non-evergreen. Understanding how many predictions we predicted evergreen, but observed to be non-evergreen is important in understanding how much incorrect corrections will cost. Another drawback to kNN is that the model could have a target value in which it predicts a "tie" because the predictor values surrounding the target are equally weighted.

Model 2 - Classification Tree:

This paragraph will discuss the classification tree model. The process for which a classification tree model works is that the model predicts a categorical outcome, in this case by determining whether a website is classified as evergreen or not, by breaking down a data set into smaller subsets to produce paths that are distinguishably different to categorize and predict the target outcome. When creating the model, the user sets the minimum "split", "bucket", and "cp" or "complexity parameter". The minsplit is the minimum number of values that must exist in a node before a split in the tree. The minbucket is the minimum number of observations that must exist in a node in order for a split to be attempted. The complexity parameter controls and selects the optimal decision tree size. This optimal size is reflected in how many divisions in the nodes of the tree until the reduction in the relative error is less than a certain value.

When evaluating the effectiveness of this model, like the kNN model, we want to look at the error rate compared to the benchmark error rate, in order to see its effectiveness in making predictions and how many it predicts incorrectly. For the StumbleUpon data set, the benchmark error rate was 49.6%, while the error rate for the model was 39.65%. We can see that this model outperforms its benchmark by 9.4%. This is a major improvement from the benchmark guess but

a 37.95% error rate is still relatively high to be confident in the predictions of the model, like kNN.

Compared to other algorithms, classification trees require less effort for data preparation during pre-processing, which is a big advantage. A classification tree is also very intuitive and easy to explain to technical teams as well as stakeholders. On the other hand, classification trees can get far more complex compared to other algorithms, which is why overfitting and underfitting can be problematic. They also take much more time to train, which can be a negative aspect as well.

Conclusion - Best Model:

At the end of our analysis, we ultimately chose kNN as our best model. Despite it only taking numerical inputs, we found it better suited to the task of predicting whether a site was evergreen or not. The error rate for kNN was lower, even if only by a slight difference, and numerical inputs seemed more important in this dataset, as there were much more of them than categorical inputs. Looking ahead to the future, kNN is a model that is easy to understand and easy to implement changes for improvements, especially when you are able to control for overfitting in a simple manner.

Project Phase 3

QTM2000-04

Executive Summary:

"Evergreen" is a term used to define if the content of a website will be relevant forever and it is what we were tasked with predicting, given a wide dataset with over twenty input columns. In order to find the best model to predict which sites could be labeled as evergreen, we had to take part in data cleaning, which fixed a multitude of data quality issues. This process, for example, fills missing information in the data and removes unnecessary columns that would not be useful in making predictions. After testing a variety of combinations of cleaning tactics, we decided our best range of models in terms of error rate came with basic cleaning (ie. correcting data types and filling in basic columns of missing values), extractions of base URLs, and complex missing values filled in with assumptions (with the exception of Alchemy Category) - with more detail explained in subsequent sections of this work. After data cleaning, we decided to utilize Ensemble Learning methods and combine our previous models (kNN, Classification Tree, and Naive Bayes) into what is known as the bagging model, using random forest techniques. This model correctly predicted whether a website is evergreen or not 71.1% of the time. We found this model superior to the other Ensemble Learning method, the stacked model, by 2.05% and was ultimately our most successful model.

Data Cleaning:

When initially looking at this dataset and its quality, we found a variety of issues. There were multiple columns with missing values or question marks instead of data, outliers that were far out of the normal range of the dataset, and other columns that did not seem useful because of 5000 instances of the same value or having too much text to be useful in predicting whether a website was evergreen or not.

The most basic strategy that we started with was ensuring that all of the variable types were correct: numerical and categorical for respective variables, based on the data dictionary. The next step was replacing the question marks with NAs (not available or applicable) throughout some of the columns: "Is News", and "News Front Page"- this made other cleaning steps easier. "Alchemy Category Score" and "Label" were exempt from this process because it was easier to fill in values for these. These last two columns were filled in with values for the mean (or the mode, for "Label", which is categorical). This answer made sense because the median and the mean of "Alchemy Category Score" were around the same number (about 0.6), which showed that it was a good measure of central tendency to select as an estimate of missing values - as we had no background information on how to calculate the values otherwise. For "Label", there was only one missing value and the split between evergreen and not evergreen was so close that it did not seem like it would make a difference with whatever way we chose to label it - so the mode was an easy choice, as another measure of central tendency.

