Predicting the popularity of user-generated discussion questions

Sarah Lim, Jennie Werner, Sameer Srivastava, Aiqi Liu
Northwestern University
{slim, jw18, sameersrivastava2017, aiqiliu2018}@u.northwestern.edu

ABSTRACT

Social web communities often rely on user-generated questions and discussions to provide information and promote camaraderie. However, there is little understanding of what makes a question engaging in this context. We analyze questions submitted to a popular discussion website, AskReddit, using time-independent features to train a number of decision tree classifiers. Alternating decision trees achieve 72.9819 accuracy with 10-fold cross-validation, an improvement over the ZeroR baseline of 51.0708. Features related to the language of the question, time and day of posting, and initial commenting behavior prove most informative.

INTRODUCTION

Question-based discussion is a crucial user engagement mechanism across a variety of web services. Examples of such services include discussion communities organized around particular areas of interest, user research platforms, and Q&A websites. Although question-based discussion has wide-ranging utility, there is little understanding of what makes particular questions more engaging or salient in a social web context.

In order to better understand what makes a question popular, we focus on a particular online community called AskReddit (http://reddit.com/r/AskReddit/). AskReddit is an online forum with over 11.5 million subscribers, where users submit open-ended questions for community discussion. Questions may relate to any topic, must be "clear and direct," and include no more than "two, short, necessary context sentences." AskReddit users interact with submissions by commenting, "upvoting," or "downvoting." An internal ranking algorithm computes a score for each submission based on factors including upvotes, downvotes, comments, and submission age; the homepage displays submissions ordered according to score.

Our main goals are (1) to predict whether a recently-submitted AskReddit question is currently popular, and (2) to understand which features and properties influence the popularity of a submission.

TASK

Problem statement

We define our problem as a binary classification task. Given a time t, and an AskReddit question $\boldsymbol{x}$ submitted within the 24 hours preceding t, we would like to predict whether $\boldsymbol{x}$ is currently popular. A question is *popular* if it ranks within the top 25 AskReddit posts at time t, according to the site's default ("Hot") sorting mechanism. These top 25 posts are colloquially known as the "front page," and community members generally consider reaching the front page to denote a meaningful threshold for a post's popularity.

Dataset

Our complete dataset consists of 674 examples, each representing a unique AskReddit submission with features captured at query time. We use the Reddit API to retrieve the top 25 questions at time t of query (our positive examples), along with 25 other randomly-selected questions posted within the past 24 hours (our negative examples) of t. Scrapes repeat as necessary until the dataset contains an equal number of positive and negative examples for the current day of the week (Sunday, Monday, Tuesday, etc.). Table 1 describes our chosen metadata features.

Language features

To extract features based on question wording, we randomly partition our dataset into 90% and 10%, using the 10% subset to train natural language models. The resulting models perform comparably to those trained on 20% and 30% partitions of the dataset.

Let L denote the subset reserved for language modeling, with bipartition L_{pos} and L_{rand}. We train dictionaries for each of the following features using L_{pos}, and an equally-sized set L_{rand} of examples randomly chosen from L, so as to distribute evenly across positive and negative examples.

The `token_score` feature represents each question as a "bag of words." More formally, for some instance $\boldsymbol{l}$ in $\boldsymbol{L}$, we tokenize its title, and remove common articles and punctuation. Let $\boldsymbol{t}$ denote the resulting vector of tokens. We train dictionaries $D_{token,\,pos}$ and $D_{token,\,rand}$ by iterating through L_{pos} and L_{rand} respectively, and storing the cumulative frequency of each token from all titles in the class. Given a new instance $\boldsymbol{x}$, we first obtain its corresponding vector of k title tokens, $\boldsymbol{t_x}$. Then its `token_score` for class c is given by

$$\frac{\sum_i^k P(t_i \mid c)}{k}$$

Feature	Type	Description
post_localTime	Numerical	Post time, author's local time zone
post_utcTime	Numerical	Post time, UTC
day_of _week	Nominal	Day of the week posted
question_type	Nominal	First question word in title
title_length	Numerical	Characters in title, excluding tags
nsfw	Boolean	Not Safe For Work flair added
serious	Boolean	Serious flair added
author_link _karma	Numerical	Net upvotes on author's prior link submissions
author_comment _karma	Numerical	Net upvotes on author's prior comments
author_account _age	Numerical	Time between author account creation and posting
author_gold	Boolean	Author has Reddit Gold status
time_ to_ first_comment	Numerical	Time between posting and first comment
10min_comment	Numerical	Number of comments, ten minutes after posting
hot	Boolean	Categorical variable

