Web Intelligence - Social Network Miniproject

Erik Sidelmann Jensen
`ejens11@student.aau.dk`

Lasse Vang Gravesen
`lgrave11@student.aau.dk`

Dennis Jakobsen
`djakob11@student.aau.dk`

1 Social Network Miniproject

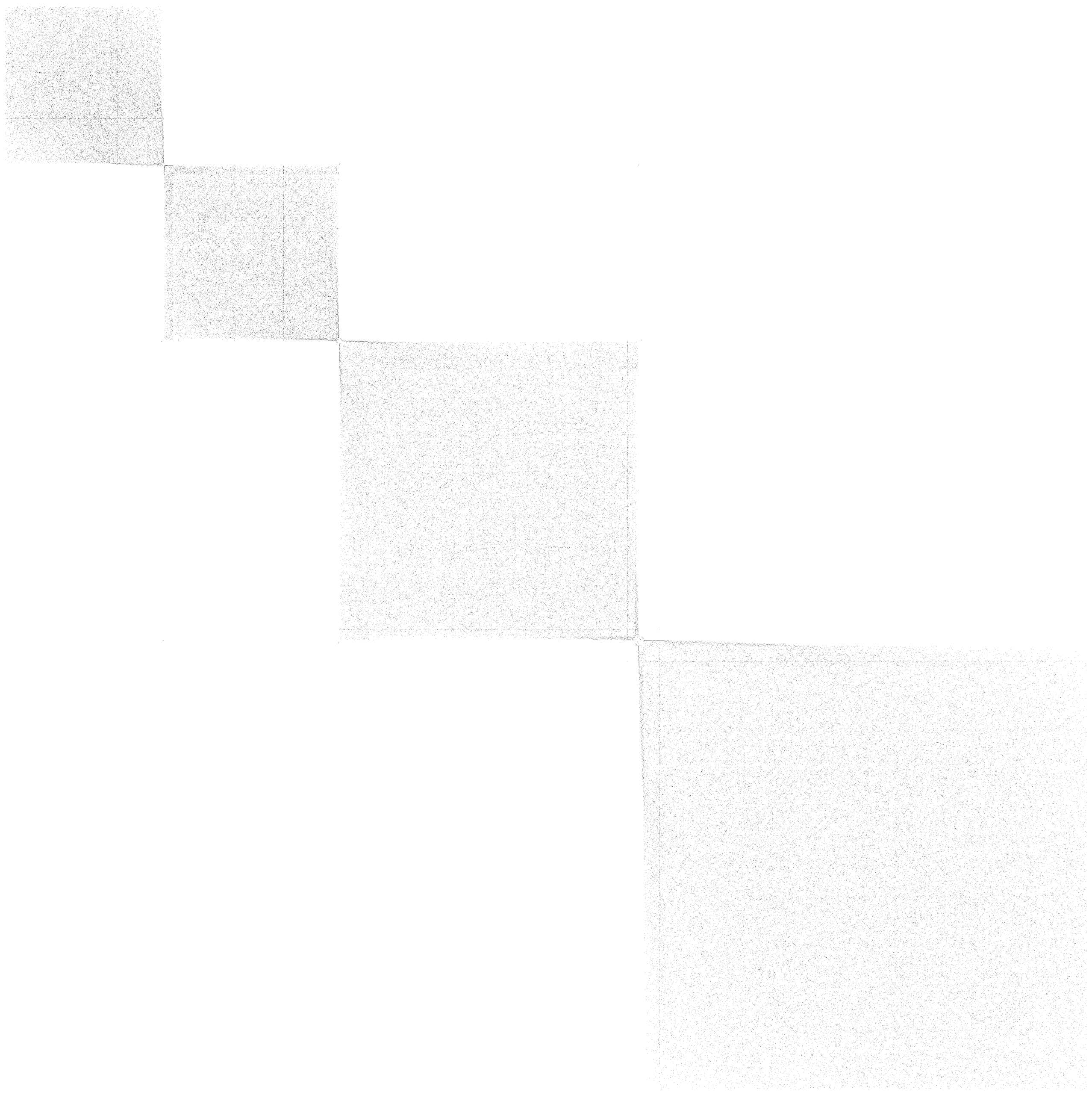
1.1 Network Communities Partitioning

Network Community partitioning was done by using spectral clustering. This was the procedure:

A User class was created to hold the basic information, the Username, the list of friends, the review, the summary, and a sortable eigen value. This is then used to load the users from the file.

