Predicting Altruism Through Free Pizza

Course: Artificial Intelligence

Faculty of Science, Department of Mathematics

University of Zagreb

Authors:

Petra Martinjak

Dinko Ždravac

23.01.2017.

Table of Contents

1. Introduction

On a globally popular news aggregation site/forum Reddit (reddit.com) for years there's been a community which, like any other community on the site, has its special set of rules that apply to it. However, this community's rules are a bit unorthodox: a user may open a thread only to request a pizza from their fellow *redditors* (users of the site) or participate in an existing thread opened with the same goal by another *redditor*. Community name is 'Random Acts of Pizza' (RAOP).

"I'll write a poem, sing a song, do a dance, play an instrument, whatever! I just want a pizza," says one hopeful poster. The community has been a huge success with users giving each other free lunches. However, not everyone get a slice and more than half of all the threads are not successful, meaning that nobody decided to buy the original poster (who opened the thread) a pizza.

The question naturally arises: what influences the request's success? Can it be measured? If so, can we successfully predict, based on the data available to us, outcome of a post before posting it? Kaggle (kaggle.com), website for learning and working with Data Science, opened a competition named 'Predicting Altruism Through Free Pizza' about this very topic: urging programmers to come up with algorithms and whole programs which could determine the outcome of a pizza request in RAOP. The site provides a collection of raw data (called dataset) collected from more than 5 000 threads on RAOP opened between 2010 and 2013, aimed at giving us a tool to complete the task. The data is supposed to be crunched by algorithms of our choice and give back the predictions about outcome of a request. Also, on the site, we're given a case study about the topic named "How to Ask for a Favor: A Case Study on the Success of Altruistic Requests" (Althoff et al.) which delves into topic about the influence of a certain aspect of RAOP request on the outcome of that request, from the linguistic and psychological side. Some conclusions from that paper are used in our project.

My colleague Petra and I have decided to solve that problem and present it as a project for the 'Artificial Intelligence' course at the Faculty of Science, Department of Mathematics (University of Zagreb). This paper is accompanied by the program itself and its program documentation and is meant to provide basic understanding of the problem presented as well as the approaches we took to solve it. In this paper we'll discuss specifics of the problem, possible approaches and obstacles that present themselves on the path to predicting successfully RAOP request outcome.

2. Problem

We are given a dataset of 5671 textual requests for a pizza from Reddit community RAOP along with metadata extracted from respective forum posts such as post time, number of upvotes/downvotes by other users, number of comments etc. and the outcome of a request. We are to take all relevant data into account and try to predict the outcome of a given request by constructing a machine learning program which will do that based on given data.

Since we handle raw data, we have no *a priori* way of knowing which parameters influence the outcome of a request and to which extent. Intuition might lead us to believe that the text body of a request plays a bigger role in request outcome than, for example, number of comments the person requesting pizza ("requestor") had since registration on the site Reddit, i.e. that people deciding whether to buy a pizza to the requestor will base their decisions on the request text more than on the profile of the requestor. Intuition and above mentioned paper on the topic will influence our construction of data classifier we're going to use.

We may take such observations into account when choosing relevant parameters considering not all of them are equally relevant. The problem then consists of deciding which data attributes may be more relevant than others and which data algorithm should be used for learning.

2.1. Approach

Since we have a large dataset of requests with multiple attributes and the defined outcomes (correct labels) for each request, an approach of supervised learning will be used for classification of inputs into correct classes. This approach belongs to a category of classification algorithms. In machine learning, it is said that learning is supervised if the training is based on data which provides correct class labels (in this case outcome of a request) for each input (in this case request and its attributes). We are to observe a large number of examples, make a connection between their attributes and use it to build a model classifier by which we may be able to predict an outcome of future test cases.