Table 1. Summary of metadata-based features.

where t_i denotes the ith token in $\boldsymbol{t_x}$, and $P(t\_i \mid c)$ is computed by checking the dictionary D_token, c.

Since `token_score` fails to account for the different meanings of individual tokens, we also compute a `sense_score` feature. Training for $D_{sense,\, pos}$ and $D_{sense,\, rand}$ follows the same process outlined for the token-based dictionaries, with one additional step: for each title token, we first compute its synset using the `pyWSD` implementation of the Lesk Word Sense Disambiguation algorithm. Each word in the resulting synset is then added to the dictionary, resulting in a dictionary that more accurately captures the meaning of question titles. Given a new instance $\boldsymbol{x}$ with title token vector $\boldsymbol{t_x}$, its `sense_score` for class c is then

$$\sum_{i}^{k} s(t_i,\, D_{sense,\, c})$$

where s denotes the path similarity between the synset of t_i and the sense dictionary for the corresponding class.

METHODS

We use Weka to train and evaluate a number of models using the remaining 90% of our original dataset. All models use 10-fold cross-validation. Our ZeroR baseline achieves 51.0708 accuracy, as we take care to balance the frequency of positive and negative instances throughout our dataset. The following section describes our selected classifiers.

One-level decision trees

We use one-level decision tree algorithms to understand which features provide the most information gain with respect to question performance.

- **OneR** produces a multi-way split on a single attribute.
- **DecisionStump**, conversely, splits on a single attribute/ value combination.

Full decision trees

Decision trees are especially well-suited for our task, because they are supervised concept-learners, and we would like to understand more about the relative information gains provided by various attribute/value splits. With the possible exception of hybrid Naive Bayes/Decision Trees, decision tree models also encode conditional dependencies between features very well.

- **J48Tree**, Weka's implementation of Quinlan's C4.5 algorithm, chooses splits that maximize information gain measured in terms of Shannon entropy.
- **NBTree** is a Naive Bayes and Decision Tree hybrid learning model. We would like to understand how this

affects attribute selection, since a number of our features (e.g. all the temporal ones) seem intuitively dependent.

- **REPTree** is similar to other decision tree learners, except that in the case of numeric values, the algorithm tries to minimize the total variance.
- **ADTree** implements the alternating decision tree algorithm, which incorporates boosting into decision tree generation by treating each predicate node as a "weak hypothesis" and generating a real-valued prediction for that node. At test time, an instance traverses the tree based on the conditions it matches, and the cumulative sum over the corresponding predictions designates its final classification.

Other

- **NaiveBayesSimple**, a Naive Bayes classifier, applies Bayes' theorem and makes strong conditional independence assumptions between features. Again, we would like to understand to what extent our feature space is linearly independent.
- **Logistic** implements a multinomial logistic regression model, which we will use to interpret the extent to which our numerical features and output classification are monotonically-related.

RESULTS

Alternating decision trees with 7 boosting iterations (as opposed to 10, the default setting) achieve the best performance of attempted classifiers. 10-fold cross-validation accuracy is 72.9819, an improvement of 21.911 over the ZeroR baseline. Table 2 gives the confusion matrix for our ADTree classifier with `numBoostingIterations` set to 7.

Table 3 summarizes the performance of all attempted classifiers, evaluated using 10-fold cross-validation.

DISCUSSION

Prediction accuracy

Given the limited size of our dataset and somewhat informal sampling methods, achieving a 21.911 improvement over

Negative	Positive	
204	106	**Negative = 0**
64	233	**Positive = 1**

Table 2. Confusion matrix for ADTree with `numBoostingIterations` set to 7.