From that list of users an adjacency matrix is created based on who they are friends with, called A. A row count diagonal matrix is created based on that adjacency matrix, called D. The laplacian is then $L = D - A$. The eigen decomposition is then found on L, and the second smallest eigenvector is found. Every value in the eigenvector is then assigned to a user object in the user list, after which the user list can be sorted and cut at the first-largest gap between eigen values. This is then run recursively on both communities until the gap becomes larger than 0.7 after which it stops and returns the list of communities, though we could have used modularity instead here we found that using a threshold value worked fine.

The results of this is 4 very segregated communities, on the first run-through the communities are obvious as can be seen in the figure below.



The spectral clustering algorithm was used because it is simple to implement, and works well.

1.2 Sentiment Analysis

Sentiment analysis of text was done using a Naive Bayes Classifier.

This was the procedure: First step is tokenising the test data and constructing features, and for sentiment analysis it is important to remember things like emoticons and negative words because those have an impact on the rest of the content. As such we took a sentiment sensitive tokeniser made in Python and converted it to C#, and added sensitivity to negative words, in that every token followed by a negative word would get ”_NEG” appended until the end of the sentence(punctuation or an emoticon).

The second step was parsing the learning data, which was done line by line, skipping the identifying text and taking the rest after having trimmed the result. Tokenising is also done in the parser, the tokeniser is called and these tokens are saved in a properties. Finally the known score of the review block is determined by the score. If the score is 3, the review is discarded. These reviews are then split into N partitions for easy learning and testing.

The third step is then constructing the Naive Bayes Classifier, which is essentially just counting. Specifically these properties of the learning data is determined through counting:

Estimation of the probability for all classes: $p(c)$. This is done through counting the amount of reviews available: N. The number of reviews with sentiment c: $N(c)$. We also have $|C|$, the amount of classes. Finally $p(c)$ can be estimated through

$$p(c) = \frac{N(c)+1}{N+|C|}$$

We also estimate probability for all words in corpus X and all possible sentiment classes in C: $p(x_i|c)$. To do this we need to count the number of times x_i appears across all reviews with sentiment c: $N(x_i, c)$. We also need $|X|$, the size of the vocabulary. Using this we can then calculate $p(x_i|c)$:

$$(x_i|c) = \frac{N(x_i, c)+1}{N(c)+|X|}$$

Then the score for a review can be determined through two tricks: First calculate score for the ”empty” review because most words from the vocabulary are not present in a given review(Trick 1). This is done using summation instead of product to avoid numerical instability(Trick 2).

$$s^*(empty, c) = (\sum_{i=1}^{n} p(\text{not } x_i|c)) + p(c)$$

The score can then be calculated using:

$$s(x, c) = s^*(empty, c) + (\sum_{j=1}^{k} \frac{p(x_j|c)}{p(\text{not } x_j|c)})$$

And that is how the Naive Bayes Classifier works, you first use the learning data to construct the probabilities and then you use those probabilities to determine the score for individual reviews.

To test the classifier we used all possible combinations of test and learning data, where test data was 1 partition and learning data was the rest. So over N partitions, that means we iteratively ran over the partitions, selected the first for testdata, the rest for learning data and then in the second iteration we select the second for testdata and so on.

Using this to improve the feature construction, we ended up with an accuracy rate of roughly 91% and an error rate of roughly 9% for each iteration. Though the classifier ended up being biased towards positive reviews in that it correctly classified them roughly 94% of the time, and for negative reviews it classified them correctly roughly 77% of the time.

1.3 Evaluation of Reviews

We then used the community partitions and the sentiment classifier to evaluate the reviews.

We did this by using the entirety of the learning data to teach the Naive Bayes Classifier. We then ran through the list of users and classified the reviews of users that had reviews/summaries using the Naive Bayes Classifier. Then we determine the communities using the spectral clustering. Then we ran through the list of users again, this time only focusing on the users that did not have any reviews or summaries, and based on their friends average review scores we determine if they are likely to purchase. A user named 'kyle' and friends in other communities count 10 times more than a regular friend, finally we determine if they are likely to purchase by checking if the average is more than or equal to 3, and if it is the user is likely to purchase otherwise not.

Our result in summary is then as follows: We classified 1347 user reviews out of 1644 as 'Positive', and 297 user reviews as 'Negative'. Of the users that did not have a review or a summary, we classified 2451 out of 2649 as likely to purchase and 197 as unlikely to purchase.