There are several viable machine learning algorithms for this kind of task. We chose Naïve Bayes Classifier which offers optimal performance and is well suited for text classification. It assumes that all observed attributes are independent variables and calculate MAP (Maximum A Posteriori) probability of a hypothesis using Bayes' theorem from Probability. That assumption, although 'naïve', still makes this algorithm sufficiently precise for predicting outcomes of unseen requests.

3. Solution

3.1. Implementation

Python 3 was used to program the classifier model. Data about all the requests was stored in a *.json* (JavaScript Object Notation) file provided by Kaggle and imported into the program. Also, keywords from certain topics that were used from the research paper were stored in *.txt* files (more on that later). As for Naïve Bayes classifier, we used NLTK API from www.nltk.org, version 3.0 and Scikit-learn multinomial Naïve Bayes implementation from www.scikit-learn.org/.

3.2. Building the classifier

Firstly, data from all requests is loaded and stored into a list. For each request, correct class label is associated in Boolean form, based on request outcome. This allows for searching, evaluating and elsewise manipulating requests list based on their outcome. For each request, we have on disposal values of every attribute that was collected (list of all the attributes and their meanings may be found in the competition introductory *readme* file), e.g. for each of the 5671 requests we may access *"request_title"* or any of the other 32 attributes.

Then we used a *feature extractor* function which converts each input value to a set of features following a certain rule, based on their attributes; this is then fed to a an algorithm that will observe values of attributes across all data, taking note which attribute values correspond to (un)successful requests and shaping the model accordingly (Figure 1).

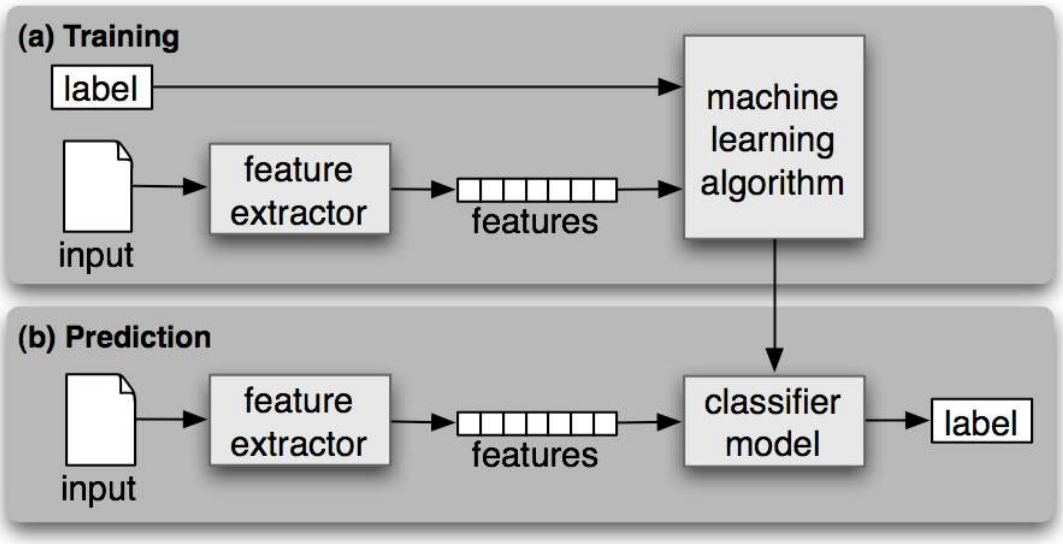

Figure 1 (from: NLTK book)

This is a two phase process: first we will take a portion of data to learn from, called 'Training Set' and apply learning algorithm to it. After we have classifier model trained, we apply trained model on a so-called 'Test Set' and measure results. Success rate (accuracy) of the model is calculated by the following formula:

$$accuracy = \frac{no.of\ correct\ clasifications}{size\ of\ test\ input}$$

However, precaution must be made: how we choose to divide complete data into these two sets may influence learning capabilities of a classifier. We had to address that issue as well. Fundamental assumption of machine learning is that distribution of inputs in our training set is identical to the distribution of those in our test set, as well as other, currently unknown, possible inputs. That assumption may be skewed a little in real life but too much of a difference may render our classifier useless outside of test examples. In other words, our training set needs to be representative of the general data for learning to be successfully applicable to outside examples and our test set needs to be representative for us to correctly measure its effectiveness.