For "Is News" and "News Front Page", we decided to fill in missing values with "False" (or 0, in a logical sense). These decisions were made because it seemed logical to assume that if

a page could not originally be classified as news, it most likely was not news, and almost 80% of the data was not classified as front-page news. Nevertheless, these were assumptions made, so they could be a weakness and a reason why our error rate could not be lowered, but it was better than basing predictions off of incomplete data.

We ultimately deleted two columns from the dataset, "URL ID" and "framebased", because they were not useful in predicting whether a site was evergreen or not. The URL ID column was just a classifier and identification tool and all of the values for "framebased" were the same, so it told us nothing.

We thought it would be helpful to extract the base URL from the URL column, to try to see if that was useful in predicting if a site was evergreen or not. Base URLs would be something similar to "www.google.com", instead of all of the tags added on with backslashes at the end. We started off with 5000 different URLs and reduced that to 2498 different base URLs. Because there were repeated values, there was a better chance of the column being useful in predicting our target. This decision ultimately ended up decreasing our error rate for all of our models, except for kNN - which makes sense, because kNN does not rely on categorical predictors.

There were a few other cleaning processes that we tried, but they did not end up giving us the best error rates, so they were left out of the final code. They included filling in the "Alchemy Category" missing values, normalizing the range of variables like "Average Link Size", "Compression Ratio", and "Image Ratio", and removing the "Boilerplate" column. First, when looking at the different factor levels of "Alchemy Category", about a third of them were question marks. There was a level labeled "unknown", but it did not have many values. We tried filling in missing values with "unknown" because it was true that the category was unknown to us, but the error rate was slightly higher than our best error rates with that process utilized. Looking at the next attempted process, "Average Link Size", "Compression Ratio" and "Image Ratio" all had outliers as numeric variables - there were some extremely large maximums and one column even had a negative value, which was not possible given the definition in the data dictionary. Normalizing the columns by range would scale the data so it was between the values 0 and 1. We thought this would make a big difference for using models like kNN, where the distance between points is the method of predicting a target, but it did not end up changing its error rate. Lastly, we thought that removing the "Boilerplate" column would be useful because every single value of it was different across the rows, with nothing similar across values, so we did not think it would be able to predict if a site was evergreen. While removing it did produce our best stacked model error rate, it did not prove to be the best decision overall, so it was left out.

The table below shows the error rates after different combinations of cleaning methods, with combinations labeled as:

1. Basic cleaning - changing variable types, filling in NAs/?s for Label & Alchemy Category Score
2. Basic cleaning + cleaned URLs
3. Basic cleaning + cleaned URLs + filled in NAs

4. Basic cleaning + cleaned URLs + filled in NAs (not Alchemy Category)
5. Basic cleaning + cleaned URLs + filled in NAs + normalized columns
6. Basic cleaning + cleaned URLs + filled in NAs (not Alchemy Category) + normalized columns
7. Basic cleaning + cleaned URLs + filled in NAs (not Alchemy Category) + no Boilerplate
8. Basic cleaning + cleaned URLs + filled in NAs + normalized columns + no Boilerplate

Cleaning → / Error rate ↓	1	2	3	4	5	6	7	8
kNN	0.405	0.405	0.405	0.405	0.405	0.405	0.405	0.405
Naive Bayes	0.3825	0.3315	0.34	0.3395	0.3395	0.34	0.3395	0.3395
Tree/Pruned	0.404	0.31	0.31	0.3095	0.3095	0.31	0.321	0.321
Random Forest	0.2915	0.2895	0.291	0.289	0.296	0.293	0.289	0.296
Stacked Model	0.404	0.31	0.31	0.3095	0.3095	0.31	0.2895	0.295

With green meaning lower error rates and red meaning higher error rates, it is very clear which cleaning method was the least effective (#1) but it is a little more difficult to see which one is the best. Cleaning the data well definitely increased model performance but sometimes cleaning too much (and filling in data based on assumptions that could ultimately be incorrect) did not help. If you wanted to choose the cleaning process based on having low error rates that were all relatively close together, you would choose method #4, but if you had a certain model in mind that you wanted to use, or you wanted to base your predictions on certain inputs over others, there is a possibility that you could change your answer.

Model Building:

The final model that we chose for predicting whether a site was evergreen or not was the Naive Bayes' model. The basic idea behind Naive Bayes' is that it calculates the probability of an event happening given other events occurring first. In the context of this problem, the model will predict the probability that a site is evergreen or not evergreen given certain values of URLs, Image Ratio, HTML Ratio, etc. From those probabilities, a site is classified as 1 (True) if it is evergreen or 0 (False) if it is not. This method only works if you assume that the inputs are conditionally independent, given a fixed value of the target - that is, the probability of two inputs given an output occurring is equal to the product of the probability of each "input, given output" pair.