Classifier	% Accuracy
ZeroR	51.0708
OneR	56.8369
DecisionStump	63.0972
J48Tree	69.028
NBTree	70.346
REPTree	70.6755
NaiveBayesSimple	62.603
Logistic	70.0165
ADTree, n = 10 (default)	71.9934
ADTree, n = 7	72.9819

Table 3. Summary of classifiers attempted, and correctly classified instances using 10-fold cross-validation. For the AD Tree classifier, `n` denotes the value of the `numBoostingIterations` parameter.

ZeroR is a promising result. The performance gains provided by alternating decision trees appear consistent with prior theoretical and empirical work on the effective nature of boosting procedures. Decreasing the number of boosting iterations by 30% (from 10 to 7) yields a 1% performance increase, which indicates that too many iterations causes overfitting to the training data.

All multi-ply decision tree methods achieve roughly comparable accuracy, and furthermore perform strictly better than Naive Bayes classification. This confirms our intuition that conditional dependencies exist within our chosen features (for instance, the time and day of posting might impact how many comments a question receives within the first ten minutes).

OneR and Decision Stump perform significantly worse than other approaches. This is likely due to the fact that single-layer decision trees are inherently less expressive than their multi-layer counterparts.

Attribute importance

Across all our decision tree algorithms, the following attributes prove consistently informative:

- *Title language.* Token-based language features provide significant information gain. Decision stumps choose `hot_token_score` as the single split, and both alternating decision trees and REP Tree split on `hot_token_score` first. C4.5 first splits on `random_token_score`. Figure 1 illustrates the

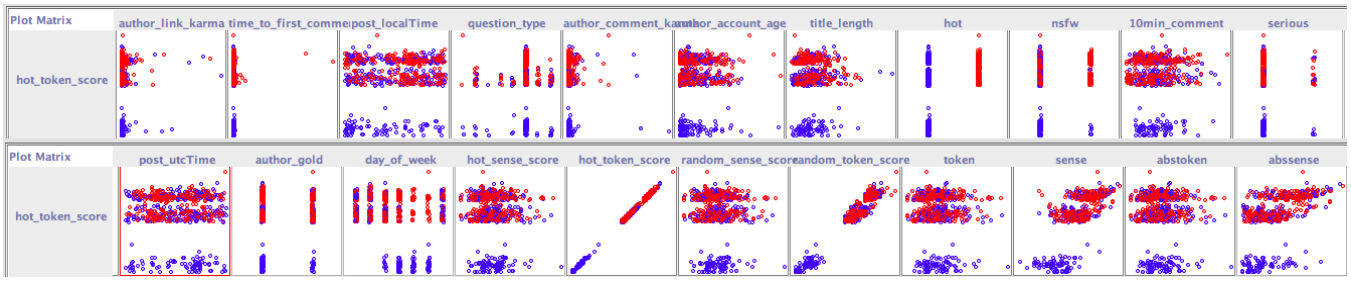

Figure 1. Splitting on `hot_token_score` significantly increases the purity of resulting data by identifying a subset of negative examples, but has difficulty distinguishing between the remaining examples. Red examples are popular/positive, and blue examples are not.

information gain yielded by splitting on `hot_token_score`.

- *Day of posting.* For the three full decision tree algorithms, `day_of_week` is among the first three splits. In general, questions posted during the weekend were much more likely to become popular. Different classifiers chose slightly different splitting thresholds for discrete values (e.g. C4.5 chooses {*Friday*, *Saturday*} as the "popular" subset of days, while REP Tree chooses {*Thursday*, *Friday*, *Saturday*} .
- *Time of posting.* `post_localTime` is the single attribute chosen by OneR, and one subtree of C4.5 only splits three times, with `post_utcTime` being one of these splits. Alternating decision trees likewise split on both features.
- *Early commenting behavior.* Questions receiving more engagement within minutes of posting are significantly more likely to achieve longer-term popularity. C4.5, alternating decision trees, and Naive Bayes Tree both split on `time_to_first_comment` within the first three levels, and likewise for Naive Bayes Tree and REP Tree with `10min_comments`.