For our sets to be representative, we included equal ratios of positive vs. negative requests in both our training and test sets. Both of those lists were shuffled before putting them into sets. Then the sets were shuffled. Ratio of 0.7 was used to split data into two sets: 70% of data was used for training and the rest for testing.

We had total of 4 classifiers (Figure 2). First three used NLTK Bayes implementation and fourth used the one from *Scikit-learn*. First classifier used all of the provided attributes as features for feature extractor function, second one used only certain, selected attributes.

	Implementation Library	Used attributes from metadata	Used text analysis
NaiveBayesAllAttributes	NLTK	All	No
NaiveBayesSomeAttributes	NLTK	Selected	No
NaiveBayesTextAnalysis	NLTK	No	Yes
Scikit Naïve Bayes	Scikit	No	Yes

Figure 2

3.3. Scikit-learn library and Bag of words model

Another approach we had was to use scikit-learn library along with the Bag of words model. Scikit-learn library (in further text sklearn) is free software machine learning library for Python. When it comes to text classification, one of most used methods for solving problems is Bag of words model (Figure 3). Let's say we're given a dataset that consists of number of documents. We also need a word corpus, or in other words a set of all words in entire document. First we mark every document with integer, and do the same for every word in corpus. Then for every #i document we count occurrences of the #j word in that request, and put result as value X[i][j], where X is a n x m matrix, n being number of documents and m number of words in corpus. Because of huge amount of data, coming from great number of documents and even greater number of words in them, this is very memory consuming algorithm. One way to solve this problem is to use sparse matrices that save data in a less memory consuming way. We can do so because there's a lot of zero in our matrix, which allows sparse matrix to compress rows and columns.

Sklearn library has a feature_extraction.text module that we used for making a Bag of words model described above. As request text as documents, we used CountVectorizer class from module and it's function fit_transform. This class is used for converting text documents to a matrix of token count. Fit_transform function takes a list of strings (generaly, it takes an iterable object, but here we''l focus on our example) and returns document-term matrix.

Furthermore, it is important to notice that sklearn's implemented classifiers take values as frequencies, not occurances, so we must have had corrected this. For this we used TfifdTransformer class, another feature_extraction.text class. It's function fit_transform takes matrix and changes it's value to frequencies, than it downscales values according to their weight. Reason for this is to reduce the impact of tokens that occur very frequently in corpus and so are empirically less informative than features that occur in a small fraction of the training set. This downscaling is called tf-idf which stands for term frequency inverse document frequency. Tf-idf is computed by following formula:

$$\text{tf-idf}(d, t) = \text{tf}(t) * \text{idf}(d, t)$$

while idf(d,t) is computed as:

$$\text{idf}(d, t) = \log [\, n / \text{df}(d, t) \,] + 1$$

where n is the total number of document, and df(d,t) is the number of documents d that contain term t.

With this we have built feature extractor. Now we have taken multinomial naive Bayes classifier and used it on train set of data.