When evaluating the performance of this model, the first step is to look at the error rate. We will use the error rate for the cleaning method #4 (see **Data Cleaning** above) - this value was 0.3395. This means that 66.05% of the time, the model predicted correctly if a site was evergreen or not. This performance was better than the kNN model, but not as good as the classification tree, random forest, or stacked models. Another evaluation process is to compare the model's

error rate to the benchmark error rate - that is, how much improvement the model gives over simply choosing the most common target value for the prediction. The benchmark error rate for this partition was 0.4935, so it is evident that we dramatically reduced the number of incorrect predictions by using this model.

An advantage of Naive Bayes' is its flexibility. Some models that accept categorical inputs only expect logical (True/False) variables to be used as predictors, but Naive Bayes' accepts all categorical inputs because it is strong enough to accurately predict a target using different factors. However, this also ties into a major disadvantage, which is that Naive Bayes' only accepts categorical inputs. This may not be a problem in some situations, but when the majority of the dataset in this context is numerical, it does not factor in all of these variables to make its prediction - which explains why other models like classification trees (that take in both numerical and categorical inputs) have lower error rates. The performance of Naive Bayes' is also much harder to visualize, as there is no simple way to plot results or measures of accuracy, but it is useful that it allows you to view probabilities as well as the actual logical predicted values for the target. It can help you understand what factors could be more useful in predicting a target than others, as long as you understand the concept of conditional probabilities and conditional independence.

Ensemble Learning:

In order to maximize our effectiveness and accuracy in our predictions, we decided to utilize Ensemble Learning. This allows us to take our base models of kNN, Classification Tree, and Naive Bayes', which alone can struggle to perform well, and combine them using some of the methods within Ensemble Learning. One of the Ensemble Learning methods we are using is the bagging model. One way of doing this is by utilizing random forests. A random forest is a collection of tree models that in turn produce a prediction. The prediction of each tree is like a vote. The aggregate of all the votes put together is the prediction of the model. In general, we see that random forests create more robust models as they combine concepts from multiple ensemble methods. When comparing our original models with our random forest, we looked at the error rate to see how the ensemble method improved in performance. Here are the error rates for our models - kNN, Classification Tree, Naive Bayes', and Bagged - respectively: 40.5%, 30.95%, 33.95%, and 28.9%. This means that 71.1% of the time the random forest will predict whether a website is evergreen or not, correctly.

The second method of Ensemble Learning that we decided to utilize is a stacked model. A stacked model differs from a random forest because, instead of aggregating the votes of the individual trees, it creates a meta-model (or new, singular model) that outputs predictions based on the predictions of our three original models. This new singular model then learns when to trust each individual "helper" model or, in this case, our original models. The stacked model's error rate is 30.95%, meaning that 69.05% of the time the model will correctly predict whether a website is classified as evergreen or not. If we compare this to our original models, we can see

that it still outperforms Naive Bayes' and kNN on the basis of error rate; however, it performs equally to the Classification Tree.

When evaluating which Ensemble method performs better on the basis of error rate, we can see that the random forest outperforms the stacked model. While only a 2.05% difference in the error rate, the difference is significant enough to clearly denote a superior model. One potential reason in which the random forest could have performed better is due to the type of our original models. In bagging, base models with low bias and higher variance perform well because we are aggregating the vote - that is, weird votes do not influence the opinion of the prediction as much because we are taking an average of all of them. In stacked models, we can see that the predictions of the meta-model are based on the outputs of the helper models. This means that it can be limited by the accuracy of its helper models because it is relying on the accuracy of those helper models to make a correct prediction. However, the added value of a meta-model is that it is able to learn over time which model to trust and when; therefore, it is able to train itself as it is introduced to more data.

Conclusion:

Despite many data quality issues, we were able to overcome the challenges to create a variety of different models, based on different combinations of variables, that were tested with predicting whether a website was evergreen or not. It was hard to choose which combination of cleaning methods worked best, as error rates across models varied depending on the solutions we chose, but ultimately the decision lies in the person. Using a range of different models that each have their own strengths and weaknesses allows us the possibility to choose, which is a powerful tool to have when making predictions on wide datasets. Overall, there are many factors that have a role in predicting whether a website is going to be relevant forever or not - and there is still much to learn from even the individual dataset itself, given more time.