Figure 2 summarizes the first four splits of alternating decision trees with 7 boosting iterations.

Limitations and future work

Our dataset is relatively small. Our chosen threshold for positive versus negative classification means each query can

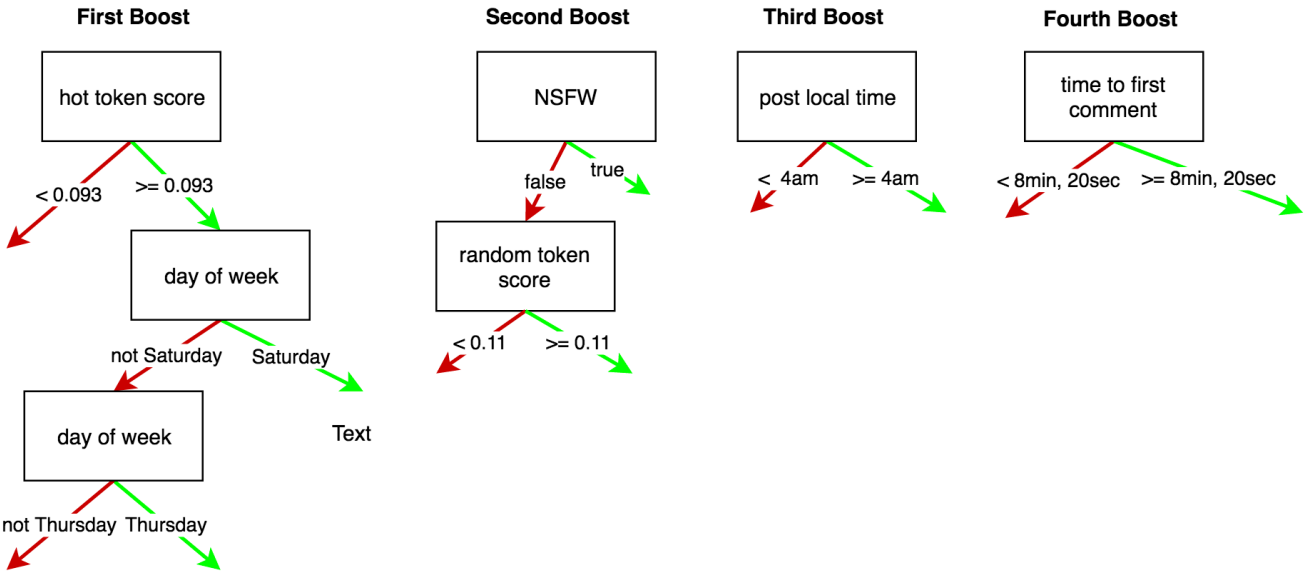

Figure 2. First four boosting iterations of our alternating decision tree classifier. `hot_token_score` and `day_of_week` are consistently informative features.

only contain 25 positive examples, and in order to preserve categorical parity within our dataset, we limit each query to the same number of negative examples. More data would not only improve the external validity of our machine learning experiments, but also create more representative dictionaries for our natural language features.

More broadly, our chosen problem framework inherently limits the predictive power of our model. We define our task using binary classification for the sake of feasibility, but the popularity of user-generated content (and a number of potentially valuable features, such as comment and upvote *rates*) can be much better understood as a function of time. As such, a regression task involving time-slicing could potentially provide a significantly more robust understanding of question popularity and engagement.

ABOUT

For the data collection and pre-processing stage, Sarah wrote the script to query the Reddit API, retrieve metadata features, and output the results to CSV. Sameer and Aiqi implemented natural language processing for our language-based features, and trained the necessary dictionaries. Jennie implemented a random scraping procedure used to obtain non-popular examples.

For the machine learning analysis stage, Aiqi and Jennie scraped data at regular intervals. Jennie ran the data through multiple Weka experiments and recorded the results. Sarah synthesized the results and wrote the report, and Sameer and Aiqi built the webpage and wrote the extended abstract.

All group members participated in all method-related decisions (e.g. feature selection, model selection, training strategy and dataset division, etc.).