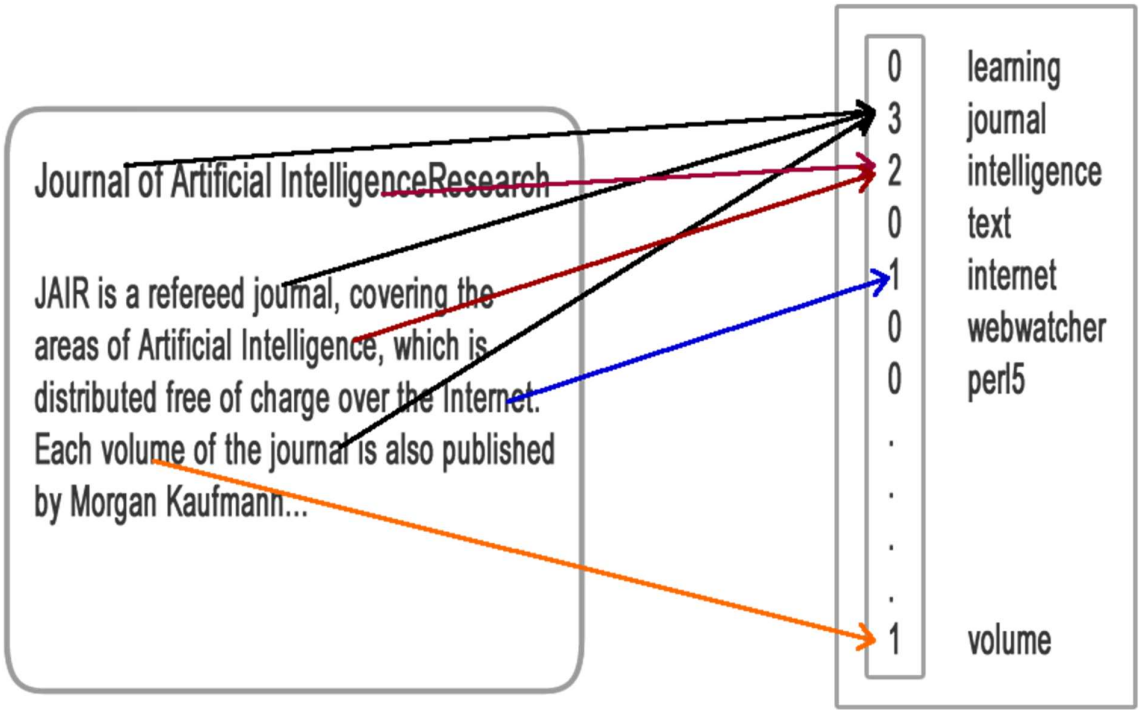
Journal of Artificial IntelligenceResearch
JAIR is a refereed journal, covering the areas of Artificial Intelligence, which is distributed free of charge over the Internet. Each volume of the journal is also published by Morgan Kaufmann...
0 learning
3 journal
2 intelligence
0 text
1 internet
0 webwatcher
0 perl5
1 volume

Figure 3

4. Results

4.1. NLTK NaiveBayesAllAttributes

Testing with NaiveBayesAllAttributes resulted in ~99% accuracy (Figure 4). That kind of precision is more than we hoped for and definitely worth looking into. When listing classifier's most informative features (features that have been most helpful in predicting labels of input), we see some awry-looking features. We can read that as: requests that have value of a feature called "*requester_user_flair*" equal to 'null' are 217.4 times more likely to have a label (outcome of a request) "False" than "True". Notice that classifier looked at exact number values of each attribute and classified all of the values discretely, not taking into account groups of requests with above or below average values.

```
NLTK NaiveBayesAllAttributes, 1:  0.9923664122137404
Most Informative Features
requester_user_flair = 'null'          False : True   =    217.4 : 1.0
requester_upvotes_minus_downvotes_at_retrieval = '1' False : True =21.5:1.0
requester_number_of_posts_on_raop_at_retrieval = '4' True : False =
19.6:1.0
requester_upvotes_plus_downvotes_at_retrieval = '2' False : True  =17.7:1.0
requester_number_of_comments_in_raop_at_retrieval = '18'True:False=15.1:1.0
```

Figure 4

Why, then, do we have such insanely "high" accuracy? It's an instance of *overfitting*: the classifier has so much (irrelevant) information to handle that it learns from noise and misinterprets it as useful information. However, that kind of trained classifier won't have good classifying skills beyond test set. Therefore, we set out to improve.

4.2. NLTK NaiveBayesSomeAttributes

Of all 32 attributes, not all are very informative. Certain features surely make more difference than others towards request outcome. Here we use some of the conclusions from the case study done by Althof et al. (and common sense) and deduce that attributes such as "*request_number_of_comments_at_retrieval*" (activity of users in the thread) might to a greater extent influence the request outcome. For second iteration, number of attributes is reduced to 10 hand-picked. Then we adjust them: pass through all the requests' selected attributes, calculate weighted average and standard deviation (using scipy.org's *numpy* Python library) for each sequence of features individually and assign following labels:

- "high", if the value is greater than (average plus 0.5 standard deviations)
- "low**"** , if the value is less than (average minus 0.5 standard deviations)
- "average", if the value falls 0.5 standard deviations within average

Once trained and tested, algorithm shows an accuracy up to 75% (Figure 5). Not only have we made a viable algorithm that doesn't fall prey to overfitting, but it has a decent success rate. A peek at its most informative features reveals how much differently the classifier acts now as opposed to the previous instance.

```
NLTK NaiveBayesSomeAttributes, 2:  0.7563123899001761
Most Informative Features
requester_number_of_posts_on_raop_at_request = '"high"'True : False=8.6:1.0
request_number_of_comments_at_retrieval = '"average"'False : True=4.5 : 1.0
requester_number_of_comments_in_raop_at_request='"high"'True:False=2.9:1.0
requester_number_of_comments_in_raop_at_request='"average"'True:False =
2.4:1.0
number_of_upvotes_of_request_at_retrieval = '"low"'False : True=2.2 : 1.0
```

Figure 5

4.3. NLTK NaiveBayesTextAnalysis

Since RAOP requests are textual and *redditors* base their conscious decision about pizza purchase on the request itself rather than analyzing metadata surrounding it, we implemented text analysis into the model. Case study by Althof et al. concluded that requests containing words from certain narratives such as *family, daughter, dollar, crave* have a higher chance of striking an emotional chord in the reader, therefore prompting them to buy requestor a pizza. Full list of keywords used in the algorithm, grouped by narratives they belong to, can be found in files provided with this project ("narratives").

First we import all the words from the narratives (keywords) provided by the study and put them in a list. Next, we pass through the requests, adding attributes for each keyword and marking them with "True" and "False", depending on whether text of a certain request contains that keyword. Note that, since there are 107 distinct keywords, as much new attributes are added to each request which complicates the analysis for the model. New classifier shows an accuracy of about 73% and shows us which keywords are most useful for the success of a request (Figure 6). As stated by the study, keywords such as drunk or beer suggesting irresponsible behavior are detrimental to the success, while the ones about user's financial situations seem to be beneficial (bucks, hire).

```
NLTK NaiveBayesTextAnalysis, 3:  0.733998825601879
Most Informative Features
          contains(hire) = True              True : False  =      5.1 : 1.0
         contains(bucks) = True              True : False  =      3.2 : 1.0
         contains(drunk) = True             False : True   =      2.9 : 1.0
         contains(hired) = True              True : False  =      2.7 : 1.0
          contains(beer) = True             False : True   =      2.7 : 1.0
```

Figure 6

Interesting side note: combining previous two algorithms by using most useful metadata attributes combined with text analysis improves model accuracy (~75% success) but not beyond that of NaiveBayesSomeAttributes, which accomplishes about same success rate with a quarter of analyzed attributes. We may find ourselves more lucky using different text analysis.

4.4. Scikit-learn

Testing with sklearn's multinomial naive Bayes resulted in accuracy of around 76 % (Figure 7). This is expected and not quite bad result. Our study of similar works on topic of text classification, along with some solutions for this very challenge had shown accuracy of succes around roughly 75% . It is also very close to success of NLTK NaiveBayesSomeAttributes method, which leads to conclusion this is indeed good basic result.

```
>>> ================================
>>>
The dataset contains 5671 samples.

(3970, 12586)
NBayes success:  0.761904761905
```

Figure 7

5. Method improvement

We've seen that success of our predictions is around 75%, regardless of method. Although good, this can be higher. One way to achieve this is by improving methods that we've already used. Following are some techniques for improvement.

Cross-validation - as mentioned, choosing training and test set is very important for accurate learning. When given limited amount of data, we need to be extra careful when choosing the size of those sets. If we make test set to small, our evaluation may not be accurate. If we make it to big, we'll have to little data to learn from. Cross-validation technique splits dataset into N subsets called folds. It proceeds to train a model for each fold, using all the data except those from fold as a training set. Afterwards we combine evaluation scores. Although each fold doesn't give very accurate evaluation, combination does because it's based on larger amount of data.

N-grams - N-grams are combination of n-sequential words. They are useful for modeling text beyond the Bag of words. Reason to do this is because sometimes a word itself can have different sentiment than it would have in combination with other words. For example, word 'good' would have positive sentiment, but this won't be very helpful if it's in the context like 'not good'. However, if we use bigrams (two word n-grams), model will probably learn that 'not good' has negative connotation. Because of expressions like 'not very good', it's even better to instead of bigrams use trigram or even 4-grams.

Grid-search - when working with sklearn, classifiers have lots of parameters we can choose from. We used default parameters, but by changing some of them, we may have received better result. Instead of trying out different parameters, we can run extensive search for best parameters on the grid within given values. First we choose which parameters we'd like to search and within which values, and then run them through grid-search function given in sklearn.

Besides improving existing method, we can also try different methods and classifiers, such as Support Vector Machine (SVM) and RandomForests methods. Since we used two different approaches on building the feature extractor, we concentrated on only one classifier (Naive Bayes), but it would be interesting to explore possibilities of others in the future. It is necessary to say that although RandomForests is very good method, and is indeed used for text classification, it's not good for high-dimensional sparse data, so it's not good for Bag of words.

6. Conclusion

We have taken Kaggle's AI challenge and tried to answer the question whether it is possible to predict altruism through requesting a free pizza. After noticing that the problem can be essentially brought down to text classification, we decided for machine learning technique. More precisely, we used supervised classification. This included data preparation, building feature extractor, and training a model using Naive Bayes classificator. All coding was done in Python 3. We also decided for two different approaches for building feature extractor.

One was using metadata that we've been given, along with appearance of some keywords in request text. This idea is partly based on paper given as a background for this competition. There were three varieties of this approach, depending on attributes given to feature extractor. Classificator was from NLTK api. Results were satisfying, for success of our best trained model, the one with selected attributes, was roughly 75%, which was expected.

Second approach was using Bag of words method. For this, as for the classificator, we used scikit-learn library. Results were again within expectations, around 76%.

We can conclude that both approaches seem equally precise, though there is much room for improvement for both of them. First one takes more data in the consideration, and with careful choice of attributes and cross-validation, can be very good. However, it is very specific for this precise problem, and cannot be easily generalized to other problems, nor it's easy to check it on new data since we also need the right metadata. On the other hand, the second approach is more typical one for given problem and can be used for various similar problems. But it is limited to solely text features, without using any additional information that is available (in the form of metadata), and some of that data have been proven quite relevant to the outcome.

7. References

https://www.reddit.com/r/Random_Acts_Of_Pizza/

https://www.kaggle.com/c/random-acts-of-pizza

Tim Althoff, Cristian Danescu-Niculescu-Mizil, Dan Jurafsky. How to Ask for a Favor: A Case Study on the Success of Altruistic Requests, Proceedings of ICWSM, 2014

Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit, Steven Bird, Ewan Klein, and Edward Loper

https://www.python.org/

http://www.nltk.org/

http://scikit-learn.org/

http://scipy.org/